



Ingeniería de procesos de software ◀

Leonardo Bermón Angarita

Facultad de Administración
Sede Manizales



UNIVERSIDAD
NACIONAL
DE COLOMBIA

► **Leonardo Bermón Angarita**

Ph. D. en Ingeniería Informática de la Universidad Carlos III de Madrid (España).
Magíster en Informática e ingeniero de sistemas de la Universidad Industrial de Santander (Bucaramanga, Colombia). Actualmente es profesor asociado del Departamento de Informática y Computación de la Universidad Nacional de Colombia - Sede Manizales. Ha sido director de este departamento y del programa curricular de Administración de Sistemas Informáticos. Perteneció al Grupo de Trabajo Académico en Aplicaciones y Herramientas Web. Director del grupo de investigación Corporate Governance, Tecnología y Complejidad, y del Laboratorio de Investigación de Software. Sus líneas de investigación principales son la ingeniería del software, los procesos de software, el desarrollo de software web y el análisis y diseño orientado a objetos con UML.

ORCID: <https://orcid.org/0000-0002-6034-0483>

Ingeniería de procesos de software ◀

Ingeniería de procesos de software ◀

Leonardo Bermón Angarita



Bogotá, D. C., 2024

© Universidad Nacional de Colombia - Sede Manizales
Facultad de Administración, Departamento de Informática y Computación
© Leonardo Bermón Angarita

Primera edición, 2024
ISBN digital: 978-958-505-658-9 (PDF)
ISBN digital: 978-958-505-659-6 (epub)

Colección Ciencias de Gestión

Edición
Editorial Universidad Nacional de Colombia
direditorial@unal.edu.co
www.editorial.unal.edu.co

John Fredy Guzmán
Coordinación editorial

Eduardo Franco
Corrección de estilo

Henry Ramírez Fajardo
Diseño de la colección

Andrea Kratzer M.
Diagramación

Comité Editorial de la Facultad de Administración,
Sede Manizales:

Jorge Alberto Forero Santos
Director

Diana Minerva Idarraga Vallejo
Editora en jefe

Diego Andrés Álvarez Marín
Departamento de Ingeniería Civil

Luz Arabany Ramírez Castañeda
Departamento de Informática y Computación

Jaime Andrés Vieira Salazar
Departamento de Administración

Luis Alberto Carmona Sánchez
Departamento de Ciencias Humanas



Creative Commons Atribución-No Comercial-Sin obras derivadas 4.0 Internacional (CC BY-NC-ND 4.0)
<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Editado en Bogotá, D. C., Colombia.

Catalogación en la publicación Universidad Nacional de Colombia

Bermón Angarita, Leonardo, 1971-

Ingeniería de procesos de software / Leonardo Bermón Angarita. -- Primera edición. -- Bogotá : Editorial Universidad Nacional de Colombia ; Manizales : Universidad Nacional de Colombia. Facultad de Administración. Departamento de Informática y Computación, 2024.

1 recurso en línea (647, páginas) : ilustraciones (principalmente a color), diagramas. -- (Ciencias de gestión)

Incluye referencias bibliográficas al final de cada capítulo e índice temático
ISBN 978-958-505-658-9 (pdf). -- ISBN 978-958-505-659-6 (ePub)

1. Ingeniería de software -- Administración -- Problemas, ejercicios, etc. 2. Proceso de desarrollo de software 3. Desarrollo de software de aplicaciones 4. Medición de software 5. Software de entornos de trabajo 6. Lenguajes de modelado (Computación) 7. Ingeniería de software -- Ciclo de vida -- Normas técnicas 8. Proceso de mejora continua -- Normas técnicas 9. Sistemas informáticos -- Gestión - - Problemas, ejercicios, etc. I. Título II. Serie

CDD-23 005.1068 / 2024

► Contenido

Introducción	23

► Parte I. Introducción	31

Capítulo 1

Introducción a los procesos de software	33

1.1 Área del conocimiento del proceso de software	34

1.2 Historia del proceso de software	37

1.3 Definición y descripción del proceso de software	38

1.3.1 Definición de proceso de software	39

1.3.2 Descripción del proceso de software	42

1.3.3 Características y factores del proceso de software	44

1.4 Proceso vs. producto	46

1.5 Actividades del proceso de software	47

1.6 Gestión del proceso de software	49

1.7 Clasificación de procesos de software	50

1.8 Diversidad del proceso de software	52

Desafíos	54

Ejemplos	55

Resumen	56

Ejercicios	57

Referencias	58

Capítulo 2

Procesos tradicionales de software	61

2.1 Modelos lineales	63

2.1.1 Modelo en cascada	63

2.1.2 Modelo en V	66

2.2 Modelos iterativos	67

2.2.1 Desarrollo incremental	67

2.3 Modelos evolutivos	71

2.3.1 Desarrollo de prototipos	71

2.3.2 Modelo en espiral	74

2.3.3 Modelo espiral <i>win-win</i>	76
2.3.4 Desarrollo basado en componentes	77
2.3.5 Desarrollo rápido de aplicaciones	79
2.4 Selección de modelos de ciclo de vida	80
Desafíos	83
Ejemplos	84
Resumen	85
Ejercicios	86
Referencias	87
Capítulo 3	
Procesos ágiles	91
3.1 Conceptos sobre procesos ágiles	93
3.1.1 Historia de los procesos ágiles	96
3.1.2 Ventajas y beneficios de los procesos ágiles	99
3.1.3 Equipos ágiles	99
3.1.4 Manifiesto ágil y principios ágiles	101
3.1.5 Grado de utilización de procesos ágiles	107
3.2 eXtreme Programming (XP)	108
3.2.1 Valores de XP	108
3.2.2 Prácticas de XP	109
3.2.3 Proceso de XP	112
3.2.4 Ventajas y desventajas de XP	115
3.3 <i>Scrum</i>	116
3.3.1 Roles	116
3.3.2 Artefactos	119
3.3.3 Eventos	123
3.3.4 Otros conceptos	129
3.3.5 Beneficios, desventajas y restricciones	132
3.4 Otros procesos ágiles	134
3.4.1 Lean Software Development (LSD)	134
3.4.2 Métodos <i>crystal</i>	136
3.4.3 Dynamic Systems Development Method (DSDM)	138
3.4.4 Feature-Driven Development (FDD)	141

3.4.5 Adaptive Software Development (ASD)	142
3.4.6 Test Driven Development (TDD)	144
3.5 Comparación entre procesos ágiles y tradicionales	145
3.5.1 Comparación entre procesos ágiles	145
3.5.2 Comparación entre procesos ágiles y procesos tradicionales	149
3.6 Escalamiento ágil	149
3.6.1 <i>Scrum</i> de <i>scrums</i>	151
3.6.2 Marco de referencia ágil escalado	152
3.6.3 Entrega disciplinada ágil	154
3.6.4 <i>Scrum</i> de gran escala	156
3.6.5 Comparativa de métodos de escalamiento ágil	158
Desafíos	159
Ejemplos	162
Resumen	164
Ejercicios	165
Referencias	167
Capítulo 4	
Estándares y procesos industriales de software	173
4.1 Conceptos sobre estándares	174
4.2 Estándares del proceso de software	175
4.2.1 Norma ISO 9000	177
4.2.2 Norma ECSS	182
4.2.3 Norma ISO/IEC/IEEE 12207	184
4.2.4 Norma DOD-STD-2167A	186
4.2.5 Norma IEEE STD 1074	188
4.2.6 Norma ISO/IEC 15288	190
4.3 Procesos industriales	193
4.3.1 Métrica v.3	193
4.3.2 El proceso unificado	196
4.3.3 EssUP	202
4.3.4 SEMAT	203
4.3.5 SWEBOK	207
Desafíos	210

Ejemplos	210
Resumen	213
Ejercicios	215
Referencias	216
 ► Parte II. Ciclo de vida del proceso de software	 221
 Capítulo 5	
Lenguajes de modelado de procesos de software	225
5.1 Conceptos sobre modelos	225
5.2 Modelos de procesos de software	227
5.3 Lenguajes de modelado de procesos de software	230
5.3.1 Software and Systems Process Engineering Metamodel	231
5.3.2 Business Process Model and Notation (BPMN)	238
5.3.3 Diagramas de actividad UML	248
5.3.4 IDEF0	251
5.3.5 Event-Driven Process Chain (EPC)	254
5.3.6 Redes de Petri	256
5.4 Comparación de lenguajes de modelado de procesos de software	257
Desafíos	259
Ejemplos	261
Resumen	263
Ejercicios	264
Referencias	265
 Capítulo 6	
Métricas de procesos de software	269
6.1 Conceptos sobre medición	270
6.2 Ontología de medición	276
6.3 Métricas de software	284
6.4 Paradigma GQM	286
6.5 Medición del proceso	287
6.5.1 Rendimiento	287
6.5.2 Estabilidad	288
6.5.3 Conformidad	289

6.5.4 Capacidad	290
6.5.5 Mejora e inversión	290
6.6 Estándar de medición del proceso ISO/IEC/IEEE 15939	291
6.7 Métricas de proceso de software	294
6.8 Métricas de modelos de procesos SPEM	298
6.9 Técnicas de medición del proceso de software	301
Desafíos	301
Ejemplos	302
Resumen	305
Ejercicios	306
Referencias	307
Capítulo 7	
Control de procesos de software	309
7.1 Control estadístico del proceso	310
7.2 Gráficos de control	311
7.2.1 Variación	311
7.2.2 Estabilidad	313
7.3 Tipos de gráficos de control	314
7.3.1 Gráficos de control de Shewhart	314
7.3.2 Gráficos de control para variables	317
7.3.3 Gráficos de control para atributos	322
7.3.4 Detección de inestabilidades y procesos fuera de control	326
7.4 Seguimiento del proceso	328
7.4.1 Seguimiento binario	330
7.4.2 Valor ganado	332
Desafíos	336
Ejemplos	337
Resumen	341
Ejercicios	342
Referencias	343
Capítulo 8	
Mejora del proceso de software	345
8.1 Conceptos sobre mejora del proceso de software	346

8.2 Ciclo de mejora del proceso de software	348
8.2.1 Identificación de mejoras	350
8.2.2 Planificación de mejoras	351
8.2.3 Implementación de mejoras	351
8.2.4 Planes piloto/refinación	351
8.2.5 Despliegue	352
8.2.6 Repetir el ciclo	352
8.3 Modelos de mejora de procesos de software	352
8.3.1 Modelo QIP	353
8.3.2 Modelo IDEAL	355
8.3.3 Modelo CMMI	357
8.3.4 Modelo TSP	365
8.3.5 Modelo PSP	368
8.3.6 Modelo Six Sigma	372
8.3.7 Modelo BSC	376
8.3.8 ISO 33000	378
8.3.9 Modelo TickITplus	381
8.4 Metodologías de evaluación del proceso de mejora	383
8.4.1 Modelo SCAMPI	384
8.5 Mejora de procesos ágiles	388
8.6 Herramientas de mejora del proceso	390
8.6.1 Hoja de verificación	391
8.6.2 Histogramas	392
8.6.3 Diagrama de Pareto	393
8.6.4 Diagrama de dispersión	394
8.6.5 Diagrama de causa y efecto	395
8.6.6 Diagrama de flujo	396
Desafíos	397
Ejemplos	398
Resumen	400
Ejercicios	401
Referencias	402

► Parte III. Temas avanzados del proceso de software 407

Capítulo 9

Infraestructura del proceso de software 409

9.1 Despliegue del proceso de software 410

9.2 Conceptos sobre infraestructura del proceso de software 414

9.3 Infraestructura organizacional y de gestión 416

9.3.1 Grupo de dirección de gestión 417

9.3.2 Gestor de proyectos 417

9.3.3 Software Engineering Process Group (SEPG) 417

9.3.4 Grupo de Process Quality Assurance (PQA) 418

9.3.5 Grupo de analistas de métricas 418

9.3.6 Grupo de formación 418

9.4 Infraestructura tecnológica 419

9.4.1 Ambientes de ingeniería de software centrados en procesos 420

9.4.2 Manuales del proceso 421

9.4.3 Guías electrónicas de procesos 422

9.4.4 Bibliotecas de activos de procesos 423

9.4.5 Wikis 426

9.5 Formación en procesos de software 427

9.6 Retorno de la inversión en SPI 429

Desafíos 432

Ejemplos 433

Resumen 434

Ejercicios 435

Referencias 435

Capítulo 10

Procesos de software para pymes 439

10.1 Pymes desarrolladoras de software 440

10.2 SPI en pymes de software 442

10.3 Estándares internacionales para SPI en pymes 444

10.4 Infraestructuras para SPI en pymes 449

10.4.1 Infraestructura MoProSoft 449

10.4.2 Infraestructura Competisoft 452

10.4.3 Infraestructura MPS.BR	454
10.4.4 Infraestructura ITmark	457
10.4.5 Infraestructura Tutelkan	458
10.5 Modelos y métodos para SPI en pymes	461
10.5.1 Modelo OWPL	461
10.5.2 Modelo MARES	463
10.5.3 Modelo EPA	465
10.5.4 Modelo Adept	467
10.5.5 Modelo Impact	468
10.5.6 Modelo MESOPYME	470
10.5.7 Modelo ASPE/MSD	472
10.5.8 Modelo iFlap	473
10.5.9 Modelo Processus	476
10.5.10 Matriz del SPM	478
10.5.11 Modelo RAPID	481
10.5.12 Modelo XPMM	482
10.5.13 Otros modelos	485
10.6 Técnicas para SPI en pymes	486
Desafíos	489
Ejemplos	490
Resumen	492
Ejercicios	493
Referencias	493
Capítulo 11	
Reúso del proceso de software	501
11.1 Conceptos sobre reúso de software	502
11.1.1 Reúso de software	502
11.1.2 Activos de software	503
11.1.3 Repositorios de reúso	505
11.1.4 Proceso de reúso	508
11.2 Reúso del proceso de software	511
11.3 Estándar de reúso	513
11.4 Componentes del proceso de software	516
11.4.1 Componentes de proceso	516

11.4.2 Líneas de procesos de software	518
11.4.3 Arquitectura del proceso de software	521
11.4.4 Patrones de proceso	523
11.5 Métricas de reúso	527
11.6 Adaptación del proceso de software	529
11.7 Gestión del conocimiento para el reúso del proceso de software	537
Desafíos	541
Ejemplos	542
Resumen	544
Ejercicios	545
Referencias	546

► Parte IV. Aplicaciones 550

Capítulo 12

Procesos de software para aplicaciones específicas 553

12.1 Desarrollo de aplicaciones de reconocimiento de patrones	553
12.2 Desarrollo de aplicaciones de internet de las cosas	556
12.3 Desarrollo de aplicaciones <i>blockchain</i>	559
12.4 Desarrollo de aplicaciones de software sostenibles	561
12.5 Desarrollo de aplicaciones de <i>machine learning</i>	563
12.6 Desarrollo de aplicaciones para automatización industrial	566
12.7 Desarrollo de aplicaciones para minería de datos	568
Desafíos	570
Ejemplos	570
Resumen	572
Ejercicios	573
Referencias	574

Capítulo 13

Casos de éxito 577

13.1 Caso de estudio: proyecto Sentinel del FBI	578
13.2 Caso de estudio: Intel	580
13.3 Caso de estudio: Google	582
13.4 Caso de estudio: BBC	584

13.5 Caso de estudio: BBVA	585
13.6 Caso de estudio: Spotify	587
13.7 Caso de estudio: NASA	589
13.8 Caso de estudio: Apple	591
13.9 Caso de estudio: Amazon	593
Desafíos	595
Ejemplos	595
Resumen	597
Ejercicios	598
Referencias	598
Capítulo 14	
Tendencias actuales de investigación del proceso de software	601
14.1 Tendencias de investigación en procesos de software	602
14.2 Minería del proceso de software	607
14.3 Aprendizaje automático y aprendizaje profundo	609
14.3.1 Aprendizaje automático	610
14.3.2 Aprendizaje profundo	612
14.4 Nuevos modelos de procesos de software	614
14.4.1 Desarrollo global de software	615
14.4.2 Ingeniería dirigida por modelos y arquitectura dirigida por modelos	617
14.4.3 DevOps	620
14.4.4 Desarrollo móvil y en la nube	623
Desafíos	626
Ejemplos	628
Resumen	630
Ejercicios	631
Referencias	632
Reflexión final	637
Índice analítico	643

► Lista de siglas y acrónimos

AE: Auto-Encoder

ALPHA: Abstract-Level Progress Health Attribute

ALPS: Arquitectura de Línea de Proceso Software

ANOVA: Analysis of Variance

ART: Agile Release Train

ASD: Adaptive Software Development

ASI: Análisis del Sistema de Información

ASPE/MS: Approach for Software Process Establishment
in Micro and Small Companies

BPL: Base Process Library

BPMN: Business Process Modeling Notation

BSC: Balance Score Card

CAL: Aseguramiento de la Calidad

CAR: Causal Analysis and Resolution

CASE: Computer Aided Software Engineering

CEP: Control Estadístico del Proceso

CIM: Computing Independent Model

CM: Configuration Management

CMI: Cuadro de Mando Integral

CMMI: Capability Maturity Model Integration

CNN: Convolutional Neural Network

COTS: Commercial Off-The-Shelf

CSI: Construcción del Sistema de Información

CSPT: Centre for Software Process Technologies

CSS: Cascading Style Sheets

DAD: Disciplined Agile Delivery

DAR: Decision Analysis and Resolution

DBN: Deep Belief Network

DMAIC: Define Measure Analyze Improve Control

DNN: Deep Neural Network

DSDM: Dynamic System Development Method

DSI: Diseño del Sistema de Información

ECSS: European Cooperation for Space Standardization

EPA: Express Process Appraisal

EPC: Event-Driven Process Chain

EPML: EPC Markup Language

EssUP: Essential Unified Process

EvalProSoft: Modelo de Evaluación de Procesos Software
de la Industria Mexicana

EVS: Estudio de Viabilidad del Sistema

FDD: Feature Driven Development

GAN: Generative Adversarial Networks

GC: Gestión de Configuración

GP: Gestión de Proyectos

GQM: Goal Question Metric

GRU: Gated Recurrent Unit

HIP: Hardening, Innovation, Planning

HMR: Hybrid Method Engineering

HTML: Hyper Text Markup Language

IAS: Implantación y Aceptación del Sistema

ICOM: Input, Control, Output, Mechanism

IDEAL: Diagnosing, Establishing, Acting & Learning

IDEF: Integration Definition for Function Modeling

IEC: International Electrotechnical Commission

IEEE: Institute of Electrical and Electronics Engineers

iFLAP: Improvement Framework Utilizing Light Weight Assessment and Improvement Planning

IPM: Integrated Project Management

IPRC: International Process Research Consortium

ISO: International Standardization Organization

KLOC: Kilo Lines of Code

KM: Knowledge Management

KPI: Key Performance Indicator

LCL: Lower Control Limit

LeSS: Large Scale Scrum

LOC: Lines of Code

LPS: Línea de Proceso Software

LSTM: Long Short Term Memory

MA: Measurement and Analysis

MARES: Methodology for Software Process Assessment in Small Software Companies

MASAM: Mobile Application Software Agile Methodology

MDA: Model Driven Architecture

MDE: Model-Driven Engineering

MOF: Meta-Object Facility

MoProSoft: Modelo de Procesos Software de la Industria Mexicana

MoSCoW: Must Have, Should Have, Could Have, Want to Have

MPS.BR: Mejora de Proceso del Software Brasileño

MSG: Management Steering Group

MSI: Mantenimiento de Sistema de Información

OCL: Object Constraint Language

OMG: Object Management Group

OPD: Organizational Process Definition

OpenUP: Open Unified Process

OPF: Organizational Process Focus

OPP: Organizational Process Performance

OT: Organizational Training

PAL: Process Asset Library

PDCA: Planning, Do, Check, Act

PI: Product Integration

PIM: Platform Independent Model

PMC: Project Monitoring and Control

PML: Process Modeling Language

PP: Project Planning

PPQA: Process and Product Quality Assurance

PQA: Process Quality Assurance

PROBE: Proxy-Based Estimating

PSEE: Process-centered Software Engineering Environment

PSI: Planificación de Sistemas de Informacion

PSM: Platform Specific Model

PSP: Personal Software Process

pyme: pequeña y mediana empresa

QFD: Quality Function Deployment

QIP: Quality Improvement Paradigm

QPM: Quantitative Project Management

RAPID: Rapid Assessments for Process Improvement for software
Development

RD: Requirements Development

REQM: Requirements Management

RM: Rango Móvil

RNN: Recurrent Neural Network

ROI: Return of Inversion

RSKM: Risk Management

RUP: Rational Unified Process

SaaS: Software as a Service

SADT: Structured Analysis and Design Technique

SAFe: Scaled Agile Framework

SAM: Supplier Agreement Management

SCAMPI: Standard CMMI Appraisal Method for Process Improvement

SDK: Software Development Kit

SEG: Seguridad

SEI: Software Engineering

SEMAT: Software Engineering Method and Technique

SEPG: Software Engineering Process Group

SOA: Service Oriented Architecture

Softex: Asociación para la Promoción de la Excelencia Brasileña en Software

SPEM: Software & Systems Process Engineering Metamodel

SPI: Software Process Improvement

SPICE: Software Process Improvement Capability Determination

SPM: Software Process Matrix

SVM: Support Vector Machines

SWEBOK: Software Engineering Book of Knowledge

TDD: Test Driven Development

TI: Tecnología de la Información

TQM: Total Quality Management

TS: Technical Solution

TSP: Team Software Process

UCL: Upper Control Limit

UML: Unified Modeling Language

VAL: Validation

VER: Verification

VPN: Valor Presente Neto

VSE: Very Small Entity

XP: eXtreme Programming

XPMM: eXtreme Programming Maturity Model

► Introducción

Este libro explica qué es la ingeniería de procesos de software abordando los aspectos que se deben conocer para aplicar un proceso exitoso en el desarrollo de productos de software.

► Industria de software

La industria de software independiente surgió en la década de 1960 con una creciente demanda de programas y servicios informáticos. Los tipos de software desarrollados actualmente varían mucho, tomándose el concepto de software tanto como un producto (empaquetado, predefinido y ofrecido a un mercado masivo, por ejemplo, sistemas operativos, paquetes de oficina, videojuegos, etc.) o un servicio (personalizado y desarrollado para un cliente específico). Su modo de entrega ha cambiado en los últimos años pasando de licencias implementadas en las propias instalaciones del cliente al acceso en línea de servicios alojados en servidores de proveedores informáticos.

La creciente demanda de software y servicios de tecnología de la información (TI) ha causado una demanda excesiva de habilidades y competencias en los mercados y ha generado oportunidades en países emergentes cuya industria de software ha crecido rápidamente. Actualmente, esta se puede considerar un mercado global en el que se requiere poseer amplios conocimientos y destrezas por parte del personal, conocimientos en numerosas tecnologías que abarcan amplios dominios de negocios y una motivación constante por la innovación y mejora continua.

► Ingeniería de procesos de software

Para afrontar los desafíos del desarrollo de software, se ha consolidado el área de conocimiento de la ingeniería de software, una disciplina que abarca el estudio sistemático del desarrollo y mantenimiento de software de una manera profesional. La ingeniería de software hace énfasis en abandonar técnicas artesanales de desarrollo que se enfocan principalmente en la

codificación de programas y reemplazarlas por un enfoque riguroso basado en principios de ingeniería que cubran todas las fases del proceso desde la especificación de requisitos de software hasta la entrega final al usuario.

Dentro de la ingeniería de software, la investigación se ha interesado en mejorar los procesos de software como un mecanismo para definir y documentar el conocimiento necesario para realizar proyectos de desarrollo de software.

La ingeniería de procesos de software, en forma resumida y concisa proceso de software, apunta a estudiar estas necesidades surgidas de la aplicación de un enfoque orientado a procesos. El proceso de software se puede considerar una serie de actividades, procedimientos, roles y artefactos relacionados para elaborar un producto de software. Esta nueva área de conocimiento de la ingeniería de software ha sido reconocida formalmente, al estar incluida en el cuerpo de conocimiento de ingeniería de software (Software Engineering Body of Knowledge [SWEBOK] v3.0).

Un buen proceso de software ayuda a integrar las diferentes actividades que lo conforman con el objetivo común de entregar productos de alta calidad a tiempo y dentro del presupuesto establecido.

Para desarrollar un software con una alta calidad de manera eficiente, las empresas deben alinear el desarrollo técnico y la gestión del proceso con sus objetivos y procesos de negocio. Para lograr esto, se necesitan ciertas pautas para el desarrollo, definiendo actividades, resultados y roles necesarios, los cuales son de interés fundamental y prioritario para el proceso de software.

► Origen de este libro

Como investigador en ingeniería de software, el autor ha tenido amplios intereses en diversos temas de esta área de la informática. Los modelos de ciclo de vida del software siempre son explicados en los cursos universitarios, con énfasis en que es mejor conocer y aplicar algún modelo que ninguno. Además, en las últimas décadas, el surgimiento y la consolidación de los procesos ágiles ha permitido que se fortalezcan aún más los practicantes e interesados en conocer, ampliar e investigar en el área de procesos de software. Todos estos esfuerzos se han reflejado en un crecimiento vertiginoso de la investigación en procesos de software con el surgimiento y

crecimiento de numerosas publicaciones, eventos académicos y libros de texto en ingeniería de procesos de software.

Las actividades de investigación del autor durante y después de su doctorado han estado enfocadas en investigar nuevas perspectivas, enfoques y aplicaciones del proceso de software abordando el tema desde múltiples aristas, como la reutilización del conocimiento, la perspectiva de agilidad y, actualmente, con la incorporación de técnicas de aprendizaje automático y aprendizaje profundo en etapas específicas del proceso.

Este libro de ingeniería de procesos de software es un intento por ofrecer una fuente de conocimiento que abarque los principios, las técnicas, los estándares y las contribuciones realizadas en el área de procesos de software. De tal manera que los practicantes, docentes, investigadores e interesados que deseen conocer y profundizar en procesos de software encuentren una fuente de consulta y punto de referencia inicial que pueda servir como guía a un viaje más extenso a tierras ignotas del proceso de software.

Para la realización de este libro, se ha consultado y extraído información de numerosas referencias bibliográficas que se incluyen al final de cada capítulo. Estas referencias han sido tomadas y consultadas a partir de búsquedas en diversas plataformas de búsqueda de información bibliográfica, principalmente Springer Link, **IEEE** Xplore, Scopus y Google Académico. Los libros y artículos encontrados abarcan trabajos publicados entre 2010 y 2020, aunque el libro incluye referencias más antiguas para explicar el origen y la evolución de los conceptos estudiados.

► Objetivos

Este libro tiene como propósito los siguientes objetivos:

- Servir como libro de texto que oriente a profesores, estudiantes, practicantes e interesados de la ingeniería de software en conocer aspectos relacionados con el proceso de software.
- Lograr que los lectores comprendan la definición, el modelado, la medición, la ejecución, el control y la mejora de los procesos de software.
- Entender el estado actual de la investigación en el área de los procesos de software para establecer nuevas rutas y preguntas de investigación que al ser resueltas fortalezcan los conocimientos del área.

- Ofrecer a los lectores los conocimientos necesarios para aplicar con éxito un enfoque de procesos en organizaciones desarrolladoras de software o en proyectos de desarrollo de software.

► **Contenido**

Un panorama global de la estructura del libro se puede observar en la figura A.

Parte IV Aplicaciones	12. Procesos software para aplicaciones específicas	13. Casos de éxito	14. Tendencias futuras de investigación del proceso software
Parte III Temas avanzados	9. Infraestructura para el proceso software	10. Proceso software para pymes	11. Reúso del proceso software
Parte II Ciclo de vida del proceso software	7. Control del proceso software	8. Mejora del proceso software	
	5. Lenguajes de modelo del proceso	6. Métricas del proceso software	
Parte I Introducción	4. Estándares del proceso software		
	2. Procesos tradicionales de desarrollo de software	3. Procesos ágiles de desarrollo de software	
	1. Introducción a los procesos software		

Figura A. Estructura global del libro

Fuente: elaboración propia.

El libro está organizado en cuatro partes con un total de catorce capítulos. La primera parte presenta una introducción a los principales conceptos, modelos y estándares relacionados con el proceso de software. Esta primera parte está organizada en cuatro capítulos.

El primer capítulo es una introducción a la ingeniería de procesos de software con énfasis en la definición del concepto de proceso de software, sus componentes, características, factores, clasificación y áreas

relacionadas. Con este capítulo, se obtiene un panorama global del proceso de software y se presentan los fundamentos esenciales para comprender los subsiguientes capítulos del libro.

El segundo capítulo presenta un conjunto de modelos de procesos tradicionales de desarrollo de software clasificados en modelos lineales, iterativos y evolutivos. Además, se muestra un conjunto de criterios que ayudan a seleccionar un determinado proceso a partir de las características del proyecto de desarrollo de software a realizar.

El tercer capítulo describe la contraparte de los procesos tradicionales: los procesos ágiles, los cuales son actualmente muy conocidos y utilizados a nivel mundial. El capítulo presenta los principales conceptos para entender la agilidad relacionada con el proceso de software y describe ampliamente los procesos *scrum* y programación extrema, ampliamente utilizados en la industria. Además, un conjunto de procesos ágiles adicionales es presentado en forma resumida. También se presenta una tabla comparativa de los procesos ágiles enunciados de acuerdo con diversas características compartidas entre ellos. Por último, se describen modelos de escalamiento ágil que permiten adaptar los procesos ágiles a entornos empresariales complejos o de mayor tamaño.

El cuarto capítulo expone diferentes estándares mundiales relacionados con el proceso de software. Estos estándares han sido propuestos por diferentes organismos de certificación ampliamente reconocidos, como la International Organization for Standardization (ISO), el Institute of Electrical and Electronics Engineers (IEEE) o el Software Engineering Institute (SEI), entre otros. También se describen algunos procesos industriales fuertemente utilizados por las compañías de software a nivel mundial.

La segunda parte del libro consta de cuatro capítulos relacionados con el ciclo de vida del proceso de software.

El quinto capítulo presenta el concepto de modelado de proceso de software, el cual es definido a través de diferentes lenguajes de modelado de procesos. Entre estos diferentes lenguajes, se describen Software Process Engineering Metamodel (SPEM), Business Process Model and Notation (BPMN), diagramas de actividad Unified Modeling Language (UML) e Integration Definition for Function Modeling (IDEF-0). Por último, el capítulo realiza una comparación entre los conceptos y las notaciones de cada uno de los lenguajes de modelado de procesos de software presentados.

El sexto capítulo expone el concepto de medición del proceso de software. Para ello, se presenta inicialmente la noción de métrica de software

y su aplicación al contexto de procesos. Además, se describen métodos asociados a la definición de un programa de métricas de proceso a nivel organizacional, como Goal-Question-Metric (GQM) y el estándar ISO/IEC/IEEE 15939.

El séptimo capítulo especifica qué y cómo se realiza el control del proceso de software. Para ello, hace énfasis en el control estadístico del proceso y se presenta un conjunto de gráficos de control que permiten evaluar el proceso de software que se está ejecutando. Por último, describe diferentes técnicas relacionadas con la realización del seguimiento de un proyecto de desarrollo de software basándose en diferentes métodos y métricas.

El octavo capítulo estudia en profundidad los conceptos relacionados con la mejora del proceso de software. Describe el ciclo de mejora del proceso de software y presenta un conjunto de modelos de mejora del proceso desarrollados y utilizados por la comunidad internacional. Cada modelo de proceso generalmente tiene asociado un modelo que permite evaluar el grado de cumplimiento del proceso o de sus áreas de proceso (denominado nivel de capacidad/madurez). Para ello, el capítulo presenta, además, un conjunto de modelos de evaluación del proceso.

La tercera parte está conformada por tres capítulos que describen temas avanzados del proceso de software.

El noveno capítulo trata sobre la infraestructura necesaria para desplegar los procesos de software definidos en las organizaciones desarrolladoras de software. Describe la infraestructura organizacional y de gestión, tecnológica y de formación para implementar los procesos. Además, incluye una serie de métricas para evaluar el retorno de la inversión de estos recursos.

El décimo capítulo presenta un conjunto de estándares, infraestructuras, modelos, métodos y técnicas orientadas a implementar programas de mejora del proceso de software en el contexto de pymes desarrolladoras de software.

El undécimo capítulo abarca los conceptos de reutilización de software, pero aplicados al dominio del proceso de software. Para ello, hace énfasis en los componentes del proceso, las líneas de procesos y la infraestructura para el reúso basada en arquitecturas, patrones de proceso y métricas de reúso. Además, se describe el estándar IEEE 1517:2010 orientado al reúso en el ciclo de vida de software. Por último, presenta los conceptos de adaptación del proceso (que consiste en ajustar el proceso estándar definido a un proceso específico de proyecto) y la aplicación de conceptos

de gestión del conocimiento como un medio para potenciar la mejora del proceso de software.

La cuarta parte consta de tres capítulos orientados a presentar modelos de procesos en aplicaciones específicas, casos prácticos de procesos de software y tendencias futuras de investigación en el área.

El duodécimo capítulo describe diversos modelos de procesos definidos para aplicaciones de software específicas. Estas aplicaciones requieren modelos de procesos versátiles que han sido propuestos en aplicaciones concretas, pero ampliamente utilizadas en el ámbito académico y comercial, como reconocimiento de patrones, internet de las cosas, *blockchain*, software sostenible, *machine learning*, automatización industrial y minería de datos.

El decimotercer capítulo presenta diversos casos de estudio sobre la definición de un proceso de software o la transformación de procesos de software realizados en empresas líderes del sector de software a nivel mundial. Se resaltan y detallan las situaciones de algunas empresas reconocidas, como Intel, Google, Apple, Amazon, Spotify, entre otras.

El último capítulo describe las principales tendencias de investigación delineadas actualmente, con el objetivo de que sirvan para definir trabajos que ayuden a contribuir al acervo teórico y práctico de la investigación en procesos de software.

Cabe resaltar que algunas figuras incluidas en este libro son elaboraciones propias y otras son tomadas y adaptadas de fuentes bibliográficas debidamente referenciadas en el pie de las figuras. Estas fueron desarrolladas en diferentes herramientas software, como Microsoft PowerPoint, StarUML, ConceptDraw y Enterprise Architect.

► Agradecimientos

El autor agradece a los integrantes del grupo de investigación **SEL** de la Universidad Carlos III de Madrid, en especial a Antonio Amescua Seco, Javier García Guzmán y Maribel Sánchez Segura, por inspirarlo a escribir este libro, el cual fue una idea que surgió durante sus estudios de doctorado, que tuvo presente durante muchos años y que al fin se logró cristalizar mucho tiempo después, a partir de numerosos esfuerzos de investigación, lecturas, dirección de trabajos de grado y cursos de ingeniería de software dirigidos y desarrollados.

También el autor agradece a los profesores del Departamento de Informática y Computación de la Universidad Nacional de Colombia Sede Manizales por su constante apoyo en su formación docente e investigadora. Finalmente, el autor quiere dedicar esta obra a su esposa Lyda Jovanna, cuyo cariño y amor sin límites lo han guiado y motivado todos los días.



► Parte I. Introducción

El objetivo de esta primera parte es ofrecer una introducción a los principales conceptos relacionados con la ingeniería del proceso de software. Aborda conceptos fundamentales sobre el proceso de software, describe dos grandes tipologías de procesos de software clasificándolos en procesos tradicionales y procesos ágiles de desarrollo de software, y presenta un panorama global de los principales estándares y procesos industriales de desarrollo de software.

► **Capítulo 1. Introducción a los procesos de software**

Este capítulo está orientado a que el lector comprenda el concepto de proceso software y su importancia para la ejecución exitosa de proyectos de desarrollo de software. Para facilitar su comprensión, se presentarán las características, los elementos, las actividades y los diversos esquemas de clasificación del proceso de software. Además, es fundamental entender la relación existente entre el proceso y el producto de software. Con la lectura de este capítulo, el lector logrará entender cuál es el propósito de la ingeniería del proceso de software y obtendrá un cuerpo de conocimiento básico que le permitirá profundizar en temas más avanzados y concretos de esta disciplina.

► **Capítulo 2. Procesos tradicionales de software**

Este capítulo aborda una primera clasificación de procesos de software denominada procesos tradicionales, también llamados procesos descriptivos u orientados al plan. Estos procesos fueron los primeros que se originaron para resolver el problema de afrontar con éxito el desarrollo de

software cumpliendo las restricciones estipuladas de tiempo y costo. Para entender los procesos tradicionales, se estudiará el concepto de modelo de ciclo de vida de desarrollo de software. También se presentarán tres tipos genéricos de modelos de ciclo de vida: lineales, iterativos y evolutivos. Para cada uno de estos modelos, se presentarán sus características, fortalezas y debilidades. El capítulo finaliza con la exposición de una serie de criterios que permiten seleccionar un determinado modelo de ciclo de vida de acuerdo con las características del proyecto de desarrollo de software.

► Capítulo 3. Procesos ágiles

En contraposición a los procesos tradicionales de desarrollo de software, han surgido los procesos ágiles que permiten a los equipos de desarrollo construir software de manera iterativa e incremental, entregar valor tempranamente, involucrar en forma permanente al usuario y responder rápidamente al cambio inherente presente en los proyectos (de acuerdo con los puntos del manifiesto ágil). El capítulo presenta los conceptos fundamentales de los procesos ágiles, describe en profundidad los principios y las prácticas de dos de los procesos ágiles más utilizados: programación extrema (eXtreme Programming [XP]) y *scrum*. Además, se realiza una descripción global de otros procesos ágiles para tener una visión más completa de esta clasificación de procesos de software. Al igual que en el capítulo anterior, se presentará un conjunto de criterios para seleccionar un proceso ágil. El capítulo contiene también una descripción de diferencias específicas entre los procesos ágiles y los procesos tradicionales de desarrollo de software. El capítulo finaliza con la descripción de diferentes métodos de escalamiento de los procesos ágiles.

► Capítulo 4. Estándares y procesos industriales de software

Este capítulo trata de los estándares mundiales desarrollados y aplicados actualmente para implementar en forma efectiva procesos de software. Muestra en qué consiste el proceso de estandarización y su relación con el proceso de software. Además de los estándares, se presentan procesos cuyo origen no surge de organismos de estandarización, sino de actividades realizadas desde la industria y la academia, pero que tienen una amplia aceptación en los entornos empresariales actuales de desarrollo de software.

Introducción a los procesos de software

► Objetivos

Los objetivos de este capítulo consisten en realizar una introducción a los conceptos fundamentales del proceso de software. Al estudiar este capítulo, el lector:

- Comprenderá el concepto de proceso de software y por qué es importante para el desarrollo de software.
- Conocerá las diferentes características y factores que poseen los procesos de software.
- Comprenderá los elementos y las actividades de un proceso de software.
- Entenderá las relaciones y diferencias que existen entre el proceso y producto de software.
- Conocerá diversos esquemas de clasificación de los procesos de software.

El desarrollo de software es una actividad crítica de nuestra sociedad. Numerosos sistemas, productos y servicios de la vida diaria utilizan software para su funcionamiento. La industria del software está bajo constante presión por desarrollar y entregar mayores volúmenes de productos y servicios de alta calidad dentro de restricciones de costos y programación cada vez más estrictos (Boehm, 2006; Kneuper, 2018; O'Regan, 2017).

Sin embargo, los proyectos de software son difíciles de gestionar y el desarrollo de software es una labor compleja. Su complejidad radica en que no solo es un problema de creación y utilización de herramientas tecnológicas y lenguajes de programación, sino que incorporan aspectos de gestión organizacional, administración de personal y aplicación de procedimientos.

La complejidad surge debido a que diferentes dimensiones del desarrollo de software no pueden ser reducidas a una sola dimensión. El proceso de software ayuda a identificar estas dimensiones del desarrollo y el problema que necesita ser resuelto estableciendo prácticas efectivas de desarrollo de software (Mustafa *et al.*, 2011). Este concepto de proceso de software viene de la ingeniería, proveniente en gran parte de la comunidad de la manufactura, y ha sido utilizado y aplicado a la producción y el mantenimiento de software (Suri y Sebern, 2004).

Un proceso de ingeniería consiste en un conjunto de actividades interrelacionadas que transforman una o más entradas en salidas mientras se consumen recursos para realizar esta transformación. Las actividades serán realizadas por ingenieros de software para desarrollar, mantener y operar software, tales como análisis de requisitos, diseño, construcción, pruebas, configuración, gestión de configuración y otras (Bourque y Fairley, 2014). Cuando se lleva esta definición al dominio de la ingeniería de software, se denomina ingeniería del proceso de software o procesos de software, en su versión más resumida.

Cada vez más, las organizaciones desarrolladoras de software están buscando aplicar conceptos de ingeniería del proceso de software como un medio para apoyar, guiar y, donde sea necesario, controlar las actividades de los proyectos de software (Humphrey, 1988).

► 1.1. Área del conocimiento del proceso de software

La figura 1.1 muestra una terna conformada por los elementos tecnología, proceso, persona para culminar un proyecto de desarrollo de software en forma exitosa.

En esta terna, los elementos constituyentes de un proyecto de desarrollo de software tienen la misma importancia. Sin procesos, es imposible apalancar todos los recursos. Sin tecnología, es difícil tratar con un mundo siempre cambiante. Por último, las personas siempre desempeñan un

papel importante coordinando procesos y tecnología. Usualmente, los tres elementos no están integrados y se requiere un amplio esfuerzo para lograr su integración. El desarrollo de software puede ser mejorado integrando estas tres perspectivas (Li, 2006).

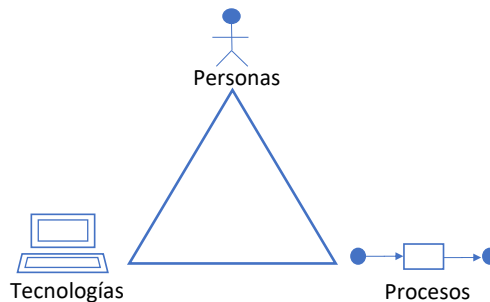


Figura 1.1. Tríada tecnología, proceso, persona

Fuente: elaboración propia con base en Li (2006).

El concepto de personas se refiere a la gestión, miembros del personal y *stakeholders*. Las personas son las responsables de alcanzar objetivos particulares y tienen diferentes roles en su trabajo diario. Es importante conocer la motivación y los objetivos de las personas y no solo medir, monitorear y mejorar sus acciones y comportamientos (Wieczorek *et al.*, 2014). Las personas conocen qué y cómo realizar sus actividades, tienen las habilidades y el conocimiento para realizar su trabajo, deben estar comprometidas para lograr un alto desempeño y estar animadas a involucrarse en proyectos de mejora (Prodan *et al.*, 2015).

El concepto de proceso generalmente se refiere a cómo alcanzar un objetivo particular. El proceso constituye la amalgama que relacionará la tecnología y las personas (Wieczorek *et al.*, 2014). Al examinar el proceso de extremo a extremo (desde los proveedores hasta el producto entregado a los clientes), es esencial analizar el proceso, identificar sus defectos y eliminarlos para tener un proceso sólido. Entregar el producto de software que se espera en el momento indicado es la meta a alcanzar por un proceso (Prodan *et al.*, 2015).

Por último, la tecnología se refiere a herramientas y técnicas utilizadas para la comunicación y realizar un trabajo eficiente. La tecnología incluye *hardware* y *software*; se vuelve cada vez más importante y visible, y está

transformando la forma de hacer negocios; el desarrollo de software no es la excepción (Prodan *et al.*, 2015).

A esta terna de elementos, se puede agregar un nuevo elemento: proyecto, el cual se refiere a cómo alcanzar un objetivo particular. Para la realización de un proyecto de software, se deben considerar aspectos como alcance, contexto, esfuerzo, plan, organización e inversión/recursos. Por tanto, con los proyectos se determinan qué se debe hacer y qué no.

El área de conocimiento del proceso de software se inició en la década de 1980 con eventos y publicaciones relacionadas con esa área. El crecimiento del interés por los procesos de software ha generado la creación de instituciones relevantes en el ámbito internacional, como el Software Engineering Institute (SEI). A su vez, algunas organizaciones internacionales de estandarización han desarrollado importantes estándares relacionados con el proceso de software, como ISO/IEC 12207 (actividades del ciclo de vida del software) e ISO/IEC 15504 (determinación de la capacidad del proceso). Los procesos de software ayudan a mejorar la productividad, los tiempos de desarrollo y, sobre todo, la calidad del producto de software.

El proceso de software de desarrollar y mantener un producto o servicio desempeña un papel crucial en la determinación del nivel de calidad del producto o servicio, pero también en el costo de desarrollarlo, darle soporte y mantenimiento (Ruiz-González y Canfora, 2004).

La ingeniería del proceso de software es el campo de estudio relacionado con el desarrollo y la aplicación de métodos para la definición, ejecución, evaluación y mejora de los procesos de ingeniería de software (Kneuper, 2018; Lonchamp, 1993).

Visto desde de un contexto sistémico, el propósito de la ingeniería del proceso de software es intervenir en un sistema de ingeniería de software introduciendo o cambiando procesos dentro de él. Para ser exitosos, los métodos de la ingeniería de procesos de software deben aplicar un enfoque holístico y sistémico que explícitamente considere y balancee múltiples perspectivas de los *stakeholders* que haga énfasis en un enfoque participativo para facilitar la construcción de consensos entre los desarrolladores (Kroeger *et al.*, 2014).

► 1.2. Historia del proceso de software

El análisis del desarrollo de software como un proceso ha ayudado significativamente a solucionar los problemas que necesitan ser resueltos para establecer prácticas efectivas de desarrollo en atención a las diferentes dimensiones del desarrollo de software (Li, 2006).

Los desarrollos iniciales en esta área datan de las décadas de 1960 y 1970 cuando los trabajos más relevantes se centran en aspectos como el desarrollo de lenguajes de programación estructurada, el desarrollo de métodos de diseño y la definición de ciclos de vida del software, como se observa en la figura 1.2.

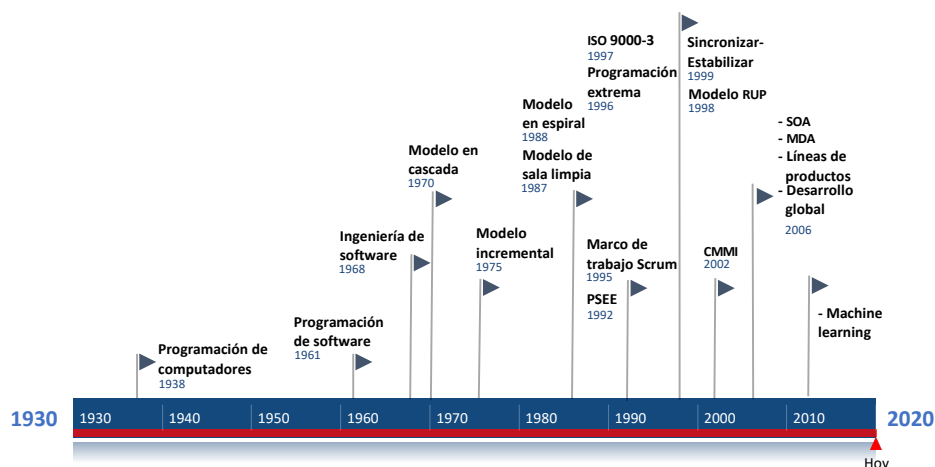


Figura 1.2. Antecedentes de los procesos de software

Fuente: elaboración propia con base en Emami *et al.* (2010).

Antes de 1960, el concepto de programación de computadores era común en ambientes científicos y de ingeniería. El término *programación de software* se introdujo a inicios de 1960. En 1968, el concepto de ingeniería de software fue propuesto. Luego, diferentes metodologías fueron apareciendo de manera secuencial.

En 1970, fue presentado el modelo en cascada por Winston Royce. En 1975, Victor Basili y Albert Turner desarrollan la técnica “mejora iterativa” para el desarrollo incremental y entrega iterativa. En 1987, Harlan Mills presentó el desarrollo de software de “sala limpia”. En 1988, Barry Boehm

presenta el “modelo en espiral”. En 1996, la “programación extrema” es propuesta por Kent Beck. En 1998, el Rational Unified Process (**RUP**) es propuesto como metodología orientada a objetos. En 1999, la metodología “sincronizar y estabilizar” es introducida por Michael Cusumano y David Yoffie.

En la década de 1980, los esfuerzos de investigación se centraron en lenguajes de modelado de procesos (Process Modelling Languages [**PML**]). Se destaca el trabajo de Osterweil (1987) con su artículo “Software process are software too” en el que destaca la importancia de realizar descripciones rigurosas del proceso que se conviertan en guías para la aplicación efectiva del cálculo computacional, y así soportar la ejecución de procesos instanciados desde estas descripciones.

Robert Balzer (1990) presentó varios problemas no resueltos en el ámbito del soporte de procesos de software:

- Relaciones de abstracción/instanciación entre procesos genéricos y específicos, y entre procesos específicos y sus ejecuciones.
- Involucramiento de personas en procesos automatizados.
- Medición de los procesos.
- Medición del progreso del proceso.
- Utilización del estado del proceso como base para realizar predicciones.
- Mejora del proceso.
- Cambio del proceso.
- Gestión de procesos inconsistentes e incompletos.

En los últimos treinta años, la investigación se ha centrado en lenguajes de definición de procesos (Finkelstein *et al.*, 1994), en la programación de procesos (Osterweil, 1987), en la construcción de ambientes de ingeniería de software centrados en procesos (Process-centered Software Engineering Environment [**PSEE**]) (Garg, 1995), en la gestión de procesos de software y la mejora y evaluación de procesos como **ISO 9000** (Anton y Anton, 2001) y **SEI-CMM/CMMI** (2010).

► 1.3. Definición y descripción del proceso de software

En este apartado, se realizará una definición de proceso de software. Luego, se mostrará una descripción del proceso junto con sus características y factores correspondientes.

1.3.1. Definición de proceso de software

En primer lugar, se debe definir qué es un proceso. Un proceso es un conjunto de pasos para realizar una tarea que deben ser organizados, gestionados, medidos, soportados y mejorados para alcanzar un objetivo (Estublier, 2005). El proceso se puede ver como un espectro donde es un *continuum* de planes en un extremo y procedimientos burocráticos en el otro (Raman, 2000). Por tanto, el proceso codifica parte del *know-how* de las empresas que de otra forma estaría únicamente en la mente de los ejecutores del proceso. Este ayuda a transferir más fácilmente el conocimiento a los nuevos integrantes, dominar los activos que se utilizan en su ejecución, incrementar su repetibilidad en diversos proyectos y permitir el análisis y la mejora de los procesos.

Es necesario diferenciar los términos *planes* y *procesos*. El plan es un diseño para realizar una tarea, pero los procesos son una serie de pasos para implementar un plan y lograr la tarea.

Un proceso de software se puede definir como “un conjunto coherente de políticas, estructuras organizativas, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto de software” (Fuggetta, 2000).

El proceso de software proporciona pasos, actividades y productos intermedios (documentos, modelos, código, etc.) y mecanismos de restricción para lograr los objetivos. Existen relaciones entre estos elementos que se manifiestan como la articulación de los mecanismos de restricción y pueden utilizarse para guiar la ejecución del proceso (Yang y Xue, 2011).

El proceso de software proporciona un marco de trabajo para gestionar actividades que pueden fácilmente salirse de control durante el desarrollo de software. Por tanto, diferentes proyectos de software requieren diferentes procesos de software (Li, 2006).

La figura 1.3 describe los procesos como elementos estáticos dentro de un sistema de ingeniería de software. Dentro de este sistema, las personas selectivamente interpretan procesos y aplican tecnologías para transformar las necesidades del cliente en productos de software. Además, el sistema opera dentro de un particular ambiente de negocios y es consecuentemente influenciado por un rango de factores económicos, sociales, tecnológicos y políticos (Kroeger *et al.*, 2014).

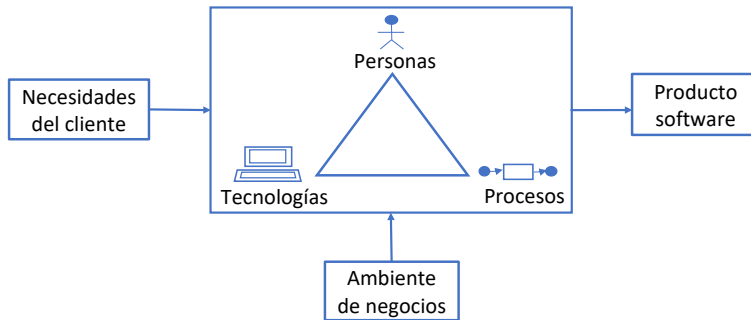


Figura 1.3. Modelo de ingeniería de software

Fuente: elaboración propia con base en Kroeger *et al.* (2014).

Una de las principales metas perseguidas por los investigadores y practicantes está centrada en el estudio del proceso con el cual el software es desarrollado. La suposición fundamental es que existe una correlación directa entre la calidad del proceso y del producto de software desarrollado (Fuggetta, 2000). Los procesos de software intentan simplificar la gestión de proyectos de software.

A continuación, se presentan las definiciones de los elementos generales de un proceso de software basado en los conceptos de Capability Maturity Model Integration (**CMMI**) (SEI, 2010) e International Organization for Standardization (**ISO**) (Anton y Anton, 2001):

- **Políticas:** Son los principios establecidos por la alta dirección para tomar decisiones.
- **Estructuras organizacionales:** Son estructuras administrativas que permiten organizar el personal para cumplir sus objetivos durante la gestión y ejecución de un proyecto.
- **Tecnologías:** Conjunto de recursos, herramientas, equipos, programas informáticos, aplicaciones, redes y medios.
- **Procedimientos:** Es un “cómo se hace” algo; son instrucciones operativas o protocolos de ejecución.
- **Artefactos:** Elementos utilizados y generados durante la ejecución del proceso, como planes de proyecto, documentación de diseño, código, casos de prueba, manuales de usuario, etc.

Un proceso encapsula la experiencia obtenida por la organización en forma de una “receta” exitosa (Jalote, 2002). Los procesos típicamente son recolectados sobre una metodología de la organización, la cual incluye mé-

todos, técnicas y herramientas de soporte, y es utilizada para planificación, entendimiento, gestión y mejora de sistemas de software. Estos procesos y metodologías se describen en términos apropiados para ser utilizados por un desarrollador, pero son frecuentemente descritos en manuales o libros que el equipo de proyecto sigue tanto como sea posible (Henderson-Sellers y González-Pérez, 2005). Sin embargo, el tamaño de los documentos relacionados con el proceso de software puede ser bastante extenso, lo que ha llevado a que este se documente a través de repositorios digitales que faciliten su acceso y consulta.

Los procesos de software aplican conceptos y desarrollos de otras áreas específicas (Fuggetta, 2000), como se observa en la figura 1.4.

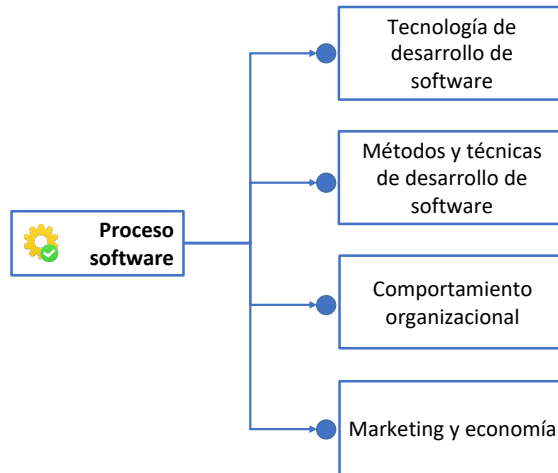


Figura 1.4. Conceptos aplicados al proceso de software

Fuente: elaboración propia.

- ▶ *Tecnología de desarrollo de software:* Es el soporte tecnológico utilizado durante el proceso, incluye herramientas, infraestructura y ambientes.
- ▶ *Métodos y técnicas de desarrollo de software:* Son guías sobre cómo utilizar la tecnología y ejecutar las actividades de desarrollo de software.
- ▶ *Comportamiento organizacional:* Los equipos de personas ejecutan el proceso de software, por lo que deben ser apropiadamente gestionados en una estructura organizacional efectiva.

- *Marketing y economía*: El software debe cumplir las necesidades de los clientes en condiciones específicas de mercado. Estas condiciones afectan la forma de estructurar el proceso a contextos de mercado específicos.

Por tanto, el proceso de software tiene en cuenta una interrelación compleja de factores organizacionales, culturales, tecnológicos y económicos.

Los procesos de software son definidos con el suficiente nivel de detalle para que puedan ser consistentemente utilizados. Los procesos son definidos para entender, comunicar, ejecutar y gestionar el proceso de desarrollo (Sutton, 2000). Su definición permitirá facilitar el entendimiento humano, la comunicación y coordinación; ayudar a la administración de proyectos; medir y mejorar la calidad de los productos de software de una manera efectiva, y soportar la mejora de procesos y proporcionar una base para el soporte automatizado de su ejecución (Bourque y Fairley, 2014).

Los procesos de software pueden estar definidos explícitamente o pueden ser procesos implícitos que se usan sin tener conciencia del proceso como tal. Entre las razones para definir procesos de software se encuentran (Kneuper, 2002):

- Los procesos definidos proporcionan una estructura para el desarrollo de software.
- Los buenos procesos de software dan soporte a los desarrolladores en su trabajo. Utilizando el proceso de software, los desarrolladores conocen qué pasos se deben realizar y proporcionan plantillas para los resultados que se producen.
- Los procesos de software definidos conforman la base para la mejora y el aprendizaje, que incluye monitorización, simulación, verificación y validación del proceso.
- Los procesos de software definidos hacen que el desarrollo sea más predecible, ayudando a obtener estimaciones de tiempo y costo más reales y obteniendo productos de calidad.

1.3.2. Descripción del proceso de software

La descripción de un proceso de software debe incluir como mínimo las entradas requeridas, las actividades de transformación y las salidas generadas. Como se observa en la figura 1.5, un proceso de software puede comprender otros elementos (Bourque y Fairley, 2014).

- ▶ *Entradas requeridas*: Pueden ser eventos o salidas de otros procesos.
- ▶ *Actividades de transformación*: Toman las entradas requeridas y al ejecutar las tareas en las que se descomponen las actividades desarrollan las salidas correspondientes.
- ▶ *Salidas generadas*: Son los productos de trabajo resultantes de ejecutar las actividades de transformación.
- ▶ *Criterios de entrada*: Estos criterios deben ser satisfechos antes que un proceso inicie.
- ▶ *Criterios de salida*: Son las condiciones especificadas que se deben satisfacer antes de que un proceso pueda concluirse, que incluye criterios de aceptación de los productos de trabajo de salida.
- ▶ *Descomposición de las actividades en tareas*: Las tareas son las unidades de trabajo más pequeñas sujetas a administración. A su vez, la descomposición conforma un flujo de trabajo, es decir, cómo se relacionan los elementos del proceso entre sí.
- ▶ *Roles y competencias del personal*: Encargados de ejecutar las actividades de transformación.
- ▶ *Soporte de tecnología de la información (TI)*: incluye técnicas y herramientas de ingeniería de software y ambientes tecnológicos necesarios para ejecutar el proceso.
- ▶ *Medidas (indicadores clave de desempeño, o Key Performance Indicators [KPI])*: Para determinar la eficiencia y efectividad de la ejecución del proceso.

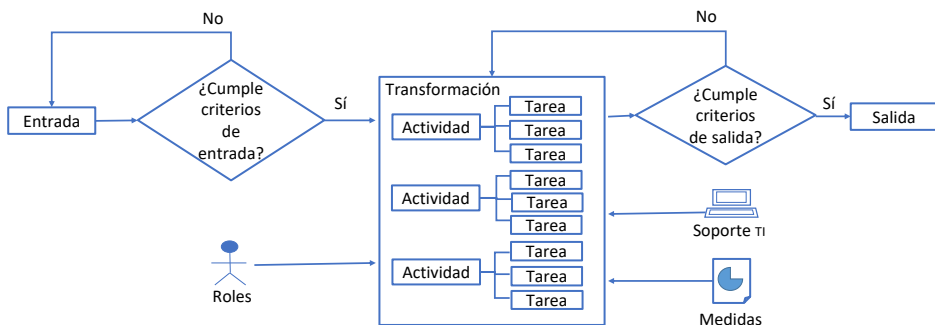


Figura 1.5. Elementos de un proceso de software

Fuente: elaboración propia con base en Bourque y Fairley (2014).

Los procesos se pueden descomponer en subprocesos, y así formar una jerarquía de procesos con un cierto nivel de profundidad. El proceso de software también puede incluir actividades intercaladas de gestión, colaborativas y técnicas.

Los procesos a su vez incluyen en forma implícita metodologías, lenguajes y modelos. Por tanto, es bueno diferenciar estos términos para que no se preste a confusiones el uso de diferente terminología (Yang y Xue, 2011).

- *Metodología de software*: Proporcionan modos de pensamiento, análisis y métodos para la resolución de problemas. Las metodologías comunes y tradicionales en el proceso de software son el análisis y el diseño de sistemas orientados a objetos.
- *Lenguajes*: Soportan la implementación de procesos de software y son utilizados para describir resultados de varias actividades del proceso de software. Se pueden utilizar tanto para el modelado (por ejemplo, Unified Modeling Language [UML]) como para escribir código (por ejemplo, Java).
- *Modelos*: Son simplificaciones de un problema o abstracciones del sistema. En general, se refieren a especificaciones, descripciones y presentación de la función, estructura o comportamiento de un sistema de software a ser construido.

1.3.3. Características y factores del proceso de software

Las características esenciales que debe tener un proceso de software ayudan a destacar las fortalezas y debilidades inevitables de los enfoques de procesos (Erdogmus, 2008), como se observa en la figura 1.6.

- *Centrado en el ser humano*: El desarrollo de software, al ser un esfuerzo intelectual y una actividad realizada en equipo, es un proceso centrado en el ser humano que hace énfasis en la colaboración, reconoce la importancia del liderazgo efectivo y satisface las necesidades de profesionales creativos. Por tanto, requiere entender qué motiva a los trabajadores, cómo ejecutan su trabajo e interactúan con otros.
- *Orientación técnica*: El proceso promueve la posesión de competencias fundamentales y técnicas. Las competencias fundamentales, como en algoritmos, estructuras de datos, paradigmas de programación, patrones de diseño, pruebas, requisitos, interfaces de usuario y arquitecturas; las competencias técnicas como lenguajes de programación, herramientas, ambientes, *frameworks* y plataformas.



Figura 1.6. Características de los procesos de software

Fuente: elaboración propia.

- ▶ *Disciplina*: La disciplina se refiere al uso sistemático de prácticas sinérgicas apropiadas para un contexto, con una sólida estructuración del flujo de trabajo y la división del trabajo. Un proceso disciplinado aplica consistentemente prácticas de gestión para coordinar actividades, compartir conocimiento y controlar artefactos.
- ▶ *Pragmatismo*: Un proceso pragmático consiste en que sus actividades deben ser continuamente factibles, adaptadas a su contexto, escalables y compatibles con la cultura cambiante de la organización. Una actividad del proceso de software debe servir con un propósito claro, ya sea satisfacer las necesidades de los clientes, de los equipos del proyecto o de los usuarios.
- ▶ *Empirismo*: Soportando la toma de decisiones a través de evidencias basadas en datos, observaciones y mediciones. De esta forma, ayuda al equipo a averiguar cuándo está fallando o logrando el éxito o si está mejorando o empeorando, descubriendo por qué las cosas suceden y tomando medidas correctivas y preventivas.
- ▶ *Experimentación*: El desarrollo de software conlleva incertidumbre. Para reducirla, el proceso requiere la utilización de prototipos, simulaciones, juegos de roles y evaluaciones tempranas de usabilidad, entre otros. Con la experimentación, se logrará el aprendizaje

a través del ensayo y el error. Un proceso que es intolerante y que castiga los resultados transitorios desfavorables desalentará la experimentación.

- *Orientación al valor*: Los proyectos de software operan bajo limitaciones escasas de recursos. Por tanto, se deben considerar los recursos en términos de costo-beneficio, productividad y calidad, repetibilidad y flexibilidad, satisfacción del cliente/usuario, predictibilidad y creatividad, autoridad y autonomía, y objetivos a corto y largo plazo.

El trabajo de Clarke y O'Connor (2012) muestra una clasificación de factores situacionales que afectan el proceso de desarrollo de software. Los factores forman un *framework* de referencia que consiste de ocho clasificaciones y 44 factores. Las clasificaciones generales son las siguientes:

- *Personal*: Constitución y características del personal no directivo involucrado en los esfuerzos de desarrollo de software.
- *Requisitos*: Características específicas de los requisitos de software.
- *Aplicación*: Características de las aplicaciones en desarrollo.
- *Tecnología*: Perfil de la tecnología que se utiliza para el esfuerzo de desarrollo de software.
- *Organización*: Perfil de la organización desarrolladora de software.
- *Operación*: Consideraciones operacionales y limitaciones.
- *Gestión*: Constitución y características del equipo de gestión de desarrollo de software.
- *Negocio*: Consideración estratégica y táctica del negocio.

► 1.4. Proceso vs. producto

La calidad del proceso se justifica por la observación natural que los buenos resultados rara vez surgen de malos procesos. La calidad se enfoca en la calidad de los recursos y actividades, en controlar los costos y la planificación, en verificar los resultados y la prevención de riesgos. La calidad del proyecto ayuda a detectar y resolver a tiempo problemas que pueden hacer fallar un proyecto de desarrollo de software.

Las empresas desarrolladoras deben mostrar buenos procesos para demostrar sus capacidades al cliente. Por tanto, deben manifestar una gestión de proyectos sin fallas. Sin embargo, los proyectos son instanciaciones

de los procesos. Por tanto, la calidad del proceso es necesaria para la calidad del proyecto y, en consecuencia, de la calidad del producto. Esta debe entregar resultados que cumplan completamente los requisitos del cliente. El control de calidad requiere un proceso bien definido y correctamente aplicado (Cignoni, 2000).

Un buen proceso es necesario para organizar y controlar las actividades de desarrollo. La calidad del proceso es fundamental para lograr la calidad del producto. Los procesos efectivos reducen defectos y mejoran el valor de los productos, los procesos eficientes decrecientan costos y los procesos flexibles permiten a la empresa enfrentar exitosamente la demanda de innovación y los cambios en el mercado (Cignoni, 2000).

Por tanto, ambos aspectos, el producto y el proceso, están relacionados. Por ejemplo, el número y la severidad de errores en un programa de computador es un ejemplo de aspecto relacionado con el producto. Si el programa es entregado según el cronograma establecido es un ejemplo de aspecto relacionado con el proceso. Ambos aspectos están interrelacionados: el proceso determina en parte la introducción y eliminación de errores, y tratar de entregar el producto a tiempo, como el objetivo principal, puede afectar su funcionalidad (Meyer, 2009).

La relación entre la calidad de un producto final y un proceso es importante debido a que la selección de procesos y la adopción general por parte de las empresas requiere que el personal comprenda los beneficios alcanzables. El personal solo adoptará y seguirá procesos que proporcionen beneficios probados. Estos beneficios pueden ser en términos de a) crear un mejor producto final (o servicio), b) hacer que las cosas sucedan más rápido o c) facilitar el trabajo de las personas (Forrester, 2006).

► 1.5. Actividades del proceso de software

Los procesos de software están conformados por secuencias entrelazadas de actividades de diversa índole, técnicas, colaborativas y administrativas para desarrollar un sistema de software. La literatura referencia numerosas actividades del proceso de software, cada una con un cierto nivel de abstracción y granularidad. La forma en que se ejecutan estas actividades depende del tipo de software, de las estructuras organizacionales y el personal. En la figura 1.7, se presenta un conjunto genérico de actividades del proceso.

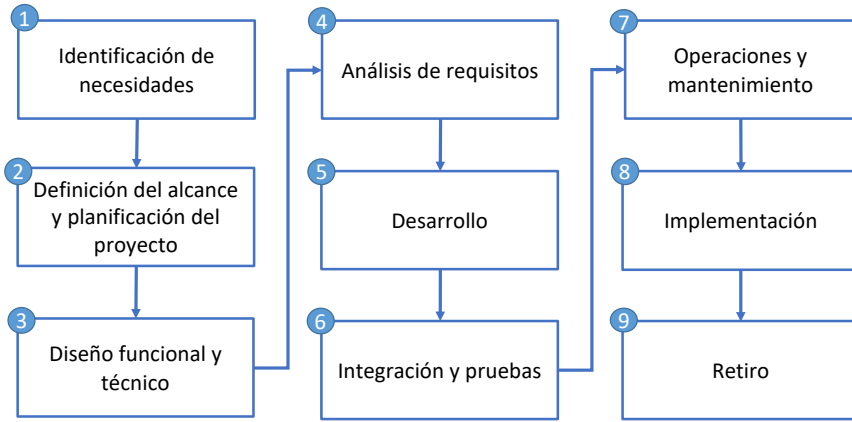


Figura 1.7. Actividades del proceso de software

Fuente: elaboración propia con base en Wieczorek *et al.* (2014).

Las actividades del proceso inician con una evolución de las necesidades del producto a desarrollar hasta que este se lanza en operación. Cada una de las actividades se describe brevemente a continuación (Wieczorek *et al.*, 2014).

1. *Identificación de necesidades:* Inicia cuando una necesidad de negocio u oportunidad se identifica. Estas necesidades son documentadas en una propuesta conceptual.
2. *Definición del alcance y la planificación del proyecto:* Se revisa la factibilidad de la propuesta conceptual, se identifica el alcance del sistema y se aprueban los recursos generales. Finaliza con la aprobación del documento de planificación y de conceptos del sistema.
3. *Análisis de requisitos:* Los requisitos funcionales y no funcionales son formalmente definidos. Finaliza con la aprobación del documento de requisitos detallados.
4. *Diseño funcional y técnico:* Se diseñan las características funcionales y técnicas del sistema, se establece el ambiente operativo, los requisitos de entrada son documentados y revisados por el usuario, las características físicas del sistema son especificadas y un diseño detallado es preparado. Finaliza con la aprobación del documento de diseño.

5. *Desarrollo*: Las especificaciones detalladas de diseño son convertidas en software ejecutable junto con *hardware* y comunicaciones. Finaliza con el reporte completo de pruebas detallado.
6. *Integración y pruebas*: Los componentes del sistema son integrados y sistemáticamente probados para conformar el modelo de calidad del producto. Finaliza con la aprobación del reporte de integración y pruebas.
7. *Implementación*: El sistema es instalado y es operacional en un ambiente de producción. Finaliza con un concepto de aprobación por parte de un departamento de operaciones.
8. *Operaciones y mantenimiento*: El sistema es monitoreado para que su desempeño cumpla con los requisitos de usuario y las modificaciones necesarias son incorporadas a medida que surjan. El sistema es periódicamente valorado para determinar cómo puede ser más eficiente y efectivo.
9. *Retiro*: Se garantiza la finalización adecuada del sistema y se preserva la información vital para que pueda reactivarse en el futuro si es necesario. Se preservan los datos procesados, de tal manera que son efectivamente migrados a otro sistema o archivados de acuerdo con las regulaciones aplicables.

► 1.6. Gestión del proceso de software

Los objetivos de la gestión del proceso son lograr la eficiencia y efectividad que resultan al aplicar un enfoque sistemático al ejecutar procesos de software y al desarrollar productos de trabajo (a nivel individual, organizacional o de proyecto), e introducir nuevos o mejorados procesos.

La gestión del proceso establece un marco de trabajo para aplicar métodos, tecnologías y personal al desarrollo de software. Esta abarca cuatro responsabilidades (Florac *et al.*, 1997): definición del proceso, medición del proceso, control del proceso y mejora del proceso, como se observa en la figura 1.8.

- *Definición del proceso*: Crea el ambiente disciplinado y estructurado requerido para controlar y mejorar el proceso. Además de diseñarlo, se definen modelos y medidas relacionados con la ejecución del proceso y se proporciona la infraestructura necesaria (métodos, personas y prácticas).

- *Medición del proceso*: La medición es la base para detectar desviaciones en el desempeño del proceso e identificar oportunidades de mejora.
- *Control del proceso*: Mantiene el proceso dentro de los límites de un desempeño normal permitiendo que el proceso se ejecute correctamente. Involucra la corrección de las desviaciones detectadas.
- *Mejora del proceso*: Los procesos deben ser tecnológicamente competitivos, adaptables y oportunos, y generar productos que cumplan consistentemente las necesidades del cliente. Los procesos se pueden mejorar con cambios que mejoren sus capacidades o reemplazando sus componentes por otros. Incluye el cambio a nuevos procesos, la mejora de procesos existentes, cambios organizacionales o el cambio de infraestructura.

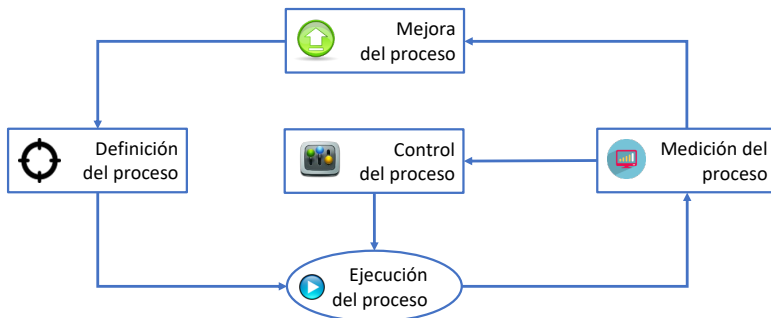


Figura 1.8. Gestión del proceso de software

Fuente: elaboración propia con base en Florac *et al.* (1997).

La ejecución del proceso tiene una forma distinta (óvalo en lugar de rectángulo) en la figura 1.8 porque no es responsabilidad de la gestión del proceso de software, sino de la gestión de proyectos si es ejecutado por un encargado del desarrollo o mantenimiento de software (Florac *et al.*, 1997).

► 1.7. Clasificación de procesos de software

En la literatura, se encuentran varias clasificaciones de procesos de software. En la figura 1.9, se determinan varias clasificaciones de acuerdo con el tamaño, el tipo o la agilidad de los procesos.

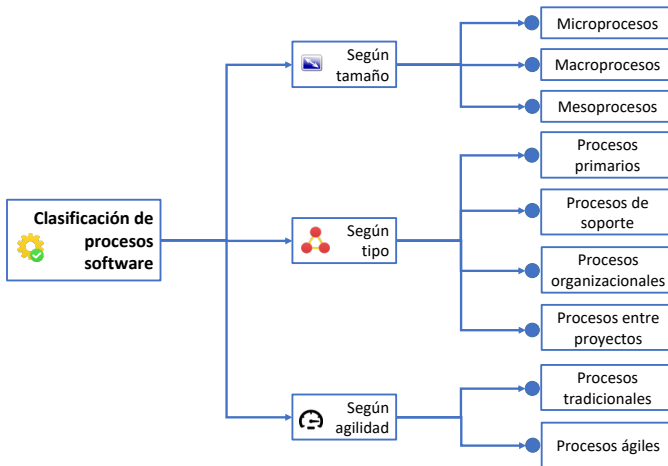


Figura 1.9. Clasificación de procesos

Fuente: elaboración propia.

Una clasificación básica de procesos puede ser en microprocesos, macroprocesos y mesopprocesos (Sutton, 2000):

- ▶ *Microprocesos:* Tienen una descripción que es precisa, completa, detallada y no ambigua, se centran en detectar fallas en el proceso y facilitar la sinergia humano-computador.
- ▶ *Macroprocesos:* Describen arquitecturas de procesos, características de comportamiento de los procesos y que estos se ajusten a las características y los sistemas organizacionales.
- ▶ *Mesopprocesos:* Tienen como finalidad cumplir los objetivos definidos en el nivel macro en términos de formalismos definidos en el nivel micro. Se interesa por procesos y plataformas de desarrollo de procesos, selección y aplicación de lenguajes de procesos y plataformas de ejecución del proceso.

Otra clasificación bastante común y referenciada en la literatura es la propuesta por Fairley (2009), Moore (2006) y Sommerville (2015):

- ▶ *Procesos primarios:* Incluyen los procesos principales orientados al desarrollo, la operación y el mantenimiento de software.
- ▶ *Procesos de soporte:* Procesos aplicados intermitente o continuamente a través del ciclo de vida para soportar procesos primarios. Por ejemplo: gestión de configuración, aseguramiento de la calidad y verificación y validación de software.

- *Procesos organizacionales*: Proporcionan soporte para la ingeniería de software. Incluyen procesos como entrenamiento, análisis de la medición del proceso, gestión de infraestructura, mejora del proceso y gestión de modelos del ciclo de vida del software.
- *Procesos entre proyectos*: Involucran más de un proyecto de software en una organización. Por ejemplo, procesos de reúso, líneas de productos de software e ingeniería de dominio.

Algunos procesos de software son pesados, con muchos artefactos a ser generados; otros son ligeros, basados en una colaboración estrecha entre los miembros del equipo del proyecto (Yang y Xue, 2011). Otra clasificación adicional de los procesos de software es agruparlos en procesos tradicionales, también llamados prescriptivos o dirigidos por plan, y procesos ágiles (Isaias y Iss, 2014):

- *Procesos tradicionales*: Todas las actividades de los procesos se planifican en etapas tempranas y el avance se mide con respecto al plan. Son procesos rígidos, se realizan paso a paso y su documentación es exhaustiva. Además, tienen los requisitos bien definidos y una tardía participación del usuario, generalmente en etapas finales como pruebas.
- *Procesos ágiles*: La planificación es incremental y el proceso evoluciona para reflejar cambios en los requisitos. Se caracterizan por estar orientados al diseño, los requisitos son flexibles, hay comunicación directa con el cliente y una participación frecuente del usuario en el proceso.

Cada enfoque es adecuado para diferentes tipos de software (Boehm y Turner, 2003). En la práctica, la mayoría de los procesos incluyen elementos de ambos enfoques, ya que no hay procesos correctos o equivocados.

► 1.8. Diversidad del proceso de software

¿Por qué cada proceso de software es diferente? Porque las organizaciones lo son y sus procedimientos, conjunto de habilidades, etc., son también diferentes. Además, dos proyectos dentro de una organización pueden ser diversos, por ejemplo, soluciones de comercio electrónico o sistemas críticos de seguridad. Así, un proceso que puede ser exitosamente aplicado a un proyecto puede ser inapropiado o producir fallas en otro proyecto (Pedreira *et al.*, 2007). Los procesos son diversos debido a las siguientes características (Mustafa *et al.*, 2011), como se observa en la figura 1.10:

- ▶ *Objetivos específicos organizacionales:* Cada organización tiene sus propios objetivos específicos. Diferentes organizaciones pueden lograr el mismo objetivo aplicando diferentes actividades. Esto permite diversos procesos para el mismo objetivo organizacional.
- ▶ *Dominio específico:* Los procesos se deben adaptar a los requisitos y al contexto específico de cada proyecto.
- ▶ *Estructura organizacional:* Un proceso es utilizado para mejorar la coordinación, colaboración y comunicación entre las personas. Si un equipo tiene más roles, productos de trabajo, revisiones y estándares, necesitará mayores y mejores metodologías.
- ▶ *Críticidad del sistema:* En sistemas críticos, un pequeño error no detectado puede causar desastres. Por tanto, se requiere un enfoque orientado a gestionar los riesgos en los que nuevos procesos son inevitables.
- ▶ *Comunicación efectiva:* Diferentes procesos hacen énfasis o no en la comunicación entre el cliente y el equipo de desarrollo. El éxito o fracaso de un proyecto depende del involucramiento del usuario y la coordinación entre los diferentes *stakeholders*.
- ▶ *Prioridades del proyecto:* Las prioridades pueden ser entregas tempranas, software libre de errores, etc. Cada prioridad produce diferentes procesos.
- ▶ *Influencia de la competencia:* Nuevos enfoques son adaptados como influencia de los competidores. Por tanto, el proceso siempre estará cambiando de acuerdo con estas circunstancias.
- ▶ *Aplicación y personalización:* Los procesos hoy día son genéricos y cuando son utilizados en aplicaciones específicas necesitan ser personalizados. Si la personalización requiere mucho esfuerzo, es mejor crear un nuevo proceso. Para ello, se requiere creatividad y desarrollar un sólido entendimiento del proceso.
- ▶ *Efectividad:* Un proceso efectivo es aquel que ayuda a producir un producto correcto. El proceso debe ser capaz de determinar lo que el cliente quiere, producirlo y verificarlo con este. Por tanto, se requiere una retroalimentación constante con el cliente.
- ▶ *Facilidad de mantenimiento:* El proceso debe lograr este objetivo, proporcionando a los desarrolladores las fallas tan pronto como sea posible para resolverlas con poco esfuerzo.
- ▶ *Predictibilidad:* El proceso debe tener la capacidad de predecir qué tanto tiempo tomará desarrollar el producto.

- *Mejora*: El cambio es inevitable, ya que no hay soluciones que se ajusten a todo. El proceso debe ser capaz de evolucionar y mejorar con el tiempo.

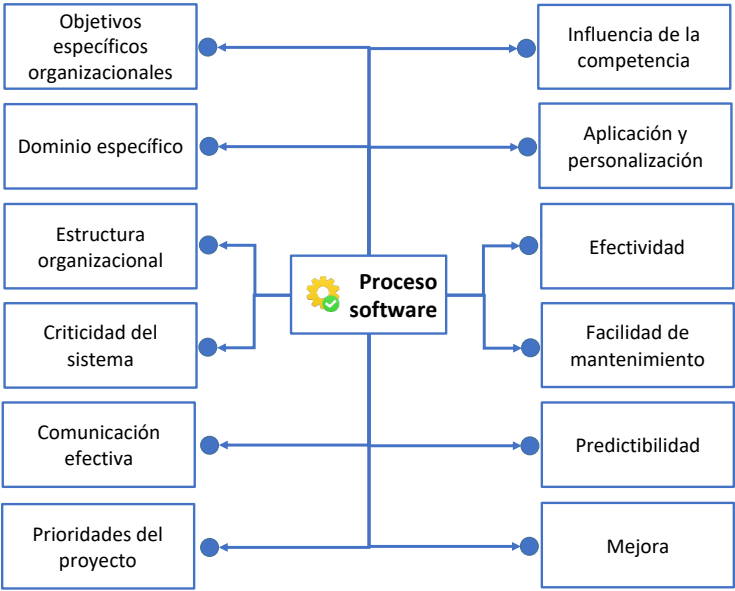


Figura 1.10. Características de la diversidad del proceso de software

Fuente: elaboración propia.

► **Desafíos**

No existe un proceso único de software o un enfoque particular que ofrezca un conjunto de principios o prácticas igualmente adecuados para cualquier propósito o que sea lo suficientemente flexible como para ser aplicable en cualquier tipo de proyecto (Brooks, 1986). El requisito básico de un proceso de software adecuado es que debe ajustarse a las necesidades del proyecto que se pretende desarrollar (Feiler y Humphrey, 1993).

Las necesidades de un proyecto se basan en el contexto de la situación en la que debe funcionar. Por tanto, el proceso de software depende del contexto. Algunos factores que caracterizan el contexto de desarrollo de software y que pueden afectar significativamente la adopción de un proce-

so de software a nivel organizacional, proyecto o equipo de desarrollo son (Clarke y O'Connor, 2012): tipo de gobierno de la organización, dominio del negocio (consideraciones estratégicas y tácticas), madurez de la organización, nivel de innovación, tipo de cultura organizacional, tamaño del sistema a desarrollar (características de los requisitos), personal (características del personal técnico y de gestión), distribución del esfuerzo en el equipo de desarrollo, esfuerzo arquitectónico (consideraciones y restricciones operacionales), tasas de cambios y criticidad (y otros requisitos no funcionales), entre otros.

La variedad y diversidad de factores que se han identificado sirve como un recordatorio del nivel de complejidad involucrado en el proceso de desarrollo de software. A su vez, la falta de apreciación de esta complejidad es también un factor que contribuye a aumentar las dificultades al afrontar un proyecto de software. Considerar estos factores es fundamental para brindar apoyo a los profesionales que enfrentan el desafío de definir y mantener procesos de software.

► Ejemplos

Una característica importante que se ha resaltado en este capítulo es que el proceso de software está centrado en las personas. Un ejemplo de esta característica es la ley de Conway (Conway, 1968) que dice: “Las organizaciones que diseñan sistemas están obligadas a producir diseños que son copias de las estructuras de comunicación de estas organizaciones”. Por tanto, esta ley se basa en el razonamiento de que para que un módulo de software funcione varios de sus autores deben comunicarse con frecuencia entre sí. La estructura del software reflejará los límites sociales de la organización que lo produjo, a través de los cuales la comunicación es más difícil.

Un ejemplo de aplicación de esta ley es el caso de una empresa desarrolladora de software que tiene cuatro grupos trabajando en un analizador del genoma humano. Por consiguiente, el software producido obtendrá un analizador del genoma de cuatro pasos. Como conclusión de la aplicación de esta ley, algunas propiedades importantes de un software dependen directamente de su proceso de desarrollo.

La aplicación de la ley de Conway tiene implicaciones interesantes en la estructura organizacional de las empresas desarrolladoras de software. Si las unidades organizacionales de las empresas no reflejan las partes esen-

ciales del producto, o si la relación entre organizaciones no refleja estas relaciones, entonces el proyecto de software puede estar en problemas. Un ejemplo adicional sería una organización que desarrolla sitios web con un contenido y una estructura que refleja las preocupaciones internas de la organización en lugar de las necesidades de los usuarios del sitio web. Por tanto, se debe asegurar que la organización sea compatible con la arquitectura del producto. Aquí se puede ver la relación estrecha que hay entre proceso y producto.

La información digital disponible en los repositorios de proyectos de software brinda la oportunidad de comprender mejor la ley de Conway en tareas y oportunidades de beneficiarse al alinear las dependencias que enfrentan los desarrolladores en sus tareas con sus actividades de comunicación y coordinación. Algunos aspectos a considerar son aprovechar los repositorios de software para las dependencias técnicas y los patrones de coordinación de los desarrolladores, identificar dependencias lógicas basadas en tareas, mantener el conocimiento de las dependencias de tareas y descubrir si la coordinación del equipo y las tareas se alinean (Kwan *et al.*, 2011).

► Resumen

El proceso de software es un área de conocimiento relevante incluida dentro de la ingeniería de software que junto con las personas y la tecnología aplica conceptos, técnicas y procedimientos que permiten desarrollar proyectos de desarrollo de software. Sus antecedentes se remontan a inicios de 1960 y aún hoy sigue siendo motivo de investigación.

El proceso de software se define como un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos necesarios para desarrollar, instalar y mantener un producto de software. Los procesos de software aplican conceptos y desarrollos de otras áreas: tecnología de desarrollo de software, métodos y técnicas de desarrollo de software, comportamiento organizacional y *marketing* y economía.

Los principales elementos de un proceso de software son entradas, salidas, actividades y tareas, criterios de entrada y salida, roles, soporte π y medidas.

El proceso de software se caracteriza por ser centrado en el ser humano, orientado a la técnica y al valor, tener disciplina y aplicar principios de pragmatismo, empirismo y experimentación.

Existe una relación significativa entre la calidad del proceso de software y la calidad obtenida en el producto de software final. La calidad del proceso reduce defectos y costos, y permite afrontar la innovación y los cambios en el mercado.

Las actividades básicas del proceso de software son identificación de necesidades, definición del alcance y planificación del proyecto, análisis de requisitos, diseño funcional y técnico, desarrollo, integración y pruebas, implementación, operación y mantenimiento y retiro.

La gestión del proceso de software establece un marco de trabajo para aplicar métodos, tecnologías y personas a la tarea de desarrollar software. La gestión del proceso abarca las responsabilidades de definición, control, medición, mejora y ejecución del proceso.

Existen varias clasificaciones del proceso de software. Una clasificación se basa en su tamaño en micro-, meso- y macroprocesos. Una segunda clasificación de acuerdo con su tipo en procesos primarios, de soporte, organizacionales y entre proyectos. Por último, una tercera clasificación en procesos tradicionales y ágiles.

Cada proceso de software aplicado en cada empresa en diferentes proyectos es diferente debido a que posee características que lo hacen diverso.

Una idea a considerar es que no hay un mejor proceso de software o conjunto de procesos de software. Estos deben ser seleccionados, adaptados o aplicados apropiadamente a cada proyecto y contexto organizacional. No existe un proceso o conjunto de procesos ideal.

► Ejercicios

1. A la tríada conformada por personas, tecnologías y procesos, ¿podría agregar, de acuerdo con su juicio, experiencia y conocimiento, más elementos esenciales?
2. De acuerdo con la definición de proceso de software, “un conjunto coherente de políticas, estructuras organizativas, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto de software”, especificar para una fase del desarrollo como el análisis: sus políticas, estructuras organizativas, tecnologías, procedimientos y artefactos.
3. Sugiera un proceso de software para las siguientes acciones:

- Cambiar un neumático de un vehículo.
 - Escribir un correo electrónico.
 - Jugar un partido de fútbol.
4. Según su experiencia desarrollando software, responda brevemente estas preguntas relacionadas con el proceso de software:
 - ¿Qué etapa es la más difícil del proceso? ¿Por qué?
 - ¿Qué etapa es la más sencilla del proceso? ¿Por qué?
 5. Lea el artículo “No silver bullet: Essence and accident in software engineering”, de Frederick Brooks. Discuta las ideas presentadas en este y su relación con el proceso de software.
 6. Realice una reflexión sobre la relación entre proceso y producto. ¿Cómo podría verificar una relación directa entre esos dos conceptos?

► Referencias

- Anton, C. y Anton, D. (2001). *ISO 9000:2000 Survival Guide: 30 Minutes to Understanding the Process*. AEM Consulting Group.
- Balzer, R. (1990). What we do and don't know about software process. En *Proceedings of the 6th International Software Process Workshop 'Support for the Software Process'* (pp. 61-62). IEEE Computer Society. <https://doi.ieeecomputersociety.org/10.1109/ISPW.1990.659574>
- Boehm, B. (2006). A view of 20th and 21st century software engineering. En *Proceedings of the 28th International Conference on Software Engineering* (pp. 12-29). <https://doi.org/10.1145/1134285.1134288>
- Boehm, B. W. y Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley.
- Bourque, P. y Fairley, R. E. (eds.) (2014). *Guide to the software engineering body of knowledge, version 3.0*. IEEE Computer Society. http://artemisa.unicauca.edu.co/~cardila/IS_SWEBOKv3.pdf
- Brookse, F (1986). No silver bullet: Essence and accident in software engineering. En *Proceedings of the IFIP 10th World Computing Conference, Dublin, Ireland* (pp. 1069-1076).
- Cignoni, G. A. (2000). Software process technologies and the competitiveness challenge. En R. Conradi (eds.), *Software Process Technology. EWSPT 2000. Lecture Notes in Computer Science* (pp. 151-155). Springer. <https://doi.org/10.1007/BFb0095024>
- Clarke, P. y O'Connor, R. V. (2012). The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5), 433-447. <https://doi.org/10.1016/j.infsof.2011.12.003>
- Conway, M. E. (1968). How do committees invent? *Datamation*, 14(4), 28-31. <https://hashingit.com/elements/research-resources/1968-04-committees.pdf>

- Emami, M. S., Ithnin, N. B. y Ibrahim, O. (2010). Software process engineering: Strengths, weaknesses, opportunities and threats. En *INC2010: 6th International Conference on Networked Computing* (pp. 1-5). IEEE.
- Erdogmus, H. (2008). Seven essentials of software process. En *Proceedings of the 1st International Workshop on Business Impact of Process Improvements* (pp. 39-40). <https://doi.org/10.1145/1370837.1370846>
- Estublier, J. (2005). Software are processes too. En M. Li, B. Boehm y L. J. Osterweil (eds.), *Software Process Workshop* (pp. 25-34). Springer. https://doi.org/10.1007/11608035_3
- Fairley, R. E. (2009). *Managing and leading software projects*. John Wiley & Sons.
- Feiler, P. H. y Humphrey, W. S. (1993). Software process development and enactment: Concepts and definitions. En *Proceedings of the Second International Conference on the Software Process-Continuous Software Process Improvement* (pp. 28-40). IEEE. <https://doi.org/10.1109/SPCON.1993.236824>
- Finkelstein, A., Kramer, J. y Nuseibeh, B. (1994). *Software process modelling and technology*. Research Studies Press.
- Florac, W., Park, R. y Carleton, A. (1997). *Practical software measurement: Measuring for process management and improvement*. CMU/SEI-97-HB-003. Software Engineering Institute, Carnegie Mellon University. https://insights.sei.cmu.edu/documents/1627/1997_002_001_16529.pdf
- Forrester, E. (2006). *A process research framework: The International Process Research Consortium (IPRC)*. Software Engineering Institute, Carnegie Mellon University. https://insights.sei.cmu.edu/documents/1301/2006_014_001_30505.pdf
- Fuggetta, A. (2000). Software process: A roadmap. En *Proceedings of the Conference on the Future of Software Engineering* (pp. 25-34). <https://doi.org/10.1109/CERMA.2012.25>
- Garg, P. K. (1995). *Process-centered software engineering environments*. IEEE Computer Society Press.
- Henderson-Sellers, B. y González-Pérez, C. (2005). A comparison of four process meta-models and the creation of a new generic standard. *Information and Software Technology*, 47(1), 49-65. <https://doi.org/10.1016/j.infsof.2004.06.001>
- Humphrey, W. S. (1988). Characterizing the software process: A maturity framework. *IEEE Software*, 5(2), 73-79. <https://doi.org/10.1109/52.2014>
- Isaias, P. y Issa, T. (2015). *High level models and methodologies for information systems*. Springer. <https://doi.org/10.1007/978-1-4614-9254-2>
- Jalote, P. (2002). *Software project management in practice*. Addison Wesley.
- Kneuper, R. (2002). Supporting software processes using knowledge management. En S. K. Chang (ed.), *Handbook of software engineering and knowledge engineering* (vol. 2, pp. 579-606). World Scientific. https://doi.org/10.1142/9789812389701_0025
- Kneuper, R. (2018). *Software processes and life cycle models: An introduction to modelling, using and managing agile, plan-driven and hybrid processes*. Springer.
- Kroeger, T. A., Davidson, N. J. y Cook, S. C. (2014). Understanding the characteristics of quality for software engineering processes: A grounded theory investigation. *Information and Software Technology*, 56(2), 252-271. <https://doi.org/10.1016/j.infsof.2013.10.003>
- Kwan, I., Cataldo, M. y Damian, D. (2011). Conway's law revisited: The evidence for a task-based perspective. *IEEE Software*, 29(1), 90-93. <https://doi.org/10.1109/MS.2012.3>

- Li, M. (2006). Expanding the horizons of software development processes: A 3-D integrated methodology. En M. Li, B. Boehm y L. J. Osterweil (eds.), *Unifying the Software Process Spectrum. SPW 2005. Lecture Notes in Computer Science* (pp. 54-67). Springer. https://doi.org/10.1007/11608035_6
- Lonchamp, J. (1993). A structured conceptual and terminological framework for software process engineering. En *Proceedings of the Second International Conference on the Software Process-Continuous Software Process Improvement* (pp. 41-53). IEEE. <https://doi.org/10.1109/SPCON.1993.236823>
- Meyer, B. (2009). *Touch of class: Learning to program well with objects and contracts*. Springer. <https://doi.org/10.1007/978-3-540-92145-5>
- Moore, J. W. (2005). *The road map to software engineering: A standards-based guide*. Wiley-IEEE Computer Society.
- Mustafa, G., Hafeez, Y. y Abbas, M. A. (2011). Fundamental characteristics creating software process diversity. En *International Conference on Computer Networks and Information Technology* (pp. 341-344). IEEE. <https://doi.org/10.1109/ICCINIT.2011.6020891>
- O'Regan, G. (2017). *Concise guide to software engineering: From fundamentals to application methods*. Springer.
- Osterweil, L. (2011). Software processes are software too. En P. Tarr y A. Wolf (eds.), *Engineering of software* (pp. 323-344). Springer. https://doi.org/10.1007/978-3-642-19823-6_17
- Pedreira, O., Piattini, M., Luaces, M. R. y Brisaboa, N. R. (2007). A systematic review of software process tailoring. *ACM SIGSOFT Software Engineering Notes*, 32(3), 1-6. <https://doi.org/10.1145/1241572.1241584>
- Prodan, M., Prodan, A. y Purcarea, A. A. (2015). Three new dimensions to people, process, technology improvement model. En A. Rocha, A. Correia, S. Costanzo y L. Reis (eds.), *New contributions in information systems and technologies: Advances in intelligent systems and computing* (pp. 481-490). Springer. https://doi.org/10.1007/978-3-319-16486-1_47
- Raman, S. (2000). It is software process, stupid: Next millennium software quality key. *IEEE Aerospace and Electronic Systems Magazine*, 15(6), 33-37. <https://doi.org/10.1109/62.847929>
- Ruiz-González, F. y Canfora, G. (2004). Software process: Characteristics, technology and environments. *SPT Software Process Technology*, 5, 6-10.
- Software Engineering Institute. (2010). *CMMI for development, CMMI-DEV version 1.3. CMU/SEI-2010-TR-033*. https://insights.sei.cmu.edu/documents/87/2010_019_001_28782.pdf
- Sommerville, I. (2015). *Software engineering: Always learning*. Pearson.
- Suri, D. y Sebern, M. J. (2004). Incorporating software process in an undergraduate software engineering curriculum: Challenges and rewards. En *17th Conference on Software Engineering Education and Training, 2004. Proceedings* (pp. 18-23). IEEE. <https://doi.org/10.1109/CSEE.2004.1276505>
- Sutton, S. M. (2000). The role of process in software start-up. *IEEE Software*, 17(4), 33-39.
- Wieczorek, M., Vos, D. y Bons, H. (2014). *Systems and software quality*. Springer. <https://doi.org/10.1007/978-3-642-39971-8>
- Yang, D. y Xue, M. (2011). Software process paradigm and its constraint mechanisms. En *2011 IEEE 2nd International Conference on Software Engineering and Service Science* (pp. 842-845). IEEE. <https://doi.org/10.1109/ICSESS.2011.5982472>

Procesos tradicionales de software

► Objetivos

Los objetivos de este capítulo consisten en obtener una panorámica global de los modelos de procesos tradicionales de software. Al estudiarlo, el lector:

- Entenderá el concepto de modelo de ciclo de vida de desarrollo de software.
- Comprenderá tres tipos genéricos de modelos de ciclo de vida de desarrollo de software.
- Aprenderá a seleccionar modelos de ciclo de vida de desarrollo adecuados con base en los criterios específicos de un proyecto de software a desarrollar.

A mediados de la década de 1960, se propusieron las metodologías de desarrollo de software como una respuesta organizada y estandarizada a un conjunto anticipado de tareas de desarrollo de software para mitigar las probabilidades de fallas en calidad, costo o tiempo (Sommerville, 1996).

Para garantizar el buen desarrollo de los proyectos, se han propuesto diferentes modelos de ciclo de vida, los cuales son marcos de referencia que contienen los procesos y las actividades involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software (International Organization for Standardization [ISO], 2008). Estos modelos de ciclo de vida también

se denominan procesos prescriptivos u orientados al plan, ya que hacen énfasis en que un plan se debe realizar al comienzo del proyecto y se debe seguir estrictamente a medida que el proyecto se está ejecutando.

Por consiguiente, un modelo de ciclo de vida de desarrollo de software incluye los procesos de software utilizados para especificar y transformar los requisitos de software en productos entregables de software (Bourque y Fairley, 2014). Un modelo de ciclo de vida típicamente incluye procesos clave de software dentro del modelo junto con sus interdependencias y relaciones temporales y lógicas.

Los modelos de ciclo de vida de desarrollo, para cubrir estas interdependencias y relaciones, incorporan mecanismos de control que aplican criterios de entrada y de salida (por ejemplo, revisiones o inspecciones); la salida de una actividad frecuentemente proporciona la entrada a otras actividades (por ejemplo, la especificación de requisitos como entrada del diseño arquitectónico); la ejecución paralela de varias actividades de un proceso puede producir salidas compartidas (por ejemplo, especificaciones de interfaces para múltiples componentes desarrollados por diferentes equipos).

El ciclo de vida se refiere al marco de tiempo que se extiende desde el desarrollo de un sistema de software hasta su eventual retiro. Es un proceso que inicia con el surgimiento de una idea, pasando por su implementación y finalizando con su retiro (Jirava, 2004). Debido a su especificidad, cada sistema de software requiere un *framework* que lo guíe para configurar, describir y monitorear el progreso del desarrollo durante todas las etapas del ciclo de vida.

La selección de un modelo de ciclo de vida de desarrollo se considera un aspecto estratégico que tendrá un gran impacto en la efectividad del sistema a largo plazo. La escogencia de un modelo equivocado puede acarrear retrasos en el proyecto, afectar la satisfacción del cliente y aun incidir en la cancelación del proyecto (ProjectSmart, 2008).

La naturaleza intangible del software permite una amplia variedad de modelos de ciclo de vida del desarrollo de software, entre ellos se encuentran (Bourque y Fairley, 2014):

- *Modelos lineales*: Las fases del desarrollo son ejecutadas secuencialmente con retroalimentación si es necesario, seguido de las actividades de integración, prueba y entrega de un único producto. Los modelos lineales son también llamados modelos de ciclo de vida de desarrollo predictivos.

- ▶ *Modelos iterativos*: El software es desarrollado en incrementos que aumentan la funcionalidad por medio de ciclos iterativos.
- ▶ *Modelos evolutivos*: Reconocen el grado de incertidumbre en ciertos proyectos y permiten a los desarrolladores realizar versiones parciales del proyecto mientras aprenden y adquieren información adicional y gradualmente evoluciona el diseño conceptual (Pressman, 2005).

A continuación, se presentará una panorámica global de los principales modelos de ciclo de vida de desarrollo de software, con el objeto de entender cuáles son los más apropiados para un proyecto específico.

▶ 2.1. Modelos lineales

Los modelos lineales desarrollan un conjunto completo de requisitos de software en la medida de lo posible durante la planificación del proyecto. Estos requisitos de software son rigurosamente controlados durante todo el proyecto. A su vez, los cambios en los requisitos son gestionados por un proceso de control de cambios.

2.1.1. Modelo en cascada

En sus inicios, el desarrollo de software consistía básicamente en construir y reparar los errores encontrados en el código. El modelo de procesos en cascada o ciclo de vida clásico (Royce, 1987) fue el primer modelo que presentó un marco de trabajo secuencial, que describe las etapas básicas que son obligatorias para un modelo de desarrollo exitoso de software (Azam *et al.*, 2010). El modelo fue aplicado en el diseño de un software para la misión de una nave espacial. Este es descrito como una secuencia de pasos unidireccionales, *top-down* y no iterativos. La esencia del modelo en cascada es que intenta proporcionar un conjunto útil de guías para el desarrollo de nuevos programas o sistemas.

El modelo en cascada está conformado por las siguientes fases: requisitos del sistema, requisitos de software, análisis, diseño del programa, codificación, pruebas y operación, como se observa en la figura 2.1. Todas las fases están orientadas a la ingeniería y no contienen otras actividades, como gestión o soporte. El modelo es secuencial (aunque la propuesta original incluía bucles de retroalimentación), así que los desarrolladores de software

deben ejecutar el modelo realizando cada fase una después de otra, por ello, se denomina en cascada. Solo se puede pasar a la siguiente fase una vez la fase anterior se haya finalizado completamente. Se debe realizar una planificación de todas las fases del proceso antes de iniciar un proyecto.

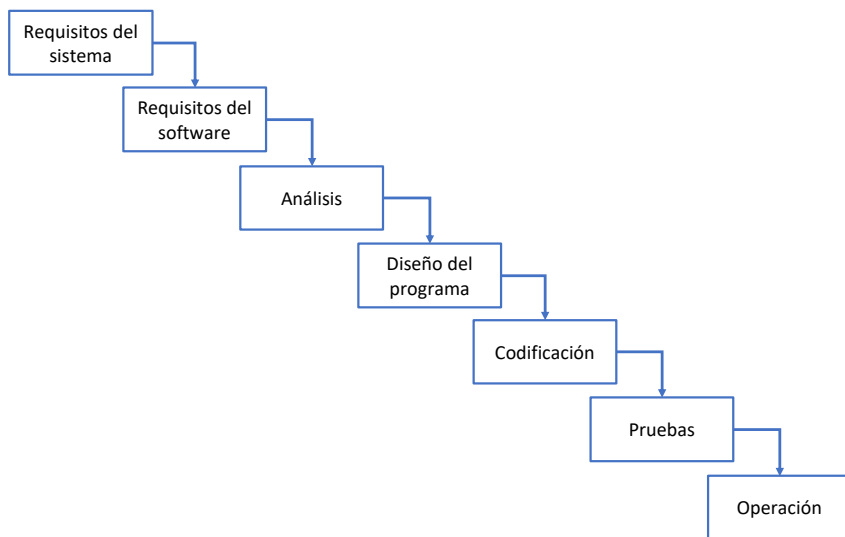


Figura 2.1. Modelo en cascada

Fuente: elaboración propia con base en Royce (1987).

Las fases del modelo en cascada son:

- *Requisitos del sistema:* Condiciones o capacidades que debe cumplir un sistema para satisfacer las necesidades de un usuario.
- *Requisitos de software:* Condiciones o capacidades que debe cumplir el software necesitado por un usuario para resolver un problema.
- *Análisis:* Se analizan las necesidades de los usuarios finales del producto de software para establecer qué objetivos se van a cumplir.
- *Diseño del programa:* Se establece una arquitectura global para el sistema a desarrollar describiendo sus abstracciones fundamentales.
- *Codificación:* Se implementa el diseño a través de un conjunto de programas.
- *Pruebas:* Se verifica que el sistema de software cumple su especificación.
- *Operaciones:* El sistema se instala y se pone en práctica.

El modelo fue popular, y se modificó y adaptó de numerosas formas de acuerdo con diferentes estudios y contextos de aplicación. La denominación de cada fase cambia de acuerdo con el contexto específico de aplicación.

Sus grandes fortalezas se fundamentan en incorporar hábitos positivos aceptados generalmente durante el desarrollo de software, como planificación temprana, amplia documentación del proceso y conceptos robustos de diseño, antes de comenzar a codificar (Munassar y Govardhan, 2010).

Este modelo se puede aplicar con éxito cuando:

- ▶ Las especificaciones del producto de software están bien definidas y son razonablemente estables, el producto es bastante maduro o el dominio del problema es bien conocido.
- ▶ Se puede aplicar con éxito en proyectos pequeños con pocos cambios en los requisitos.
- ▶ A pesar de no ser flexible, fue popular como una base conceptual para otros *frameworks* y modelos.

Algunos inconvenientes del modelo en cascada son (Azam *et al.*, 2010; Boehm, 1996; Royce, 1987):

- ▶ Los procedimientos que incluye son inflexibles y el diseño es bastante rígido.
- ▶ La gestión de cambios en los requisitos se hace difícil, ya que se restringen los movimientos hacia fases anteriores o posteriores del modelo.
- ▶ Si no se tiene un conjunto completo de requisitos, el resto de fases del proceso se realizarán también en forma incompleta. Luego, si las fases se terminan, moverse hacia etapas anteriores y reconstruirlas conlleva una sobrecarga de trabajo.
- ▶ Cuando una fase finaliza, no se repite de nuevo y se pasa a la siguiente. Pasar a fases anteriores no se permite. Esta linealidad no corresponde con la realidad de los proyectos de software.
- ▶ Las fechas de entrega son difíciles de cumplir en proyectos grandes.
- ▶ Como la implementación es una etapa final, en proyectos grandes el cliente comienza a revisar productos funcionales en fechas tardías.
- ▶ Los defectos encontrados en el software son costosos de reparar.
- ▶ El modelo está orientado a la obtención de documentos, los cuales se consideran criterios de finalización de cada fase.

2.1.2. Modelo en V

El modelo en V (Tian, 2005) es una variante del modelo en cascada presentado por Paul Rock a finales de la década de 1980. Hace énfasis en la conexión existente entre cada una de las etapas del proceso de desarrollo y sus respectivas etapas de pruebas. Al enfocarse en estas relaciones, asegura que las medidas adecuadas de calidad y pruebas son aplicadas a lo largo de todo el ciclo de vida.

El modelo en cascada se dobla hacia arriba después de la fase de codificación para formar la figura de una V, como se observa en la figura 2.2. Para cada una de las fases en la mitad izquierda de la figura, hay una correspondencia con fases de la parte derecha de esta. Las fases de la parte derecha ascendente de la V son actividades que permiten probar y validar el trabajo realizado en las fases de la parte izquierda.

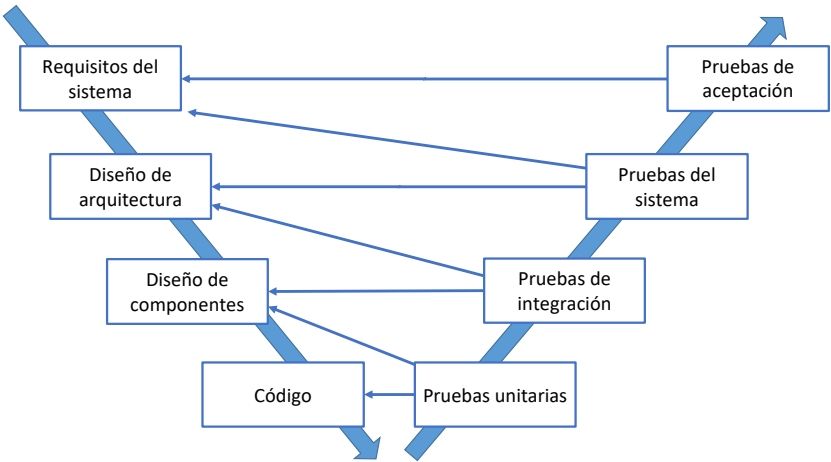


Figura 2.2. Modelo en V

Fuente: elaboración propia con base en Sharma y Singh (2015).

Con la realización de pruebas constantes y su respectiva documentación, es posible incrementar la eficiencia total del proceso debido a que eventuales problemas pueden ser detectados y resueltos tempranamente (Mathur y Malik, 2010).

No hay diferencias significativas entre el modelo en cascada y el modelo en V, solo cambia su representación, ya que el modelo sigue siendo lineal-secuencial, pero muestra las fases de pruebas de una forma más de-

tallada para validar las actividades previas. Este modelo enseña cómo diferentes tipos de pruebas están relacionadas con varias fases del desarrollo de software.

El objetivo principal del modelo es ilustrar la importancia de las relaciones entre tareas de desarrollo y pruebas. Debido a que el modelo resuelve sus errores poco después de ser identificados, se hace menos costoso resolverlos. El hecho de que las pruebas se realizan desde el principio del proceso aumenta su eficiencia (Isaias y Iss, 2014).

El modelo en V presenta los siguientes inconvenientes (Azam *et al.*, 2010; Isaias e Iss, 2014):

- ▶ Se dirige al desarrollo de software dentro de un proyecto en lugar de toda una organización.
- ▶ No es tan completo como lo argumenta ser, ya que abarca todas las actividades a un nivel demasiado abstracto.
- ▶ Es rígido y tiene poco espacio para adaptaciones flexibles, cualquier modificación de los requisitos hará la documentación y pruebas obsoletas.
- ▶ Como requiere grandes recursos, es adecuado solo para grandes proyectos dentro de grandes organizaciones.

▶ 2.2. Modelos iterativos

Los modelos iterativos producen software entregable mediante incrementos sucesivos de trabajo. El modelo está basado en la fragmentación de los requisitos de software a implementarse en cada iteración. Los requisitos son rigurosamente controlados o presentan alguna flexibilidad en su revisión a medida que los incrementos de software se van desarrollando. El modelo iterativo más conocido y utilizado es el modelo de desarrollo incremental.

2.2.1. Desarrollo incremental

El modelo de desarrollo incremental (Sommerville, 1996) se centra en descomponer el alcance del trabajo en partes más pequeñas permitiendo un proceso más flexible y requiriendo una planificación menos amplia. Los requisitos de software deben especificarse al comienzo del desarrollo. A continuación, un conjunto limitado de requisitos es asignado a cada incremento. Los requisitos funcionales de mayor prioridad se entregan primero. Cada

incremento libera un conjunto de requisitos en forma sucesiva hasta que la entrega final satisface todos los requisitos, como se observa en la figura 2.3.

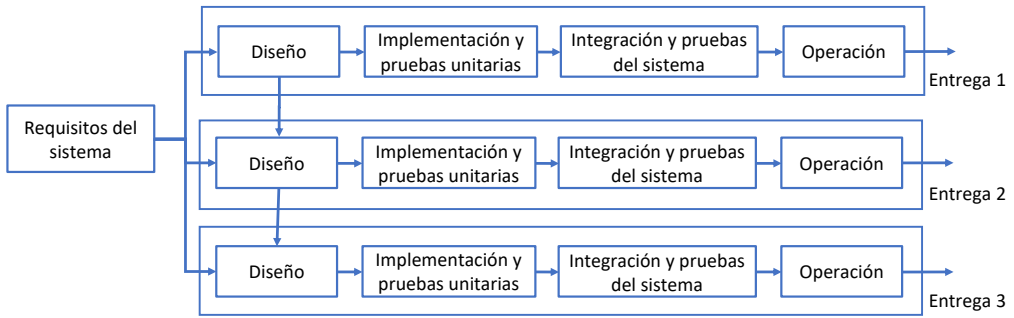


Figura 2.3. Modelo de desarrollo incremental

Fuente: elaboración propia con base en Tilloo (2013).

Este modelo aplica secuencias lineales de manera escalonada a medida que avanza la ejecución de un proyecto de software. Cada incremento es una unidad funcional de software, la cual es susceptible de ser entregada al cliente (Pressman, 2005). Una vez un incremento ha sido desarrollado, se somete a evaluación por parte de este. Como resultado de la evaluación, se obtiene un plan para el siguiente incremento.

El proceso es realizado por medio de esfuerzos más pequeños, estos incrementos permiten mejorar la funcionalidad del producto y pueden ser monitorizados, de tal manera que el progreso puede medirse hasta que se termina el producto final.

Entre las ventajas de este modelo se encuentran:

- En cada incremento se entrega un producto que funciona.
- El cliente puede ver software funcionando desde etapas tempranas, de tal manera que puede evaluarlo y ofrecer retroalimentación inmediata a los desarrolladores.
- Es útil cuando no se cuenta con personal disponible para la implementación completa de funcionalidades en los plazos establecidos.
- La planificación de los incrementos puede incluir riesgos técnicos que permitirán mitigarlos o prevenirlos planificándolos en incrementos posteriores adecuados.
- Es más barato y fácil realizar cambios en el software a medida que se vaya diseñando.

- ▶ Permite la incorporación de necesidades que no han sido obvias al inicio del proceso.
- ▶ Aunque el producto no se entrega rápidamente, es más fácil probarlo y eventualmente corregirlo.

Las desventajas del modelo son (Madachy, 2008; May y Zimmer, 1996; Sommerville, 1996; Tilloo, 2013):

- ▶ Un alcance mal definido puede variar el producto de incremento a incremento.
- ▶ Pueden existir sobrecargas de trabajo cuando hay un cambio de contexto rápido entre varias actividades.
- ▶ Cada iteración es seguida de una evaluación que asegura que los requisitos del usuario han sido cumplidos. Esta evaluación después de cada iteración puede consumir mucho tiempo.
- ▶ Al desarrollarse rápidamente el software, no es factible en costos producir documentos para cada versión del sistema. Por tanto, el proceso no es fácilmente visible.
- ▶ La estructura general de sistema se degrada con cada incremento. Se deben aplicar técnicas de refactorización para mejorar la estructura del sistema.
- ▶ Puede ser más costoso desarrollar y entregar múltiples versiones del producto.
- ▶ Cuando un último incremento es desarrollado debido a nuevos problemas encontrados, puede tener problemas de compatibilidad con versiones anteriores del producto.

Algunos ejemplos de modelos de desarrollo incremental son:

- ▶ Modelo de desarrollo paralelo (Wallin y Land, 2005): Los diferentes subsistemas se desarrollan al mismo tiempo, pero por equipos diferentes, como se observa en la figura 2.4. Si hay suficientes recursos disponibles, puede disminuir el tiempo calendario necesario para el desarrollo. Por el contrario, pueden surgir riesgos de integración si el tiempo aumenta.
- ▶ Modelo de sala limpia (Mills *et al.*, 1987): Se hace mucho énfasis en prevenir defectos en lugar de corregirlos, desarrollando software a través de prácticas enmarcadas en el control estadístico de la calidad y entregando software con una alta fiabilidad. Cada incremento se especifica formalmente, como se observa en la figura 2.5.

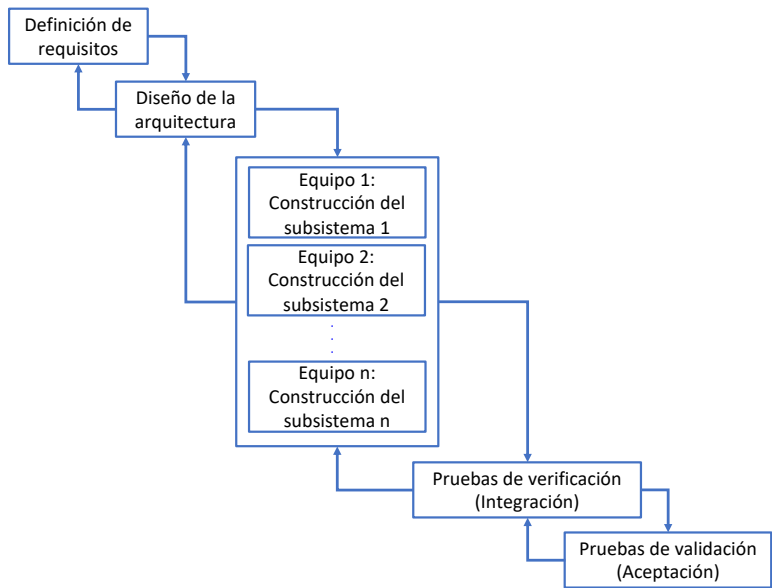


Figura 2.4. Modelo de desarrollo paralelo

Fuente: elaboración propia con base en Wallin y Land (2005).

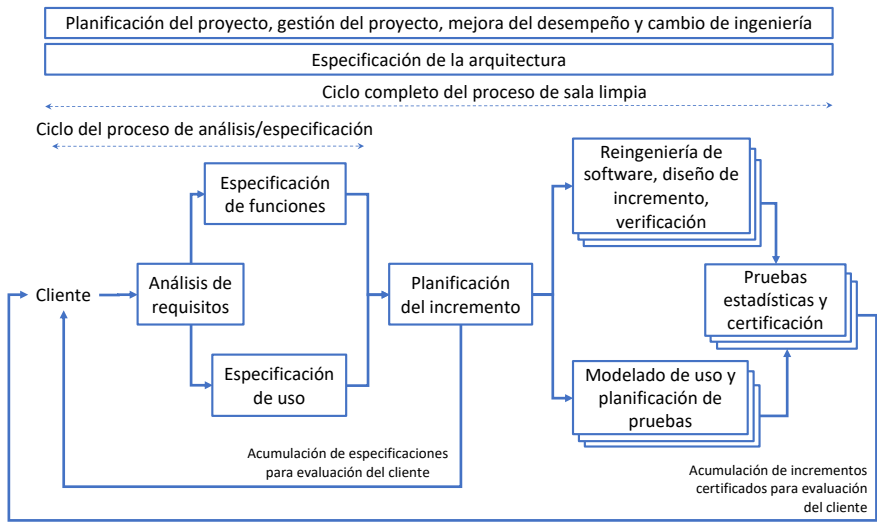


Figura 2.5. Flujo de proceso de sala limpia

Fuente: elaboración propia con base en Mills *et al.* (1987).

► 2.3. Modelos evolutivos

En los modelos evolutivos, se identifican las necesidades del usuario y se permiten cambios en las expectativas a medida que transcurre el desarrollo. Los proyectos que utilizan estos modelos se caracterizan por un alto riesgo tecnológico. Son efectivos en ambientes de cambio intensivo o donde hay una fuerte resistencia al cambio. Entre los modelos evolutivos, se encuentran el desarrollo de prototipos, el modelo en espiral, el modelo en espiral *win-win*, el desarrollo basado en componentes y el desarrollo rápido de aplicaciones.

2.3.1. Desarrollo de prototipos

El enfoque basado en prototipos es un *framework* aplicado desde la década de 1980. Carr y Verner (1997) intentaron resumir los modelos de prototipo en un *framework* más consistente. Su modelo está basado en la idea de crear parte o enteramente un sistema en una versión piloto, llamada prototipo. El objetivo es construir software en varias versiones y refinar consistentemente esas versiones hasta alcanzar un producto final.

En algunos proyectos de desarrollo, se presentan dudas; como la idea del producto es general, los requisitos no son claros, el sistema operativo no es conocido, no se ha decidido aún el lenguaje de programación o no se conoce cómo será la interfaz de usuario. Para estos casos, el desarrollo de prototipos es una buena opción (Pressman, 2005). Es un enfoque centrado en el usuario en el que su retroalimentación es fundamental para desarrollar subsecuentes prototipos (Sabale y Dani, 2012). El desarrollo de prototipos debe ser rápido, controlando los costos y que los usuarios experimenten tempranamente el sistema.

El proceso de desarrollo de prototipos se observa en la figura 2.6. El proceso comienza con la recolección de los principales requisitos funcionales, los cuales son identificados y analizados. Sigue un rápido diseño que continúa con el desarrollo del prototipo. Luego, este es evaluado por los usuarios y clientes, que pueden probarlo y proporcionar retroalimentación y experiencia en tiempo real. Los desarrolladores modifican el prototipo hasta que no haya cambios sustanciales o actualizaciones y el cliente y los usuarios estén satisfechos.

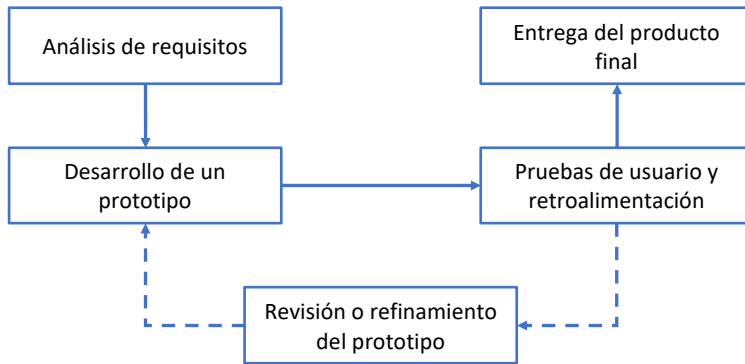


Figura 2.6. Prototipos

Fuente: elaboración propia con base en Carr y Verner (1997).

Existen dos tipos de prototipos: evolutivos y desechables (Capers, 2012). Los prototipos evolutivos añaden más características y gradualmente evolucionarán hasta convertirse en un producto finalizado, y los prototipos desechables, que pueden ser descartados una vez hayan servido para un propósito. Estos prototipos pueden ser más seguros debido a que los evolutivos pueden tener bajo control de la calidad y bajo rendimiento.

Otra tipología de prototipos es la propuesta por Floyd (1984), que se resumen en tres categorías: exploración, experimentación y evolutivos:

- *Enfoque exploratorio*: Los requisitos son explorados en cada iteración. Se realiza un prototipo rápido que permita explorar las necesidades y los requisitos de cada versión, evaluarlos con su uso y pruebas correspondientes, que se seguirán perfeccionando en las siguientes versiones.
- *Enfoque experimental*: Se entregan las características más esenciales del producto para que el usuario tenga una idea de su apariencia. Se evalúa a través de la aplicación experimental.
- *Enfoque evolutivo*: Cada prototipo es una versión del producto y cada versión sirve como prototipo a la siguiente.

El modelo de prototipos es apropiado para ambientes dinámicos en los que los requisitos cambian rápidamente. Entre algunos de sus beneficios se encuentran (Pressman, 2005; Sabale y Dani, 2012; Sommerville, 1996):

- Hacen mayor énfasis en el desarrollo de software que en la documentación.

- ▶ Proporcionan una mayor satisfacción del usuario, ya que sirven para pruebas iniciales de conceptos y los usuarios pueden observar software real funcionando desde etapas tempranas.
- ▶ Los marcos de tiempo de entrega son cortos. Los desarrolladores construyen software de inmediato y con rapidez.
- ▶ Ayudan y resuelven dudas durante la selección y validación de requisitos. Se adaptan a los cambios de requisitos debido a su constante retroalimentación por parte de los usuarios.
- ▶ Ayudan a evaluar la factibilidad en cuanto a soluciones específicas de diseño.
- ▶ El prototipo sirve como “versión inicial” y lo más probable es que sea descartado, aunque algunos pueden evolucionar para transformarse en el sistema real.
- ▶ Presentan un bajo costo de desarrollo, riesgos bajos y altas probabilidades de éxito.

Algunas limitaciones del desarrollo por prototipos son (Boehm, 1988; Boehm y Hansen, 2000; Madachy, 2008; Pressman, 2005):

- ▶ Carencia de información sobre el número exacto de iteraciones y el periodo requerido para actualizar el prototipo, y así satisfacer a usuarios y clientes.
- ▶ Debido a la premura por cumplir fechas de entrega, los desarrolladores no consideran todas las funcionalidades del prototipo al no realizar actividades específicas de análisis. Además, conlleva selecciones inadecuadas de sistemas operativos o lenguajes de programación.
- ▶ Los prototipos carecen de atributos clave, como calidad, facilidad de mantenimiento, seguridad, tolerancia a fallos, procesamiento distribuido, etc. Además, son débiles en aspectos relacionados con el análisis y diseño.
- ▶ Los usuarios pueden pensar que el prototipo desarrollado es el producto de software, proponiendo algunas modificaciones para convertirlo en el producto final, sin darse cuenta de que es un prototipo que requiere aún de mucho esfuerzo para convertirlo en un producto de calidad.

Lo fundamental con este enfoque es que los prototipos sirvan para la definición de requisitos y luego se deben aplicar técnicas rigurosas de ingeniería de software; su aplicación es ideal en proyectos grandes y particularmente centrados en los usuarios.

2.3.2. Modelo en espiral

El modelo en espiral (Boehm, 1988) se remonta a finales de la década de 1980. En lugar de presentar las fases de desarrollo de manera secuencial, lo hace en forma de espiral, como se observa en la figura 2.7. El modelo combina características del modelo en cascada y prototipos.

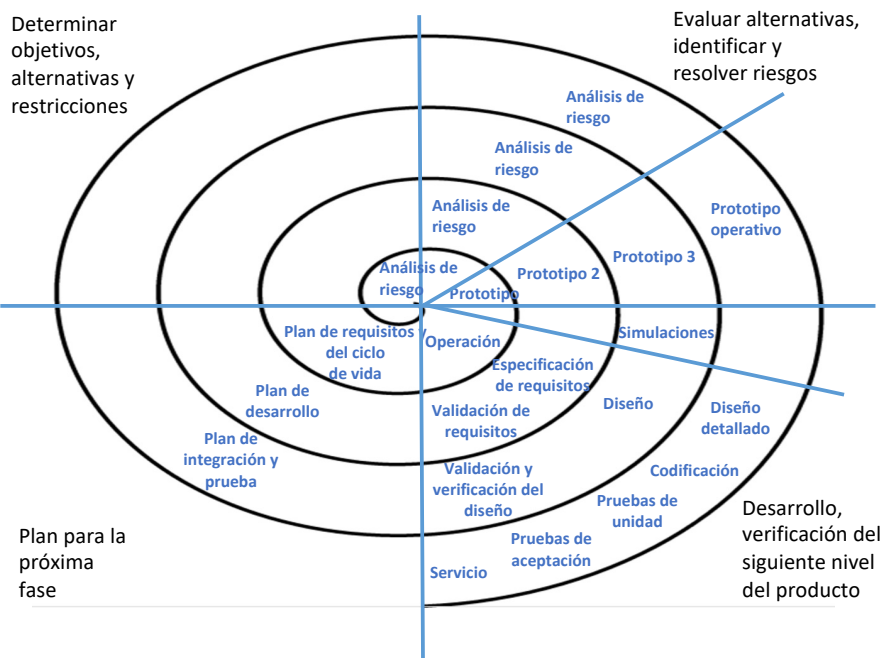


Figura 2.7. Modelo en espiral

Fuente: elaboración propia con base en Boehm (1988).

El modelo en espiral es un modelo evolutivo, el proceso inicia en el centro de la espiral y luego se mueve en el sentido de las agujas del reloj. Después se mueve en recorridos transversales de la espiral hacia el exterior, donde cada uno de ellos, una porción del proceso, es ejecutado obteniéndose un entregable. El proceso se va moviendo por cada segmento hasta finalizarlo (Emami *et al.*, 2010).

La espiral del modelo está dividida en regiones de tareas:

- ▶ *Planificación*: Los requisitos de software son determinados en esta fase.
- ▶ *Análisis de riesgo*: Se identifican riesgos que pueden afectar el proyecto y las posibles alternativas de solución. Un prototipo puede producirse al finalizar esta fase.
- ▶ *Ingeniería*: Se produce el software y se realizan las pruebas correspondientes.
- ▶ *Evaluación del cliente*: El usuario evalúa los resultados del proyecto antes que el proyecto continúe la siguiente sección de la espiral.

La gestión del riesgo se puede considerar el componente central del modelo. Cada fase es controlada por un criterio específico de gestión del riesgo que asegura la toma de decisiones utilizando factores críticos.

Boehm y Hansen (2000) definen para cada ciclo del proceso seis características particulares denominadas invariantes:

- ▶ *Invariante 1*: Definición de artefactos clave, como conceptos, requisitos, plan, diseño y codificación. Si la definición es secuencial, el proyecto puede tener preconcepciones excesivamente rígidas.
- ▶ *Invariante 2*: Cada ciclo debe seguir las cuatro regiones de tareas. Si no se sigue este esquema, puede afectarse negativamente el proceso entero.
- ▶ *Invariante 3*: El nivel de esfuerzo está determinado por las consideraciones de riesgo de cada actividad.
- ▶ *Invariante 4*: El grado de detalle es conducido por consideraciones del riesgo.
- ▶ *Invariante 5*: Utilizar hitos denominados “Objetivos del ciclo de vida”, “Arquitectura del ciclo de vida” y “Capacidad operacional inicial” en los que el cliente revisará los artefactos clave de cada etapa.
- ▶ *Invariante 6*: El desarrollo debe enfocarse en el ciclo de vida total considerando objetivos de los *stakeholders* y artefactos utilizados.

Entre los beneficios del modelo se encuentran (Boehm y Hansen, 2000; ProjectSmart, 2008):

- ▶ Proporciona mecanismos para incorporar objetivos de calidad de software.
- ▶ Se enfoca en la eliminación de errores y alternativas no atractivas tempranamente.
- ▶ Evalúa qué tantos recursos o actividades son suficientes.

- Se enfoca en la gestión del riesgo en todas las etapas del proceso. Por ello, es un modelo ideal para proyectos grandes de misión crítica.

Algunas desventajas del modelo son (Madachy, 2008):

- No se especifica ninguna técnica de análisis de riesgo. Si este es pobre, también lo será el producto final. La gestión del riesgo requiere experiencia y necesita adaptarse a las necesidades del producto.
- No es fácil ajustar el modelo a fechas contractuales de entrega.
- Es de vital importancia tener experiencia en análisis de riesgos.
- No es aplicable a proyectos pequeños en los que la gestión del riesgo puede incrementar los costos del desarrollo.

2.3.3. Modelo espiral *win-win*

El modelo espiral *win-win* (Boehm y Hansen, 2000) utiliza el enfoque de la teoría W (*win-win*). En este modelo, el desarrollador y el cliente negocian los requisitos como un conjunto satisfactorio de objetivos, restricciones y alternativas. Estos requisitos estarán basados en la funcionalidad, el desempeño, costo, tiempo, etc. La mejor negociación se esfuerza por obtener un resultado en el que ambas partes ganen, tanto el desarrollador como el cliente.

El análisis detallado del riesgo impone muchas restricciones, objetivos y alternativas que consumen mucho tiempo. Pero nunca los riesgos determinados varían de proyecto a proyecto (Boehm, 1988).

La figura 2.8 ilustra el modelo espiral *win-win*. Las seis secciones de la espiral son:

- Identificar los clientes del siguiente nivel.
- Identificar las condiciones ganadoras de los clientes.
- Reconciliar las condiciones ganadoras.
- Establecer los objetivos, las restricciones y alternativas.
- Evaluar alternativas de procesos y productos, y resolver riesgos.
- Definir el siguiente nivel del producto y proceso que incluye particiones.
- Validar definiciones de proceso y producto.

Al finalizar un ciclo de la espiral, se revisan los compromisos adquiridos en el nivel.

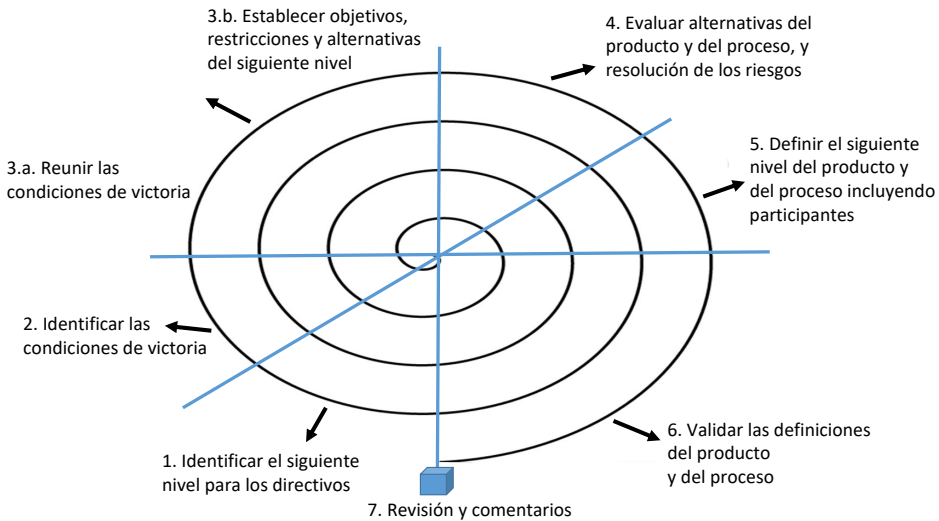


Figura 2.8. Modelo espiral win-win

Fuente: elaboración propia con base en Boehm y Hansen (2000).

2.3.4. Desarrollo basado en componentes

En el desarrollo de software actual, el producto puede llegar a ser tan complejo que es prácticamente imposible para una compañía construir enteramente el producto por sí misma. La especificación del sistema de software contiene especificaciones de subsistemas o componentes implementados en otros sistemas. Por tanto, se promueve utilizar una estrategia de reúso de componentes (Wallin y Land, 2005).

Los componentes comerciales de software (Commercial-Off-The-Shelf [COTS]) son fragmentos de software prefabricados que se integran al producto de software que se está desarrollando. La estrategia consistirá en encontrar e integrar componentes que satisfagan los requisitos definidos del sistema.

El ciclo de desarrollo de un sistema basado en componentes se observa en la figura 2.9. Las etapas del proceso son (Crnkovic y Larsson, 2001):

1. Búsqueda y selección de componentes que se ajusten a los requisitos del producto.

2. Creación de un componente propietario que será utilizado en el producto.
3. Adaptación de los componentes seleccionados al modelo de componentes existente o la especificación de requisitos.
4. Despliegue del producto a través de una infraestructura para componentes.
5. Reemplazo de las antiguas versiones del producto por las nuevas.

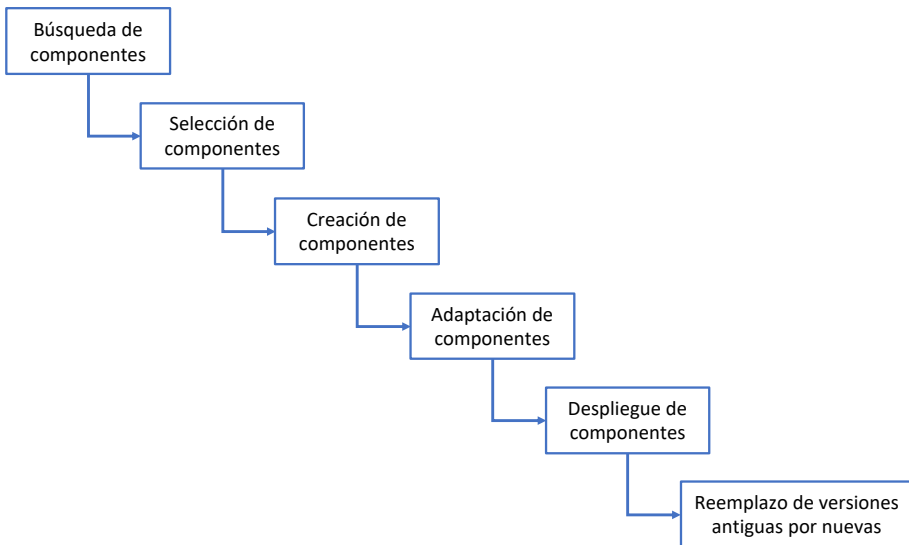


Figura 2.9. Modelo de desarrollo basado en componentes

Fuente: elaboración propia con base en Crnkovic y Larsson (2001).

Existen en la literatura dos enfoques orientados al reúso de componentes (Jurgens, 2009):

- *Desarrollo con reúso:* Hace énfasis en el desarrollo de nuevo software reutilizando componentes de software disponibles.
- *Desarrollo para reúso:* Se enfoca en el desarrollo de componentes para reutilizar en otras aplicaciones o proyectos futuros.

Entre los beneficios del uso de este modelo, se encuentran (Banker *et al.*, 1993; Henninger, 1997):

- Se reduce el ciclo de tiempo de desarrollo al igual que el costo del proyecto.

- ▶ La calidad de los componentes desarrollados es conocida y está inmediatamente disponible.
- ▶ Los detalles de implementación de los componentes están ocultos. La implementación de los componentes puede cambiar sin afectar al sistema.
- ▶ Los componentes son independientes entre sí y no dependen de otros.
- ▶ La comunicación entre los componentes y el sistema desarrollado es a través de interfaces bien definidas.

Algunos problemas que se presentan cuando se trabaja con este modelo son (Crnkovic y Larsson, 2001):

- ▶ *Confianza en los componentes*: ¿Cómo confiar en un componente, del cual no se tiene el código fuente?
- ▶ *Certificación de componentes*: ¿Quién certifica la calidad de los componentes?
- ▶ *Gestión de requisitos y selección de componentes*: Es difícil encontrar componentes que satisfagan completamente los requisitos del sistema. Además, los requisitos pueden cambiar con el tiempo y los componentes seleccionados quedarían obsoletos.

2.3.5. Desarrollo rápido de aplicaciones

Aunque el desarrollo rápido de aplicaciones fue conceptualizado en la década de 1970, el modelo fue formalizado por James Martin en los inicios de 1990. Su premisa básica es que se puede entregar software rápidamente manteniendo altos estándares de calidad (Isaias e Iss, 2014).

El modelo de desarrollo rápido de aplicaciones (Pressman, 2005) es una adaptación del modelo de desarrollo basado en componentes y también un modelo de desarrollo incremental que hace énfasis en ciclos cortos de desarrollo. Se puede aplicar con éxito si el alcance del proyecto está bien delimitado y los requisitos están bien entendidos.

El modelo comprende un conjunto de herramientas y guías en un marco de tiempo. El producto evoluciona en incrementos sucesivos cumpliendo las prioridades establecidas por las necesidades del negocio (Gottesdiener, 1995). Entre las herramientas y guías para su aplicación efectiva, están el modelado de procesos y datos, la generación de código, pruebas y métodos de planificación (Agarwal *et al.*, 2000).

El desarrollo rápido de aplicaciones sigue un ciclo de cuatro etapas (Agarwal *et al.*, 2000): planificación de requisitos, diseño de usuario, construcción y transición (figura 2.10).

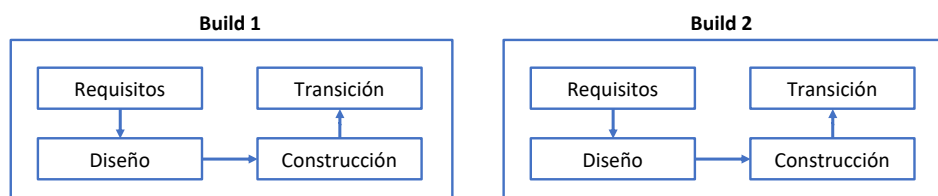


Figura 2.10. Modelo de desarrollo rápido de aplicaciones

Fuente: elaboración propia con base en Isaías y Iss (2014).

En la planificación de requisitos, se establecen los objetivos y el alcance, se analizan los requisitos y se establece el marco de tiempo, *timebox*, periodo fijo durante el cual un incremento específico es desarrollado. Cada incremento se desarrolla utilizando el modelo espiral que incluye diseño, prototipos y pruebas.

El beneficio fundamental de la utilización de este modelo es su enfoque en la entrega rápida y una eficaz comunicación desarrollador-cliente (Isaías e Iss, 2014).

Algunas desventajas del modelo son (Pressman, 2005):

- Las características funcionales entregadas son reducidas o básicas debido a los pequeños marcos de tiempo de entrega.
- La escalabilidad se reduce porque el software desarrollado comienza como un prototipo y evoluciona hacia una aplicación terminada utilizando componentes existentes que se integran.
- Presenta altos riesgos técnicos debido a las nuevas tecnologías en un mercado de software global que cambia.

► 2.4. Selección de modelos de ciclo de vida

La selección cuidadosa de un modelo de ciclo de vida de desarrollo de software adecuado para un proyecto no es una tarea fácil. Una forma de hacer tal selección es realizar *benchmarkings* de aplicaciones que utilizan

varios modelos y luego seleccionar el modelo que permita los mejores resultados para tamaños y tipos específicos de proyectos.

Además, los elementos de diferentes modelos de ciclo de vida de desarrollo de software pueden incorporarse en un modelo específico. Por ejemplo, el modelo de desarrollo incremental puede incorporar aspectos lineales-secuenciales durante las fases de especificación de requisitos y diseño, pero puede permitir flexibilidad en la revisión de los requisitos y en el diseño arquitectónico durante la construcción de software.

También se debe considerar una *checklist* de elementos que necesitan evaluarse cuando se selecciona un modelo. Entre los elementos a evaluar se encuentran (Capers, 2012):

- ▶ *Idoneidad para el tamaño de la aplicación:* Los métodos secuenciales son apropiados para proyectos grandes y estandarizados.
- ▶ *Idoneidad para el tipo de aplicación:* Se debe estudiar qué tan bien el modelo de proceso funciona de acuerdo con su tipo. Por ejemplo, para software embebido, aplicaciones web, aplicaciones comerciales, juegos, software militar, etc.
- ▶ *Idoneidad de acuerdo con la naturaleza de la aplicación:* Se estudia qué tan bien el modelo de proceso funciona para nuevos desarrollos, mejoras, reparaciones de garantía y renovación de aplicaciones heredadas.
- ▶ *Idoneidad por atributos:* Qué tan bien el modelo soporta atributos de software específicos, como prevención de defectos, eficiencia de eliminación de defectos, vulnerabilidades de seguridad, desempeño óptimo e interfaces de usuario óptimas, entre otras. Por ejemplo, un modelo que no incluya medición y control de calidad no es apropiado para sistemas críticos.
- ▶ *Idoneidad por actividad:* Qué tan bien el modelo soporta actividades puntuales, como requisitos, arquitectura, diseño, codificación, reusabilidad, inspecciones, análisis estático, pruebas, gestión de configuración, aseguramiento de la calidad, mantenimiento, mejora y soporte al cliente.

Cada modelo de proceso descrito en este capítulo tiene sus fortalezas y debilidades. En la tabla 2.1, se presenta un listado de características de cada modelo que permite realizar una comparación entre ellos y facilita su selección de acuerdo con las características y propiedades de un proyecto de software.

Tabla 2.1. Características de los modelos de ciclo de vida de desarrollo de software

Característica	Cascada	Modelo V	Incremental	Prototipo	Espiral	win-win	RAD	Componente
Especificación de requisitos	Inicio	Inicio	Inicio	Tarde	Inicio	Inicio	Inicio	Inicio
Control de recursos	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Complejidad	Simple	Intermedia	Intermedia	Simple	Alta	Alta	Intermedia	Alta
Análisis de riesgos	No	Solo al inicio	No	No	Sí	Sí	Bajo	No
Involucra al usuario	Inicio	Intermedio	Alto	Alto	Alto	Alto	Inicio	Intermedio
Flexibilidad	No	Menos flexible	Sí	Sí	Sí	Sí	Sí	Sí
Costo	Bajo	Bajo	Alto	Alto	Alto	Alto	Bajo	Medio
Reusabilidad	Limitada	Sí	Sí	Baja	Sí	Sí	Sí	Alta
Entendimiento de requisitos	Sí	Sí	No	No	No	No	No	No
Solapamiento de actividades	No	Sí	Sí	Sí	Sí	Sí	No	No
Tiempo de implementación	Largo	Medio	Menos	Rápido	Largo	Largo	Rápido	Medio
Habilidades de gestión	No	No	Sí	No	Sí	Sí	Sí	Sí
Entrega temprana	No	No	Sí	No	No		Sí	Sí
Gestión del cambio	No	Sí, bajo	Sí	Sí	Sí	Sí	Sí	No
Calidad del producto	No	Sí, bajo	Sí	No	Sí	Sí	Sí	Sí, bajo
Tiempo de desarrollo	Largo	Medio	Corto	Corto	Largo	Largo	Corto	Corto
Tamaño del proyecto	Grande	Medio	Pequeño	Medio	Pequeño	Pequeño	Medio	Medio

Fuente: elaboración propia con base en Rastogi (2015) y Sharma y Singh (2015).

Algunos criterios clave a considerar para la selección son los requisitos, la fecha de entrega, la dimensión, la localización del equipo de desarrollo y recursos (Isaias e Iss, 2014).

Los requisitos de software son un criterio clave para la selección del modelo. Si los requisitos son estrictos e inmutables, se puede adoptar el modelo en cascada. Si no son claros desde el comienzo o cambian a menudo, es mejor trabajar con algún modelo iterativo o evolutivo.

Con respecto a la fecha de entrega, si el calendario es estricto, un modelo lineal, con amplia documentación y pruebas realizadas en etapas tardías, no es una opción recomendable.

Otro criterio es la dimensión del proyecto; cuanto más grande el proyecto, más rígido se vuelve el modelo de ciclo de vida.

Por otro lado, si la localización del equipo es geográficamente dispersa, un modelo en cascada es un buen enfoque por la claridad de las etapas y tareas. Para equipos que trabajan juntos, una clara comunicación es importante con modelos iterativos y evolutivos.

Finalmente, desde el punto de vista de los recursos, si los proyectos involucran complejas dinámicas y demandan un uso particular de experiencias y recursos, son más abordables con modelos que tengan una planificación estricta, como los modelos lineales.

► Desafíos

Los proyectos de software que utilizan procesos tradicionales se caracterizan por desarrollar un plan inicial, el cual se considera fundamental para evaluar posibles participaciones en licitaciones y negociar contratos antes de iniciar proyectos (Diebold y Zehler, 2016). La utilización de procesos concretos y estandarizados favorece la transparencia de los proyectos, mejora la gestión y aumenta las posibilidades de éxito. De esta manera, estos procesos ayudan a mejorar la predictibilidad, repetibilidad y optimización (Solinski y Petersen, 2016).

Los procesos tradicionales hacen énfasis en etapas concretas, como la arquitectura y el diseño, las cuales son relevantes en sistemas de software a gran escala (Petersen *et al.*, 2009). Además, son apropiados cuando los requisitos del producto se pueden especificar desde el principio y no se necesita involucrar a los usuarios finales durante el proyecto (Diebold y

Zehler, 2016; Špundak, 2014) y cuando el contexto de desarrollo implica una alta criticidad en la que los requisitos deben especificarse y analizarse tempranamente.

Generalmente, los enfoques de procesos tradicionales se aplican a grandes organizaciones, las cuales hacen énfasis en el control y la supervisión de los proyectos, en que es de vital importancia que el proyecto siga un plan sin desviaciones ni cambios bruscos (Špundak, 2014).

Sin embargo, los procesos tradicionales presentan algunos desafíos, como tener un contrato bien especificado y definido al inicio del proyecto, lo cual puede causar tensiones entre los desarrolladores y clientes. Los contratos no deben ser flexibles y sin muchas maniobras para adaptarse al cambio. Pueden ocurrir problemas cuando prima el cumplimiento del contrato sobre la necesidad de alcanzar buenos resultados.

En el modelo en cascada, las pruebas del software se realizan al finalizar el proyecto, y si hay retrasos o muchos defectos, el software estará comprometido por el retrabajo (Petersen *et al.*, 2009).

Otra crítica asidua a los procesos tradicionales es la excesiva documentación que se debe generar y que algunas veces no se utiliza, lo que conlleva un desperdicio de esfuerzo (Petersen y Wohlin, 2010). Por otra parte, los procesos tradicionales son difíciles de adaptar a proyectos pequeños e implica culturas y métodos excesivamente burocráticos que hacen que el cumplimiento de contratos pueda bloquear el desarrollo de software.

► Ejemplos

Las grandes organizaciones de desarrollo de software tienen sus propios procesos internos que están diseñados para sus necesidades específicas. Por ejemplo (Arms, 2022):

- *Amazon (empresa especializada en el comercio electrónico)*: Hace un uso extensivo del desarrollo ágil de software basado en *sprints*. La mayoría del desarrollo de software se divide en incrementos de aproximadamente cuatro semanas transcurridas.
- *Lockheed Martin (empresa contratista del Gobierno estadounidense)*: Sigue un proceso secuencial que encaja con la forma en que el Gobierno de Estados Unidos administra los contratos de software.

- ▶ **SAP** (*software empresarial*): Hace énfasis en la funcionalidad de sus clientes empresariales. Gran parte del desarrollo es adecuado para un proceso secuencial.
- ▶ **Microsoft** (*software para computadores personales*): Hace énfasis en la fase de pruebas con una variedad amplia de equipos y en la compatibilidad con versiones anteriores. Gran parte del desarrollo utiliza un proceso en espiral.

▶ Resumen

En este capítulo, se ha descrito una amplia variedad de modelos de ciclo de vida de desarrollo de software, cada uno de los cuales posee sus propias características para responder a necesidades específicas de los proyectos de software. El espectro de modelos se puede clasificar en modelos lineales, iterativos y evolutivos.

Los modelos lineales se basan en una secuencia de etapas que se desarrollan una después de otra, los requisitos de software se mantienen estáticos, se realiza una amplia documentación del proceso y se obtiene software funcionando para mostrar al cliente en etapas tardías. Ejemplos de estos modelos son el modelo en cascada y el modelo en V.

Los modelos iterativos se centran en desarrollar software en varios ciclos en los que cada uno de ellos agrega funcionalidad en forma progresiva hasta obtener el producto final completo. El desarrollo incremental es un ejemplo de esos modelos.

Por último, los modelos evolutivos están enfocados en gestionar el cambio constante en el software y en una definición cíclica de los requisitos. Ejemplos de esta tipología son el desarrollo de prototipos, el modelo en espiral (y su adaptación, el modelo *win-win*), el desarrollo rápido de aplicaciones y el desarrollo basado en componentes.

De esta panoplia de modelos de ciclos de vida del desarrollo de software se concluye que escoger cuál modelo adoptar para un proyecto particular puede ser una tarea compleja. No obstante, algunos aspectos y características particulares del proyecto pueden facilitar la selección del modelo. Su escogencia es una decisión crucial para la planificación del proyecto. La definición de modelos nuevos, adaptados o híbridos continuará siendo una actividad con mucho esfuerzo de estudio e investigación.

► Ejercicios

1. Presentar tres ejemplos de proyectos de software que pueden desarrollarse utilizando el modelo en cascada. Justifique su respuesta.
2. Presentar tres ejemplos de proyectos de software que pueden desarrollarse utilizando el modelo en espiral. Justifique su respuesta.
3. Presentar tres ejemplos de proyectos de software que pueden desarrollarse utilizando el modelo incremental. Justificar su respuesta.
4. Determinar el proceso de software a aplicar para los siguientes sistemas:
 - Un sistema orientado al *blockchain*, pero el equipo de desarrollo no tiene experiencia previa en este tipo de sistemas.
 - Un sistema de información académica de una universidad con unos requisitos de software bien definidos.
 - Una aplicación orientada al metaverso con requisitos del cliente por definir.
 - Un software de contabilidad que implemente unas nuevas directrices gubernamentales.
5. Especificar tres argumentos para la utilización del modelo basado en prototipos.
6. Especificar dos similitudes y diferencias entre el modelo **RAD** y el modelo basado en prototipos.
7. Se debe desarrollar un sistema de software empresarial para manejo de inventarios utilizando un estándar nuevo que la organización no conoce, el tiempo de desarrollo es corto, pero con altos estándares de calidad, el equipo de desarrollo es de tamaño pequeño (menos de diez personas). Para estos requerimientos, ¿cuál modelo de proceso de software recomienda y por qué?
8. ¿Es posible combinar varios modelos de proceso? Si su respuesta es afirmativa, presente un ejemplo.
9. ¿Por qué un proceso de software iterativo hace más fácil administrar las solicitudes de cambio en el software?

► Referencias

- Agarwal, R., Prasad, J., Tanniru, M. y Lynch, J. (2000). Risks of rapid application development. *Communications of the ACM*, 43(11), 177-188. <https://dl.acm.org/doi/pdf/10.1145/352515.352516>
- Arms, W. Y. (2022). *Examples of software development processes*. Cornell University Computing and Information Science. <https://www.cs.cornell.edu/courses/cs5150/2017sp/slides/3-process-examples.pdf>
- Azam, F., Gull, H., Bibi, S. y Amjad, S. (2010). Back and forth (BnF) software process model. En *2010 Second International Conference on Computer Engineering and Applications* (vol. 1, pp. 426-430). **IEEE**. <https://doi.org/10.1109/ICCEA.2010.89>
- Banker, R. D., Kauffman, R. J. y Zweig, D. (1993). Repository evaluation of software reuse. **IEEE Transactions on Software Engineering**, 19(4), 379-389. <https://doi.org/10.1109/32.223805>
- Boehm, B. W. (1996). Anchoring the software process. **IEEE Software**, 13(4), 73-82. <https://doi.org/10.1109/52.526834>
- Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5), 61-72. <https://doi.org/10.1109/2.59>
- Boehm, B. W. y Hansen, W. J. (2000). *Spiral development: Experience, principles and refinements. Special Report CMU/SEI-2000-SR-008*. Software Engineering Institute, Carnegie Mellon University. https://insights.sei.cmu.edu/documents/5439/2000_003_001_13655.pdf
- Boehm, B. W. y Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley.
- Bourque, P. y Fairley, R. E. (eds.) (2014). *Guide to the software engineering body of knowledge, version 3.0*. **IEEE Computer Society**. http://artemisa.unicauca.edu.co/~cardila/IS_SWEBOKv3.pdf
- Capers, J. (2012). *Software engineering best practices: Lessons from successful projects in the top companies*. McGraw-Hill.
- Carr, M. y Verner, J. (1997). *Prototyping and software development approaches*. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0b05add730e04843e-234937a070f24b19efaadc3>
- Crnkovic, I. y Larsson, M. (2001). *Component-based software engineering: New paradigm of software development*. Mälardalen University.
- Diebold, P. y Zehler, T. (2016). The right degree of agility in rich processes. En M. Kuhrmann, J. Münch, I. Richardson, A. Rausch y H. Zhang (eds.), *Managing software process evolution* (pp. 15-37). Springer. https://doi.org/10.1007/978-3-319-31545-4_2
- Emami, M. S., Ithnin, N. B. y Ibrahim, O. (2010). Software process engineering: Strengths, weaknesses, opportunities and threats. En *INC2010: 6th International Conference on Networked Computing* (pp. 1-5). **IEEE**.
- Floyd, C. (1984). A systematic look at prototyping. En R. Budde, K. Kuhlenkamp, L. Mathiasen y H. Züllighoven (eds.), *Approaches to prototyping* (pp. 1-18). Springer. https://doi.org/10.1007/978-3-642-69796-8_1
- Gottesdiener, E. (1995). RAD realities: Beyond the hype to how **RAD** really works. *Application Development Trends*, 2(8), 28-38.

- Henninger, S. (1997). An evolutionary approach to constructing effective software reuse repositories. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(2), 111-140. <https://doi.org/10.1145/248233.248242>
- International Organization for Standardization. (2008). *ISO/IEC 12207:1995/AMD2:2004. Information Technology - Software life cycle processes - Amendment 2*.
- Isaias, P. y Issa, T. (2015). *High level models and methodologies for information systems*. Springer. <https://doi.org/10.1007/978-1-4614-9254-2>
- Jirava, P. (2004). *System development life cycle*. <https://dk.upce.cz/bitstream/handle/10195/32471/CL456.pdf?sequence=1&isAllowed=y>
- Jurgens, D. (2009). *Survey on software engineering for scientific applications*. Institute for Scientific Computing. https://leopard.tu-braunschweig.de/servlets/MCRFileNodeServlet/dbbs_derivate_00006306/Juergens-Survey-Software-Eng-Scientific-Applications.pdf
- Madachy, R. J. (2008). *Software process dynamics*. John Wiley & Sons.
- Mathur, S. y Malik, S. (2010). Advancements in the V-Model. *International Journal of Computer Applications*, 1(12), 29-34. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=04aca97824d178d7ca3688bbed2118d0115dfaba>
- May, E. L. y Zimmer, B. A. (1996). The evolutionary development model for software. *Hewlett Packard Journal*, 47, 39-41. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5304a6d70439f180af1e349d518cb1d20b99e4a8>
- Mills, H. D., Dyer, M. y Linger, R. C. (1987). Cleanroom software engineering. *IEEE Software*, 4(5), 19-25. <https://doi.org/10.1109/MS.1987.231413>
- Munassar, N. M. A. y Govardhan, A (2010). Comparison between five models of software engineering. *International Journal of Computer Science Issues*, 7(5), 94-101. <https://www.ijcsi.org/papers/7-5-94-101.pdf>
- Petersen, K. y Wohlin, C. (2010). The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study. *Empirical Software Engineering*, 15, 654-693. <https://doi.org/10.1007/s10664-010-9136-6>
- Petersen, K., Wohlin, C. y Baca, D. (2009). The waterfall model in large-scale development. En F. Bomarius, M. Oivo, P. Jaring y P. Abrahamsson (eds.), *Product-Focused Software Process Improvement. PROFES 2009. Lecture Notes in Business Information Processing* (pp. 386-400). Springer. https://doi.org/10.1007/978-3-642-02152-7_29
- Pressman, R. S. (2005). *Software engineering: A practitioner's approach* (6.^a ed.). McGraw-Hill.
- ProjectSmart. (2008). *Which life cycle is best for your project?* <https://www.projectsmart.co.uk/agile-project-management/which-life-cycle-is-best-for-your-project.php>
- Rastogi, V. (2015). Software development life cycle models-comparison, consequences. *International Journal of Computer Science and Information Technologies*, 6(1), 168-172. <https://www.academia.edu/download/40003520/ijcsit2015060137.pdf>
- Royce, W. W. (1987). Managing the development of large software systems: Concepts and techniques. En *Proceedings of the 9th International Conference on Software Engineering (ICSE '87)* (pp. 328-338).
- Sabale, R. y Dani, A. (2012). Comparative study of prototype model for software engineering with system development life cycle. *IOSR Journal of Engineering*, 2(7), 21-24. [https://www.iosrjen.org/Papers/vol2_issue7%20\(part-2\)/D0272124.pdf](https://www.iosrjen.org/Papers/vol2_issue7%20(part-2)/D0272124.pdf)

- Sharma, P. y Singh, D. (2015). Comparative study of various **SDLC** models on different parameters. *International Journal of Engineering Research*, 4(4), 188-191. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2628386ec0e41ed06dbb604bf9729e679f394cb2>
- Solinski, A. y Petersen, K. (2016). Prioritizing agile benefits and limitations in relation to practice usage. *Software Quality Journal*, 24, 447-482. <https://doi.org/10.1007/s11219-014-9253-3>
- Sommerville, I. (1996). Software process models. *ACM Computing Surveys (CSUR)*, 28(1), 269-271. <https://dl.acm.org/doi/pdf/10.1145/234313.234420>
- Špundak, M. (2014). Mixed agile/traditional project management methodology: Reality or illusion? *Procedia-Social and Behavioral Sciences*, 119, 939-948. <https://doi.org/10.1016/j.sbspro.2014.03.105>
- Tian, J. (2005). *Software quality engineering: Testing, quality assurance, and quantifiable improvement*. John Wiley & Sons.
- Tilloo, R. (2013). *What is incremental model in software engineering?* <http://www.technotrice.com/incremental-model-in-software-engineering>
- Wallin, C. y Land, R. (2005). *Software development lifecycle models: The basic types. Research methodology for computer science and engineering*. Mälardalen University.

Procesos ágiles

► Objetivos

Los objetivos de este capítulo consisten en introducir los procesos ágiles de desarrollo de software. Al estudiarlo, el lector:

- Conocerá los conceptos fundamentales que subyacen en los procesos ágiles de desarrollo de software.
- Comprenderá los principios y las prácticas aplicados en los procesos ágiles más utilizados actualmente: la programación extrema (eXtreme Programming [XP]) y *scrum*.
- Obtendrá una panorámica global de otros procesos ágiles utilizados en la industria de software.
- Analizará cuáles son los criterios utilizados para seleccionar un proceso ágil adecuado a las características particulares de un proyecto de desarrollo de software.
- Entenderá diferentes modelos de escalamiento de los procesos ágiles.
- Conocerá las diferencias entre los procesos ágiles y tradicionales.

La agilidad en los procesos se refiere a la capacidad del equipo de desarrollo de responder y moverse rápidamente, lo que ayuda a que el proceso de desarrollo de software se vuelva más eficiente (Mahmud y Abdullah, 2015). La agilidad es la habilidad para hacer frente a la creciente complejidad (Tiltmann, 2007) y se

puede ver como un medio para obtener una calidad gradual (Lui y Piccoli, 2006).

El foco de la agilidad se encuentra en los desarrolladores y clientes para producir software que funcione rápidamente y aceptar cambios a través de su ciclo de vida (Abrahamsson *et al.*, 2003; Pohl y Hof, 2015; Singhal, 2012). Por tanto, la agilidad es un enfoque ampliamente utilizado por la industria del software y es apropiada en ambientes rápidamente cambiantes (Bartsch, 2011; Baca y Carlsson, 2011; Ghani y Yasin, 2013; Pohl y Hof, 2015).

El concepto de agilidad se puede estudiar desde el punto de vista de la *dinaxidad*, la cual define cuatro zonas de impacto según la complejidad del producto (estructura) y de la complejidad del proceso (dinámica). Las zonas identificadas son estática, dinámica, turbulenta y caótica (Rieckmann, 1992). Cada zona requiere por parte de las personas involucradas un mayor nivel de tareas a resolver y capacidad para afrontar turbulencias (Rick *et al.*, 2010). La figura 3.1 ilustra las cuatro zonas y el rol de la agilidad.

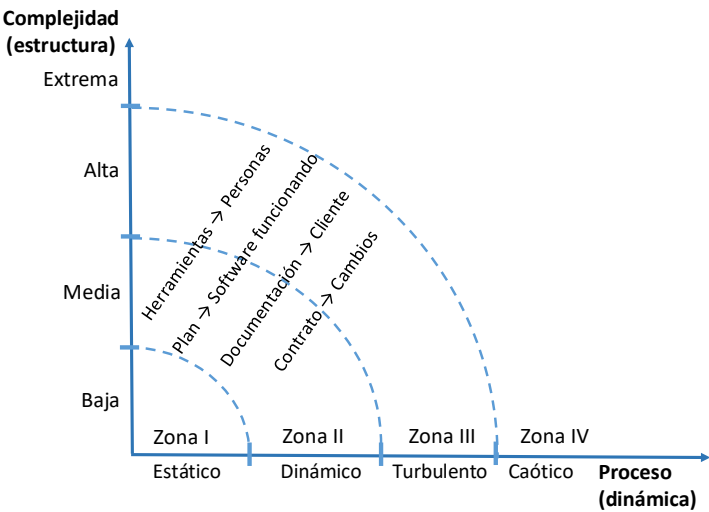


Figura 3.1. Zonas de agilidad según el modelo de dinaxidad

Fuente: elaboración propia con base en Tiltmann (2007).

En la zona I, la complejidad de las actividades de desarrollo es baja y la dinámica del ambiente es estática. En esta zona, las prácticas de desarrollo están bien establecidas, el dominio es estable, existen claras relaciones

de causa y efecto, y las respuestas son simples. La agilidad puede utilizarse para resolver estos tipos de problemas, pero no es la mejor opción, ya que los procesos bien definidos y estructurados pueden ajustarse mucho mejor.

En la zona **II**, la complejidad y la dinámica son más altas, lo que hace surgir múltiples respuestas previsibles, pero no inmediatas, a partir de la consulta de expertos e investigación de varias opciones y con la utilización de métricas. Aunque la agilidad puede aplicarse a algunos problemas de este tipo, no es la mejor solución.

En la zona **III**, la complejidad y la dinámica son imprevisibles debido a cambios técnicos, políticos y económicos. Por ello, se requieren enfoques innovadores y creativos, y niveles más altos de interacción y comunicación. La agilidad es particularmente apropiada para situaciones en esta zona, donde se pueden explorar, inspeccionar y adaptar las soluciones propuestas.

En la zona **IV**, la complejidad y la dinámica son tan extremadamente altas que se vuelven caóticas. Se debe responder en forma inmediata y analizar si la situación se ha estabilizado. Se buscan soluciones que restablezcan el orden en lugar de conseguir respuestas, ya que las causas y los efectos no son claros. La agilidad no es apropiada para este tipo de situaciones en las que se requiere que alguien asuma el mando y realice acciones inmediatas.

Estos cambios de zonas reflejan la variación de situaciones dinámicas a caóticas. Similar a la *dinaxidad*, la habilidad de las empresas desarrolladoras de software para hacer frente a productos complejos y condiciones del ambiente es referida como agilidad (Tiltmann, 2007).

► 3.1. Conceptos sobre procesos ágiles

Según Highsmith y Cockburn (2001), los procesos ágiles hacen hincapié en dos conceptos principales: la implacable honestidad del código que funciona y la eficacia de las personas que trabajan juntas con buena voluntad.

Un proceso de desarrollo de software se puede decir que es un proceso ágil de desarrollo cuando está enfocado en las personas, orientado a las comunicaciones, es flexible (preparado para adaptar cambios esperados o inesperados en cualquier tiempo), rápido (anima el desarrollo rápido e iterativo de productos en entregas pequeñas), ligero (se enfoca en marcos de tiempo cortos, costos bajos y mejora de la calidad), sensible (reacciona apropiadamente a los cambios esperados e inesperados) y orientado al aprendizaje (se enfoca en la mejora durante y después del desarrollo del producto) (Qumer y Henderson-Sellers, 2008).

Los procesos ágiles se caracterizan por ser procesos ligeros que utilizan ciclos iterativos cortos, involucran activamente a los usuarios para establecer, priorizar y verificar requisitos, y confían en el conocimiento tácito del equipo de desarrollo como contraposición a la documentación (Larman, 2004). Un proceso ágil verdadero debe ser iterativo (incluye varios ciclos para completar el producto), incremental (no entrega el producto entero de una sola vez), autoorganizado (los equipos determinan la mejor manera de gestionar su trabajo) y emergente (los procesos, los principios y las estructuras de trabajo son identificados durante el proyecto en lugar de ser predeterminados) (Boehm y Turner, 2005; Cohen *et al.*, 2004).

Los procesos ágiles están diseñados para proporcionar una mayor flexibilidad con respecto a cambios en ambientes internos y externos del sistema. Son una alternativa a los procesos tradicionales pesados (Khatri *et al.*, 2014) y especialmente apropiados cuando los requisitos se mantienen cambiando y la escala del proyecto es pequeña (Meng *et al.*, 2007). En lugar de intentar predecir el futuro, estos procesos animan el diseño incremental y la implementación dirigida por una retroalimentación del usuario a partir de software que funciona. La mayoría de los procesos ágiles son ligeros e intentan minimizar el riesgo, desarrollando y entregando software en iteraciones cortas.

El proceso ágil no solo significa un desarrollo rápido de aplicaciones, también hace énfasis en una adaptación rápida y flexible al cambio del proceso, producto y ambiente. Los procesos ágiles se caracterizan por tener los siguientes atributos (Aoyama, 1998; Bessam *et al.*, 2009; Fowler, 2005), como se observa en la figura 3.2.

- *Procesos ligeros*: Los procesos tradicionales son monolíticos y pesados; en contraste, los procesos ágiles son ligeros y tienen unidades de tamaño pequeñas.
- *Procesos adaptativos*: El cambio es bienvenido en diferentes niveles de abstracción. El cambio permite una mayor creatividad y entrega un valor más rápido al cliente. Se realizan adopciones rápidas al cambio de requisitos y en los ambientes circundantes.
- *Procesos modulares livianos*: Los procesos ágiles son más modulares y livianos (es decir, con menos artefactos, roles y eventos) que los procesos tradicionales.
- *Procesos con ciclos rápidos y entrega frecuente*: La planificación de muchas entregas en cortos periodos obliga a implementar las fun-

cionalidades que tienen mayor prioridad y a entregar valor prontamente al cliente.

- ▶ *Procesos basados en el tiempo*: El diseño del proceso está basado en el tiempo mejor que en el volumen. Los volúmenes grandes de desarrollo deben dividirse en múltiples versiones que pueden ser iterativas o concurrentemente desarrolladas.
- ▶ *Procesos concurrentes y asíncronos*: El progreso en la elaboración del diseño de software no es uniforme a través de las diferentes partes o subsistemas de software.
- ▶ *Macroprocesos incrementales y microprocesos iterativos*: Desde el punto de vista macroscópico, las funcionalidades son divididas en subsistemas o grupos de funciones, y desde el punto de vista de un incremento, un microciclo de vida es iterado hasta que los subsistemas alcancen un nivel determinado de funcionalidad.
- ▶ *Procesos centrados en las personas*: Los desarrolladores no son elementos aislados en una gran organización; por el contrario, son grandes facilitadores en equipos pequeños. Cada equipo se encarga de completar el trabajo para una unidad significativa del sistema desde el análisis hasta las pruebas. Las personas son los conductores primarios para el éxito de los proyectos.

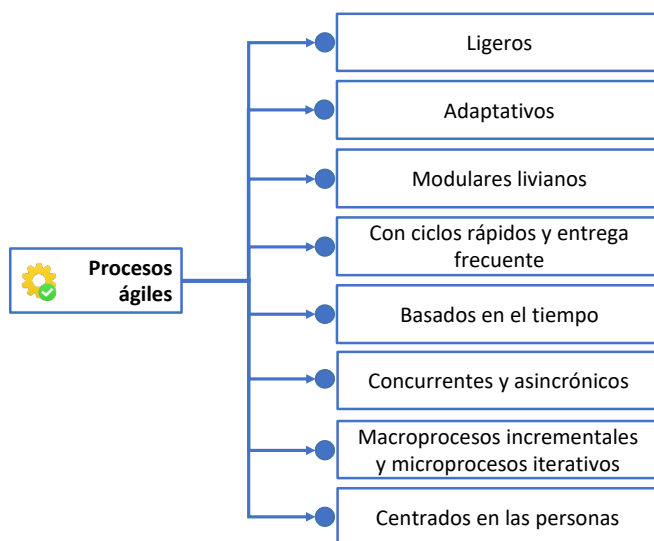


Figura 3.2. Características de los procesos ágiles

Fuente: elaboración propia.

En la figura 3.3, se presenta un esquema general que los procesos ágiles comparten. El proceso contiene tres fases (Ambler, 2009):

- *Inicio*: En esta fase, se define el alcance del proyecto y el modelo de la arquitectura inicial del sistema.
- *Construcción*: Durante esta fase se desarrolla el software en un conjunto de iteraciones. Para cada iteración, el cliente y el equipo de desarrollo determinan el objetivo de la iteración y seleccionan un conjunto de historias de usuarios para lograr la meta. Luego, obtienen los requisitos para las historias, actualizan el diseño para resolver estos y codifican un incremento de software que responde a los requisitos, y es potencialmente entregado al cliente. El equipo realiza una presentación de las historias de usuarios construidas y revisa la eficiencia del equipo al final de la iteración.
- *Transición*: En esta fase, se realizan pruebas de integración para fortalecer el incremento y que esté listo como un producto final en un entorno de producción.

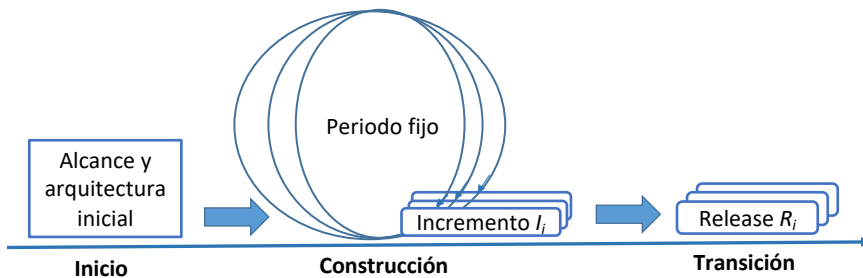


Figura 3.3. Proceso ágil de desarrollo de software

Fuente: elaboración propia con base en Ben Othmane *et al.* (2014).

3.1.1. Historia de los procesos ágiles

El concepto de proceso ágil de desarrollo fue primeramente aplicado por Aoyoma (1998). Incluía una familia de procesos de desarrollo que promovía el cambio evolutivo a través del ciclo de vida del proyecto.

El proceso de software está continuamente evolucionando y la necesidad de una comunicación completa y de entregas de software en iteraciones cortas son la motivación para el auge de los procesos ágiles.

La industria actual se caracteriza por tiempos cortos de lanzamiento de productos al mercado y por la necesidad de desarrollo distribuido de software debido al impacto global del *outsourcing* (Aoyama, 1998). Las empresas desarrolladoras de software han notado la tendencia de pasar de un desarrollo centralizado a una descentralización/distribución del desarrollo de sus productos. Como resultado, los procesos se han desplazado de un modelo secuencial centralizado a un proceso concurrente distribuido, como se observa en la figura 3.4.

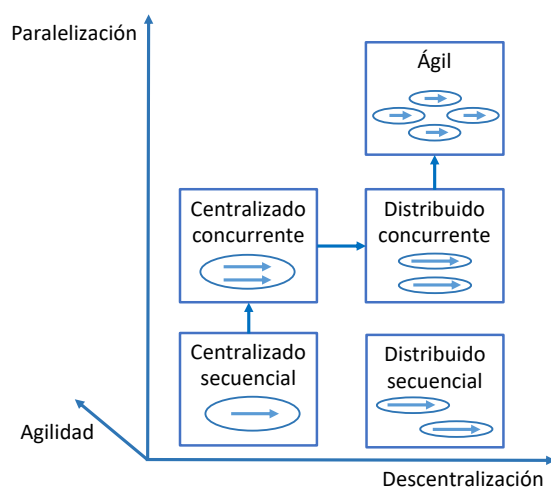


Figura 3.4. Evolución de los procesos ágiles

Fuente: elaboración propia con base en Tahir y Manarvi (2013).

Inicialmente, el desarrollo de software estaba basado en un enfoque secuencial (primero centralizado y luego distribuido) en el que los conceptos a desarrollar se mantenían constantes antes que la implementación comience. Este enfoque secuencial no permite responder a las turbulencias en ambientes complejos en un tiempo apropiado. Para afrontar estos problemas, se desarrolló el enfoque concurrente que paraleliza las etapas del desarrollo interfuncional para mejorar el flujo de tiempo y el trabajo en equipo multidisciplinario. En los procesos concurrentes distribuidos, las funciones del software son concurrentemente desarrolladas por múltiples equipos de desarrollo en múltiples sitios geográficamente distribuidos. Este enfoque ha requerido que se replanteen las actividades desde el desarrollo individual al proyecto entero y dio origen a las propuestas de procesos ágiles.

Como antecedentes de los procesos ágiles, se encuentran los trabajos de Deming sobre el ciclo Plan-Do-Check-Act (**PDCA**), o planificar, hacer, verificar y actuar, *kanban*, *lean manufacturing* y el proceso de desarrollo de software adaptativo. Luego, en la década de 1990, surgieron los primeros procesos y prácticas ágiles que serían incluidos en futuros procesos: *crystal*, refactorización, programación en parejas, *scrum*, Feature-Driven Development (**FDD**), desarrollo de software adaptativo, eXtreme Programming (**XP**) e integración continua.

En 2001, el término *ágil* fue acuñado después de una formulación apropiada de sus características y prácticas por un grupo de desarrolladores (Misra *et al.*, 2009). Luego, un grupo de profesionales asociados con una categoría de desarrollo denominada “ligera” utilizó por primera vez el término *método ágil*. Estos profesionales formaron la Agile Alliance y adoptaron un manifiesto ágil que afirma que el valor de los individuos, la colaboración con el cliente, el software que funciona y la respuesta al cambio son los fundamentos del movimiento ágil. Los desarrolladores antes de la alianza habían utilizado la mayoría de las técnicas ágiles, pero hasta después de la conformación de la alianza esas técnicas fueron agrupadas en un marco de trabajo operacional (Scale Agile Framework, 2020; Highsmith, 2002).

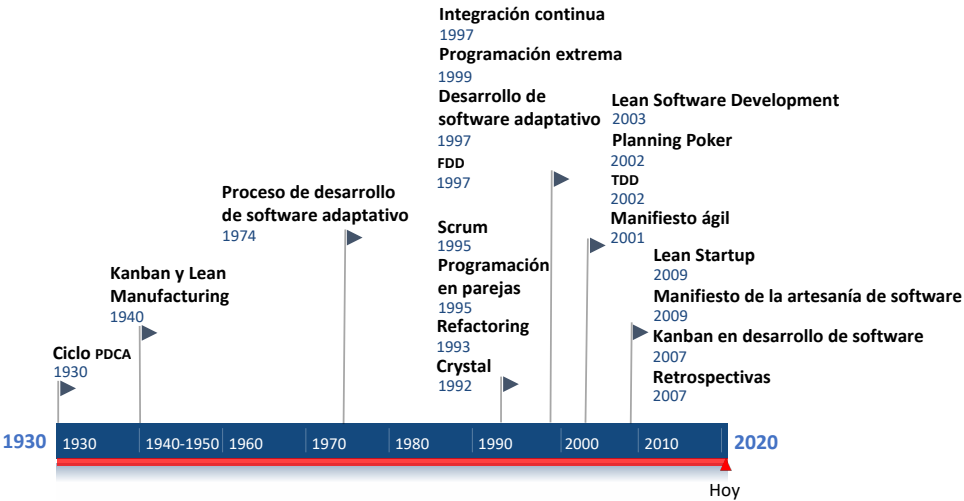


Figura 3.5. Línea de tiempo de los procesos ágiles

Fuente: elaboración propia.

El nuevo siglo también propició el surgimiento de nuevos procesos y prácticas ágiles, como desarrollo dirigido por pruebas, *planning poker*, Lean Software Development (**LSD**), retrospectivas, *kanban* para el desarrollo de software, *software craftsmanship* y *lean startup*.

La figura 3.5 presenta una línea de tiempo con la evolución de las principales prácticas y los procesos ágiles.

3.1.2. Ventajas y beneficios de los procesos ágiles

Los procesos ágiles producen software que potencialmente funcione en intervalos regulares (Ambler, 2009), lo que permite ofrecer a los clientes características de alto valor (funcionalidades del producto de software valoradas por el cliente) en un corto periodo. El enfoque se basa en el uso de patrones, principios y mejores prácticas para desarrollar buen software. Los procesos ágiles poseen varias ventajas (Ben Othmane *et al.*, 2014):

- ▶ Reducen la posibilidad de fracaso del proyecto, ya que permite la detección temprana de brechas entre las expectativas del negocio y el entendimiento de los desarrolladores.
- ▶ Permiten el descubrimiento de las necesidades del cliente en lugar de sus deseos, ya que los clientes pueden ver demos del producto a medida que se desarrolla y adaptar los requisitos en función de sus necesidades.
- ▶ Posibilitan el descubrimiento temprano de barreras técnicas, ya que los desarrolladores experimentan sus ideas y utilizan los resultados para adaptar la arquitectura del sistema y el plan de trabajo.

Las razones fundamentales para implementar métodos ágiles se fundamentan en que la mejor medida de progreso es entregar frecuentemente al cliente software que funciona y en identificar funciones de bajo valor o innecesarias tempranamente, lo que permite no gastar tiempo implementándolas (Boehm y Turner, 2005).

3.1.3. Equipos ágiles

Los equipos ágiles están compuestos de “individuos que gestionan su propia carga de trabajo, asignan su trabajo ellos mismos basados en la necesidad y mejor ajuste, y participan en la toma de decisiones del equipo” (Highsmith, 2004). Los equipos ágiles son equipos democráticos en los que todos los miembros son considerados pares al mismo nivel, sin una je-

rarquía estricta. Los equipos ágiles se caracterizan por ser autogestionados y multifuncionales:

- *Equipos autogestionados*: Son responsables de gestionar, monitorear y mejorar sus propios procesos (Trist, 1981). Los equipos autogestionados pueden organizarse continuamente en varias configuraciones para cumplir los desafíos a medida que estos vayan surgiendo (Highsmith y Cockburn, 2004). Los equipos tienen autoridad y liderazgo compartido para tomar sus propias decisiones (Kirkman y Rosen, 1999; Morgan, 2006). El liderazgo puede asignarse a la persona que tenga el conocimiento, las habilidades y destrezas clave en un momento o situación particular del proyecto (Hewitt y Walz, 2005; Pearce, 2004). Para que este enfoque de trabajo funcione con éxito, los miembros del equipo deben tener múltiples destrezas para ser capaces de ejecutar actividades de otros miembros y sustituirlos cuando las circunstancias lo requieran (Morgan, 2006).
- *Equipos multifuncionales*: Tienen la experiencia necesaria para cada fase involucrada en el desarrollo de software. Para ello, deben contar con la habilidad y autoridad necesarias para entregar conjuntos de características útiles a clientes independientes de otros equipos. Los equipos deben ser formados a partir de líneas de características o servicios, es decir, se subdivide el equipo a lo largo de subsistemas visibles por los usuarios, cada uno cubriendo un subconjunto de funcionalidades (Beck, 2000).

Los líderes de los equipos ágiles son responsables de establecer la dirección, alinear las personas, obtener los recursos y motivar al equipo (Anderson *et al.*, 2003). Los proyectos ágiles tienen títulos específicos para los roles de los líderes como *scrum master* (Schwaber y Beedle, 2001) y *XP Coach* (Fraser *et al.*, 2003), en lugar de los administradores tradicionales.

Generalmente, los equipos ágiles realizan reuniones diarias y comparten un mismo lugar de trabajo, de tal manera que pueden ver a los demás miembros del equipo y saber qué tareas están realizando. Los equipos muestran evidencia inmediata de su progreso y puede ajustar su trabajo apropiadamente conociendo los responsables de las tareas. Los equipos realizan una planificación *bottom-up* del trabajo, el cual crea un entendimiento común de este. Esta forma de trabajo de los equipos ágiles hace las tareas más predecibles.

3.1.4. Manifiesto ágil y principios ágiles

En la reunión convocada por Kent Beck en 2001, se trataron técnicas y procesos para desarrollar software y se acuñó el término *métodos ágiles* para definir los métodos que estaban surgiendo como alternativa a las metodologías formales o estandarizadas. Los asistentes a esta reunión denominaron manifiesto ágil al conjunto resumido de principios en que se basaban los nuevos métodos propuestos. El manifiesto ágil es un texto compacto y ligero que está conformado por cuatro postulados que se ilustran en la figura 3.6.

Manifiesto por el desarrollo ágil de software

Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

- Individuos y sus interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre seguir un plan.

Esto es, aunque se valoran los segundos elementos, los firmantes del manifiesto ágil valoran más los primeros elementos.

Figura 3.6. Manifiesto ágil

Fuente: elaboración propia con base en Agile Manifesto (2001).

En la tabla 3.1, se presenta una breve explicación de cada uno de los puntos que constituyen el manifiesto ágil.

Tabla 3.1. Descripción de los puntos del manifiesto ágil

Puntos del manifiesto	Descripción
Individuos y sus interacciones sobre procesos y herramientas	Las personas son los ingredientes más importantes para el éxito. Un buen proceso no salvará un proyecto de su fracaso si el equipo no tiene buenos individuos. Incluso un grupo de buenos individuos puede fallar si no trabajan como un equipo.
Software funcionando sobre documentación extensiva	El objetivo fundamental de los proyectos es entregar software que funcione en iteraciones cortas y con retroalimentación del cliente. No se produce ningún documento a menos que su necesidad sea inmediata y significativa.

Puntos del manifiesto	Descripción
Colaboración con el cliente sobre negociación contractual	Los proyectos exitosos involucran una retroalimentación frecuente del cliente. En lugar de depender de un contrato, el cliente trabaja con el equipo de desarrollo proporcionando comentarios frecuentes sobre sus esfuerzos. Se enfatiza en la colaboración en lugar de especificar detalles del alcance y calendario a un costo fijo.
Respuesta ante el cambio sobre seguir un plan	Es difícil escribir requisitos confiables. Es probable que los clientes modifiquen los requisitos una vez vean el sistema funcionando. Una mejor estrategia es hacer planes detallados de las semanas próximas para aquellas tareas que son inmediatas. Sin embargo, el plan general sigue siendo flexible y sus partes de menor resolución se pueden cambiar con relativa facilidad.

Fuente: elaboración propia.

A su vez, el manifiesto ágil proporciona doce principios para incrementar el entendimiento del proceso (Agile Manifesto, 2001).

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.

10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, los requisitos y diseños emergen de equipos autogestionados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Estos principios abarcan tanto cuestiones técnicas como de gestión. A continuación, se describirá en detalle cada uno de los principios.

1. *Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor* (Agile Manifesto, 2001): El cliente es el centro del proceso y, por tanto, está involucrado en todo el proceso. En los procesos tradicionales, el cliente interviene al comienzo del proceso durante la definición de requisitos y luego al final durante las pruebas de aceptación del usuario. Para mantener al cliente satisfecho, los procesos ágiles mantienen el contacto con el cliente en etapas intermedias. Los clientes son invitados permanentes a reuniones del proyecto, pueden interactuar libremente y realizar pruebas de las versiones incrementales entregadas tempranamente. Algunos procesos ágiles permitirán que el cliente o uno de sus representantes formen parte del equipo de desarrollo. Las entregas tempranas y frecuentes posibilitarán al cliente evaluar el producto especificado y colocar el sistema en producción si la funcionalidad alcanzada es suficiente o reportar cambios que ellos desean que se implementen.
2. *Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo*. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente (Agile Manifesto, 2001): Los enfoques tradicionales asumen que, una vez obtenida la especificación de requisitos, no cambiará en tiempos futuros. Sin embargo, la realidad es otra: la retroalimentación del cliente hace que los requisitos cambien frecuentemente. En los procesos ágiles, el equipo tiene una actitud favorable a recibir, gestionar e implementar cambios en el software. El reto es mantener una estructura flexible del software para que los cambios puedan ser implementados y el impacto en el sistema sea mínimo. Los procesos ágiles entienden que la retroalimentación constante del cliente implica cambios y que facilitar el cambio es más efectivo que intentar prevenirlo.

3. *Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo más corto posible* (Agile Manifesto, 2001): El objetivo es obtener una retroalimentación rápida del software entregado al usuario, aun si este incluye solo un subconjunto de la funcionalidad esperada. La duración de una iteración para entregar determinada funcionalidad se fija por adelantado. Si al final del tiempo asignado no se completa la funcionalidad esperada, esta se pasa a la siguiente iteración o se descarta por completo, pero la fecha límite no cambia. Lo importante es entregar software que funcione, que pueda ser evaluado por el cliente en periodos cortos. Los procesos ágiles animan a entregar software, no documentos, los cuales no se consideran entregas verdaderas.
4. *Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto* (Agile Manifesto, 2001): Al principio de los proyectos, no se puede esperar tener un conjunto detallado de requisitos por parte del cliente. En su lugar, se tiene una visión panorámica global de los requisitos, los cuales deben ser ajustados y modificados con el tiempo. Para que estos ajustes y cambios puedan realizarse con éxito, se requiere que los responsables del negocio o representantes del cliente trabajen conjuntamente y a diario con el equipo de desarrollo. Los clientes deben estar involucrados estrechamente en todo el proceso de desarrollo, proporcionando y priorizando requisitos y evaluando los incrementos obtenidos con la ejecución de las iteraciones.
5. *Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo* (Agile Manifesto, 2001): Las personas son las que garantizan el éxito o fracaso de los proyectos, a pesar de contar con las mejores herramientas, tecnologías y procesos. Las personas deben tener mucha confianza en el trabajo que están realizando y, para ello, deben participar en la toma de decisiones sobre aspectos que conocen y dominan. Sus habilidades deben ser reconocidas por la administración. A su vez, esta debe confiar en los equipos de desarrollo y facilitarles un entorno de trabajo adecuado para que puedan desarrollar sus propios esquemas de trabajo.
6. *El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara*

(Agile Manifesto, 2001): Las personas generalmente están trabajando en oficinas con puertas cerradas o en cubículos. En estos ambientes, no se facilitan las conversaciones ni transmiten un sentimiento de un grupo de personas que realizan un trabajo que están emocionados de realizar. En los procesos ágiles, se recomienda utilizar espacios abiertos en lugar de cubículos, donde si alguien necesita ayuda puede dirigirse directamente al interesado y consultarlo directamente. Los procesos ágiles hacen énfasis en espacios de trabajo colaborativos y conversacionales. Por otro lado, el problema no es la documentación sino el entendimiento del producto y, para ello, los procesos ágiles promueven técnicas de comunicación más directas.

7. *El software funcionando es la medida principal de progreso* (Agile Manifesto, 2001): Existen muchos métodos para evaluar el estado actual de un proyecto de desarrollo de software, entre ellos se encuentran la identificación de la fase en la que se encuentra el proyecto, la cantidad de artefactos o documentos producidos o la cantidad de código desarrollado. Sin embargo, para los procesos ágiles la principal medida de progreso es la cantidad de requisitos funcionales que han sido implementados y probados por el equipo de desarrollo, y que además hayan sido aprobados por el cliente. El porcentaje de avance es el porcentaje de funcionalidad que funciona y que ha sido entregada y aprobada.
8. *Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida* (Agile Manifesto, 2001): Los procesos ágiles hacen énfasis en el rol central de los desarrolladores y la necesidad de darles condiciones de trabajo que les permita desarrollar su máximo potencial. Una práctica común en los procesos tradicionales es aceptar compromisos no realistas y luego se obliga a que el equipo de desarrollo los cumpla bajo presión, con extenuantes días laborales y sacrificando fines de semana. Los procesos ágiles enfatizan en la necesidad de proveer a las personas con buenas condiciones de trabajo que les posibilite mantener un ritmo constante de trabajo. El desarrollo sostenible se refiere a que el equipo de desarrollo encuentra un ritmo de 40 horas de trabajo semanales que puede sostener sin sobresaltos, llegado el caso de que el ritmo será rápido pero sostenible manteniendo los estándares de calidad.

9. *La atención continua a la excelencia técnica y al buen diseño mejora la agilidad* (Agile Manifesto, 2001): Para un desarrollo ágil, se debe mantener el software limpio y robusto tanto como sea posible. Esta práctica se realiza una vez se termine una funcionalidad específica, no se deja para más adelante, cuando el equipo tenga más tiempo. El diseño es una actividad continua que es ejecutada a lo largo del proyecto, en cada iteración se realizarán actividades de diseño de excelente calidad. La calidad del diseño se reflejará en mejores rendimientos durante la realización de pruebas y mantenimiento. El diseño del proyecto se mejora continuamente durante todo el proyecto.
10. *La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial* (Agile Manifesto, 2001): Muchos productos de software desarrollan una gran cantidad de funcionalidades que no es fundamental o que necesitan realmente pocos usuarios. Las características adicionales aumentan la complejidad del código y, por tanto, aumentan los costos de forma no lineal. Además, para facilitar las actividades de diseño, este debe ser simple, pero debe abarcar los principales aspectos del sistema. Los desarrolladores ágiles realizan un trabajo diario simple, pero de excelente calidad, confiando que será fácil la realización de cambios si aparecen futuros problemas. La simplicidad se aplicará tanto al producto como al proceso, de tal manera que se reduzca la complejidad del sistema.
11. *Las mejores arquitecturas, requisitos y diseños emergen de equipos autogestionados* (Agile Manifesto, 2001): Las responsabilidades del equipo no son gestionadas por miembros individuales externos; en su lugar, son gestionadas por el equipo como un todo. Los mismos integrantes del equipo deciden cómo cumplir sus responsabilidades. De este modo, los procesos ágiles permiten el surgimiento de propiedades emergentes del equipo, como la innovación y la creatividad, las cuales son fomentadas por los equipos autogestionados con altas interacciones entre los miembros y pocas reglas. No existen estructuras jerárquicas predeterminadas, el mismo equipo se organiza de acuerdo con las responsabilidades y compromisos adquiridos.
12. *A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en*

consecuencia (Agile Manifesto, 2001): Los procesos ágiles incentivan la reflexión interna del equipo de desarrollo para evaluar sus esquemas de trabajo. La reflexión está orientada a la mejora continua, identificando al final de cada iteración mejores prácticas, lecciones aprendidas y oportunidades de mejora para las siguientes iteraciones o, incluso, proyectos futuros. De esta forma, los equipos ágiles mejoran continuamente, realizando una introspección de sus prácticas y entorno, para responder de manera oportuna a cambios en el ambiente, eliminar prácticas obsoletas y difundir las mejores prácticas con base en la experiencia.

3.1.5. Grado de utilización de procesos ágiles

Actualmente, se están aplicando muchos procesos ágiles de desarrollo en la industria del software. En la figura 3.7, se presentan porcentajes de utilización de métodos y prácticas ágiles en diferentes proyectos tomados del 17.º Reporte Anual del Estado Ágil (Digital.ai, 2023). Según esta encuesta, la mayoría de los usuarios ágiles a nivel de equipo (63%) siguen la metodología *scrum*. Esto no es sorprendente, desde su primera encuesta en 2006, *scrum* ha sido la metodología ágil más popular.

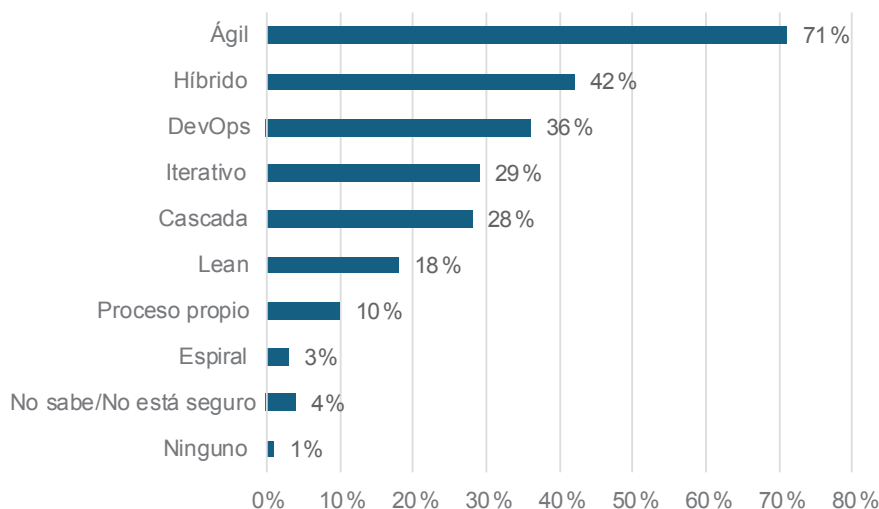


Figura 3.7. Métodos y prácticas ágiles

Fuente: elaboración propia con base en Digital.ai (2023).

En este capítulo, se describirán en forma extensa estos dos procesos ágiles, ya que son ampliamente utilizados en la industria y abundantemente referenciados en la literatura. Se iniciará con la programación extrema (eXtreme Programming [XP]) y luego con *scrum*. Más adelante, se mostrarán otros procesos ágiles, pero en una forma breve y concisa.

► 3.2. eXtreme Programming (XP)

XP es un proceso ágil y ligero de desarrollo de software creado por Kent Beck. Los propulsores de XP creen que los cambios en los requisitos y el medio ambiente son una parte inevitable del desarrollo de software. Por ello, los equipos de desarrollo deben aceptar el cambio en lugar de tratar de predecirlo.

3.2.1. Valores de XP

XP mejora los proyectos de software en atención a cinco valores esenciales: respeto, comunicación, simplicidad, retroalimentación y coraje (Meng *et al.*, 2007).

- *Respeto*: El respeto se presenta en dos vías. Primero, desde el punto de vista del equipo de desarrollo, para que cada miembro se considere una parte valiosa de este. En segundo lugar, desde la gestión, para que el equipo se considere responsable de lo que se comprometa.
- *Comunicación*: Para desarrollar un software con calidad, se requiere una comunicación clara y frecuente entre el cliente y el equipo de desarrollo. En lugar de una documentación exhaustiva, se recomienda una interacción más asidua y cercana con el cliente.
- *Simplicidad*: La simplicidad se refiere a satisfacer la necesidad requerida y nada más, en lugar de funcionalidades complejas que nunca se necesitarán. Tomando como punto de partida una línea base simple de código, la cual se pueda actualizar de forma fácil y rápida.
- *Retroalimentación*: La integración continua y las pruebas garantizan una retroalimentación constante para que el código no se desestabilice cuando se le realicen modificaciones. La retroalimentación obtenida puede ser agregada al código o incluirse en la siguiente iteración.

- *Coraje*: El coraje es un deber que tienen los equipos de desarrollo al autoorganizarse y realizar sus propias estimaciones de esfuerzo. Deben tener coraje para detectar los defectos tempranamente y responder de manera efectiva al cambio y las retroalimentaciones obtenidas desde el cliente y el equipo de desarrollo.

3.2.2. Prácticas de xp

xp está diseñada para utilizarse en equipos pequeños y proyectos de tamaño mediano-pequeño que necesitan desarrollar software rápidamente en ambientes con cambios frecuentes de requisitos. **xp** se implementa por medio de doce prácticas (Abrantes y Travassos, 2011; Millett *et al.*, 2011; Newkirk, 2002). Las prácticas están relacionadas con los cuatro valores del manifiesto ágil y se muestran en la figura 3.8.

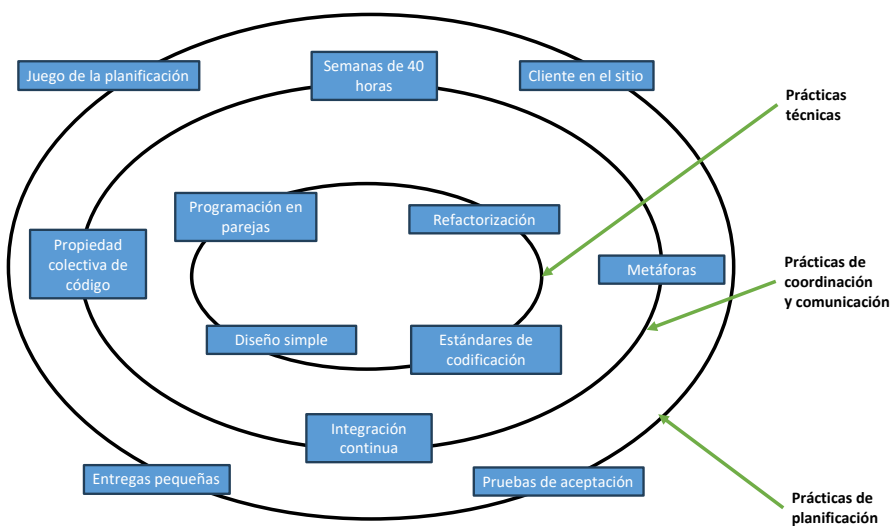


Figura 3.8. Prácticas de xp

Fuente: elaboración propia con base en Lindstrom y Jeffries (2004).

- *Juego de la planificación*: Con esta práctica, se determina el alcance de la siguiente iteración. Los desarrolladores trabajan junto con el cliente para producir las características del sistema como historias de usuario que serán desarrolladas en las diferentes entregas del sistema. Estas son priorizadas por los clientes y, luego, los desarro-

lladores proporcionarán estimaciones técnicas para su implementación. Como resultado, se obtiene una planificación a corto plazo con tareas de desarrollo junto con el compromiso de cada persona involucrada.

- *Cliente en el sitio*: El cliente forma parte del equipo y está disponible todo el tiempo para responder preguntas, resolver inquietudes, establecer prioridades, ejecutar pruebas de aceptación y asegurar que el desarrollo está progresando como se espera. Por tanto, el cliente debe tener un espacio físico propio asignado en el equipo de desarrollo.
- *Metáforas*: El uso de metáforas permite adoptar una historia compartida y simple entre los desarrolladores y el cliente. La metáfora ayudará a obtener un entendimiento común sobre cómo funciona el sistema globalmente, es decir, una arquitectura de alto nivel. Las metáforas sirven para inspirar el diseño y el desarrollo, proporcionan un vocabulario común de términos en el equipo de desarrollo y los clientes, que permite que los sistemas complejos puedan ser explicados fácilmente.
- *Entregas pequeñas*: Consiste en llevar el sistema a producción rápidamente para obtener retroalimentación temprana por parte del cliente. Cada entrega produce un sistema de software útil y funcional que genera valor de negocio al cliente. Al final de cada entrega, el cliente revisa el producto, identifica defectos y realiza ajustes de requisitos futuros. Esta estrategia evita integrar grandes subsistemas en etapas finales del proyecto.
- *Diseño simple*: Uno de los valores clave es la simplicidad, eliminando código extrainnecesario que hace al sistema más complejo, enfocándose en las necesidades actuales en lugar de cambios futuros. El diseño de alto nivel se realizará al inicio y durante cada iteración.
- *Programación en parejas*: Es un requisito que el código debe ser producido por dos personas al mismo tiempo y en la misma máquina. Los desarrolladores no pueden programar solos durante los proyectos, alternan los roles de conductor y navegador. El conductor controla el mouse y el teclado, implementa el código y lo explica al navegador, y desarrolla casos de prueba. El navegador revisa defectos de sintaxis y semántica, decidiendo si el código generado es la mejor forma de implementar la funcionalidad esperada. De esta

forma, el código se revisa en tiempo real, produciendo niveles de calidad que las revisiones e inspecciones tradicionales no logran.

- ▶ *Propiedad colectiva de código:* Cualquier miembro del equipo de desarrollo puede cambiar cualquier parte del código del sistema en cualquier momento. Para ello, debe existir un repositorio de código de fácil acceso y sin necesidad de pedir permiso a otros roles. No hay partes del sistema que sean propiedad o responsabilidad de un desarrollador o grupo de desarrolladores. Cualquier parte del código puede ser consultado o modificado por cualquier desarrollador. De esta manera, se contrarresta que el sistema dependa de expertos individuales.
- ▶ *Estándares de codificación:* Debido a que diferentes desarrolladores codifican diferentes partes del sistema, los estándares de código son necesarios. Los estándares mejoran la comunicación, ya que el código se desarrolla de una manera consistente, es más fácil de entender y se incrementa su legibilidad.
- ▶ *Semanas de 40 horas:* Se debe trabajar el tiempo necesario para que el equipo de desarrollo sea productivo a un ritmo sostenible. Se recomienda trabajar 40 horas a la semana, 8 horas diarias. Generalmente, si se presentan horas extras, la calidad de los productos es baja. Los requisitos seleccionados para las iteraciones deben ser planificados de tal manera que no haya necesidad de tiempos extra para su terminación. De esta forma, se obtiene un ritmo sostenible que ayuda a planificar temas, como vacaciones, ausencias no planificadas y situaciones de emergencia que necesitan resolverse inmediatamente.
- ▶ *Integración continua:* Los miembros del equipo de desarrollo deben frecuentemente integrar y construir el producto de software muchas veces al día en un ambiente de desarrollo compartido, cada vez que nuevos cambios o tareas se finalicen. De esta manera, se detectan problemas y defectos en el sistema tempranamente y se corrigen lo más rápidamente posible, obteniendo frecuentemente código que funciona. Generalmente, cada miembro del equipo integra una vez al día. Un servidor de integración continua extrae el código base del repositorio, lo construye y ejecuta todas las pruebas automatizadas para asegurar que su construcción no contenga errores. Todas las pruebas diseñadas deben pasar la integración, si no el nuevo código debe ser descartado.

- *Refactorización*: Consiste en reestructurar el código existente o mejorar constantemente su entendimiento y facilidad de mantenimiento. Su diseño es mejorado, alterando su estructura interna sin realizar cambios en su comportamiento externo. La refactorización es parte de las actividades diarias de los desarrolladores. Involucra simplificación de sentencias complejas, abstracción de soluciones comunes en código reutilizable y eliminación de código duplicado. El código refactorizado debe pasar los casos de pruebas unitarias.
- *Pruebas de aceptación*: Los desarrolladores deben escribir pruebas unitarias y los clientes deben escribir pruebas de aceptación. Las pruebas se vuelven parte del diseño natural del sistema. Pasar las pruebas se considera un requisito para entregar el producto y son indicadores clave de terminación del proyecto. Se promueve la realización de pruebas automatizadas tanto como sea posible.

3.2.3. Proceso de xp

El proceso de **xp** está conformado por seis fases: exploración, planificación, iteraciones, producción, mantenimiento y cierre (Ambler, 2002). Estas fases se ilustran en la figura 3.9.

- *Fase de exploración*: Los usuarios escriben las historias de usuario a incluir en la primera versión. Cada historia de usuario describe una funcionalidad a incorporar al programa. Además, el equipo de proyecto se familiariza con las tecnologías y herramientas a utilizar y construye prototipos para probar posibles arquitecturas. La duración de esta fase puede variar de unas pocas semanas hasta pocos meses, según la familiaridad con las tecnologías.
- *Fase de planificación*: Se priorizan las historias de usuario y se define el contenido de la primera entrega. Se le realiza una estimación a cada historia de usuario. La primera entrega no debe tardar más de dos meses en desarrollarse. Esta fase dura pocos días.
- *Fase de iteraciones*: Se realizan diferentes actividades para obtener la primera versión del producto. La planificación se divide en diferentes iteraciones de una a cuatro semanas. Los usuarios deciden qué historias desarrollar en cada iteración en atención a que en la primera iteración se desarrolla la arquitectura del sistema. Las pruebas creadas por el cliente son ejecutadas al final de cada iteración, de tal manera que al finalizar esta fase se obtiene una versión lista para producción.

- *Fase de producción:* Se realizan pruebas de rendimiento y funcionamiento necesarias para entregar el producto al cliente. Las iteraciones de esta fase no duran más de tres semanas.
- *Fase de mantenimiento:* Cuando se obtenga la primera versión del producto, el proyecto se mantiene en entorno de producción siempre que haya iteraciones en fase de producción.
- *Fase de cierre del proyecto:* No se tienen más historias de usuario a implementar, ya que se satisfacen todas las necesidades de los clientes y otros requisitos no funcionales. Se realiza la documentación del proyecto.

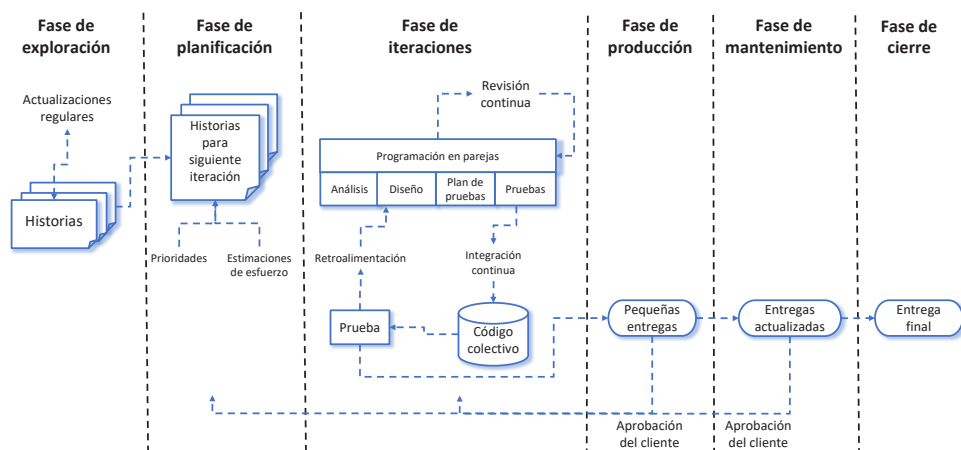


Figura 3.9. Proceso de xp

Fuente: elaboración propia con base en Ambler (2002).

Los diferentes roles que intervienen en un proyecto **XP**, según las tareas que desarrollan y los propósitos a lograr, son:

- *Cliente:* Encargado de escribir las historias y las pruebas de aceptación. Decide cuando un requisito se satisface y establece la prioridad de las funcionalidades.
- *Programador:* Escribe las pruebas funcionales y el código de la aplicación.
- *Tester:* Ayuda a escribir las pruebas funcionales, las ejecuta y transmite los resultados de las pruebas.

- *Rastreador*: Realiza seguimiento al proyecto informando de posibles desviaciones para mejorar las futuras estimaciones. Además, realiza seguimiento de las iteraciones y valora si los objetivos se pueden alcanzar con los recursos disponibles.
- *Coach*: Responsable de aplicar correctamente el proceso. Tiene conocimientos en proyectos **XP** y guía al equipo en la adopción efectiva de **XP**.
- *Consultor*: Es un miembro externo que tiene los conocimientos técnicos necesarios para guiar al equipo en la solución de problemas específicos.
- *Manager*: Toma decisiones críticas y trabaja con el equipo para afrontar situaciones difíciles.

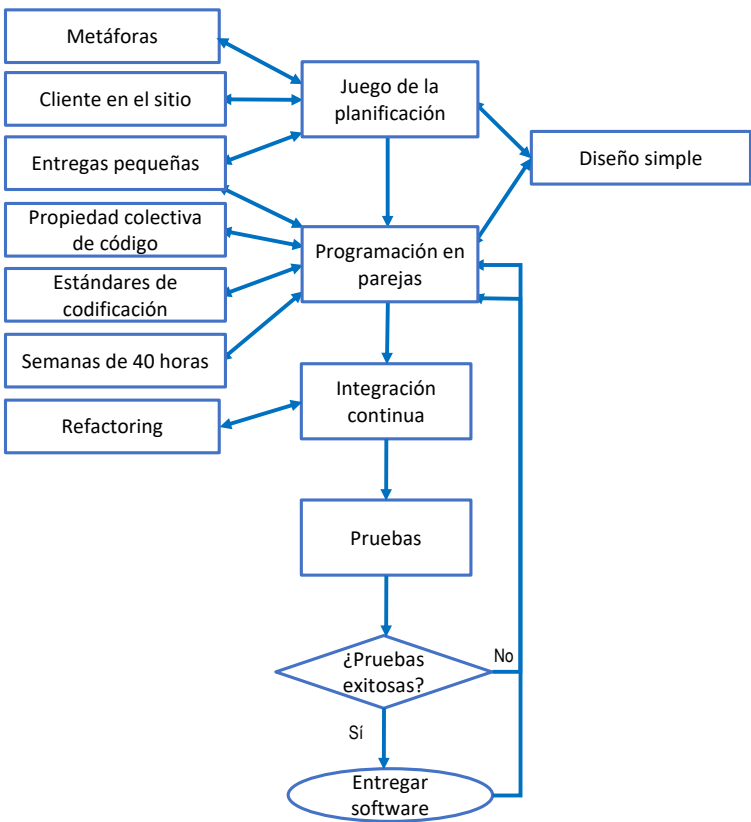


Figura 3.10. Proceso de **XP**

Fuente: elaboración propia con base en Adelyar y Norta (2016).

En la figura 3.10, se presenta la relación de las diferentes prácticas de **XP**. Se comienza con la recolección de requisitos en la que se aplica la práctica de planificación. Otras prácticas involucradas indirectamente son cliente en el sitio, metáforas, diseño simple y entregas pequeñas. Un equipo de desarrollo implementa las historias de usuario en las que la práctica principal es la programación en parejas. Otras prácticas involucradas son estándares de codificación, diseño simple, entregas pequeñas, propiedad colectiva de código y semanas de 40 horas. Los desarrolladores determinan las tareas necesarias para implementar las historias de usuario para la iteración actual. Luego, las características funcionales son integradas al software en que la práctica principal es la integración continua. Con las prácticas de refactorización y diseño simple, los desarrolladores rediseñan y refactorizan partes relevantes del software. El proceso se repite hasta que todas las entregas pequeñas han sido probadas exitosamente y luego se realiza la entrega final del software.

3.2.4. Ventajas y desventajas de **XP**

Entre las ventajas y fortalezas de **XP**, se encuentran (Beck, 2000; Beck y Fowler, 2001):

- ▶ Considera mejor las necesidades del cliente, mejora la calidad del software y reduce las demandas de recursos.
- ▶ El objetivo del proyecto puede ajustarse al ambiente y las necesidades del mercado. Cuestiones relevantes del proyecto pueden ser ignoradas hasta que se obtenga la información necesaria del ambiente. Por tanto, el riesgo se reduce y la productividad aumenta.
- ▶ El proyecto puede finalizar con cualquier entrega intermedia, ya que es completamente funcional y cubre las características funcionales más importantes y priorizadas.

Entre las principales debilidades de **XP** están (Meyer, 2014):

- ▶ La mayoría de las dificultades están de lado del cliente, quien puede influir en el proyecto, pero no pueden controlarlo.
- ▶ Para proyectos pequeños, **XP** permite una revisión completa del diseño total del proyecto. Sin embargo, en proyectos grandes no se permite debido a restricciones de tiempo. Por tanto, el software tiende a optimizarse en mínimos locales y no necesariamente a óptimos globales.

- Los equipos tardan más tiempo en integrarse en los proyectos debido a que no hay documentación que apoye esta labor.
- No es apropiada para todos los proyectos, depende de las necesidades del cliente. Puede ser aplicada si los proyectos no consisten en constructos complejos que están cercanos a la interfaz de usuario del sistema.

► 3.3. *Scrum*

Jeff Sutherland y su equipo en Easel Corporation crearon el proceso *scrum* en 1993, combinando conceptos del desarrollo orientado a objetos, control de proceso empírico, desarrollo iterativo e incremental, investigación en productividad, procesos de software y sistemas adaptativos complejos.

El término *scrum* no es un acrónimo, proveniente del *rugby*, es la forma en que se reinicia el juego después de una infracción accidental o cuando el balón ha salido de juego, y se observan dos conjuntos de masas corporales apretujadas para ganar la posesión del balón.

Scrum es un marco de trabajo basado en pocos eventos, artefactos, roles y algunas reglas que los entrelazan (Sutherland y Schwaber, 2020). Es un proceso de desarrollo ágil, iterativo e incremental que asume que el cambio es inherente en el ciclo de vida de los proyectos. Hace énfasis en la planificación y el seguimiento de las iteraciones, organiza los requisitos con base en el valor técnico y de negocios, y selecciona las tareas más prioritarias (Meng *et al.*, 2007). *Scrum* es apropiado en proyectos con cambios rápidos o con requisitos altamente emergentes, tales como proyectos web o desarrollo de productos para nuevos mercados (Cohn y Ford, 2003).

En la figura 3.11, se presentan los principales roles, artefactos y actividades de *scrum*, y cómo se relacionan entre ellos. A continuación, se describen estos elementos y otros conceptos adicionales de *scrum*. Esta información ha sido tomada principalmente de Maximini (2015), Millett *et al.* (2011), Rossberg (2014) y Sutherland y Schwaber (2020).

3.3.1. Roles

Scrum incluye dos tipos de roles genéricos: los interesados y los comprometidos. Los roles interesados en *scrum* son personas con disímiles cargos, como administradores de negocios, directores de unidades organizacionales, clien-

tes, vendedores, patrocinadores y *stakeholders* en general. Ellos no están activamente involucrados en el desarrollo del producto y participan en eventos concretos como revisiones del *sprint* en los que brindan retroalimentación valiosa sobre el producto desarrollado. Los comprometidos son únicamente tres roles: *scrum master*, propietario del producto (*product owner*) y equipo de desarrollo. Estos tres roles conforman un equipo *scrum*. A continuación, se describen brevemente los roles comprometidos (figura 3.11).

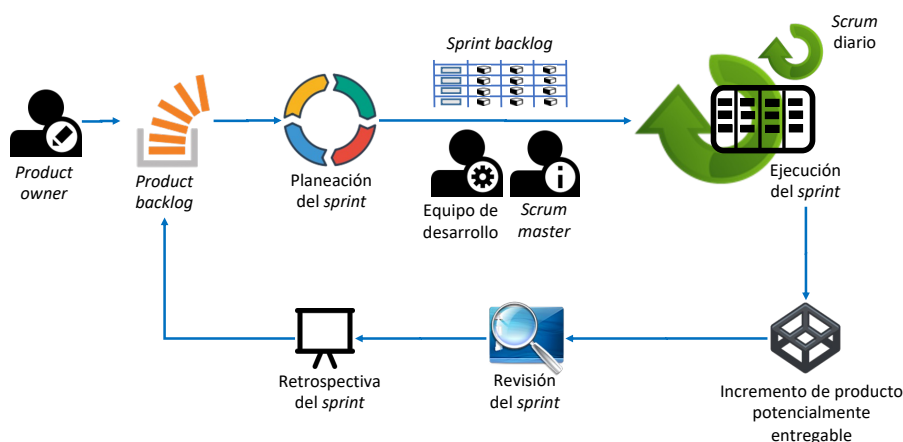


Figura 3.11. Marco de trabajo *scrum*

Fuente: elaboración propia.

3.3.1.1. *Product owner* (propietario del producto)

Es el propietario del producto de software a desarrollar y, por tanto, el responsable de tomar todas las decisiones que maximizarán el valor del producto y afectarán su futuro. Entre sus responsabilidades está gestionar el *product backlog*; para ello, debe expresar claramente los elementos que lo conforman. Luego, debe ordenar y priorizar esos elementos, de tal manera que los elementos que se construyan primero produzcan el mayor valor de retorno de la inversión. El *product owner* realiza esta priorización consultando a los principales *stakeholders* del producto en reuniones previamente realizadas con ellos.

El *product owner* es la única persona que conoce los requisitos desde la mejor perspectiva del cliente. Por tanto, construye la visión del producto a desarrollar y toma la decisión final de lanzar un producto al mercado una vez que evalúe su valor de negocio.

3.3.1.2. *Scrum master*

El *scrum master* es el propietario del proceso *scrum*. Por tanto, debe tener un conocimiento experto del proceso *scrum*. Entre sus responsabilidades, está asegurar que el proceso *scrum* es entendido y que su teoría, prácticas y reglas sean adoptadas por todos los miembros del equipo *scrum*. Es un líder que está al servicio del equipo *scrum* y lo protege de posibles interrupciones u obstáculos que puedan afectar al proyecto.

Otras responsabilidades a su cargo son incrementar la productividad del equipo *scrum* y compartir el objetivo común de los *sprints*; remover o eliminar impedimentos que surjan en el equipo para cumplir sus objetivos; realizar actividades de entrenamiento a los participantes de proyectos *scrum*; facilitar la ejecución de los eventos *scrum*, y encargarse de las comunicaciones con otros equipos *scrum* o con la administración.

El *scrum master* debe tener buenas habilidades de comunicación y ser un motivador del equipo ayudándolo a resolver posibles conflictos que se pueden presentar.

3.3.1.3. Equipo de desarrollo

El equipo de desarrollo es el encargado de transformar los requisitos del producto en un producto terminado por medio de un trabajo iterativo e incremental realizado en periodos cortos, los *sprints*. Solo los miembros del equipo de desarrollo participan de la creación del producto de software. Los equipos de desarrollo tienen las siguientes características:

- *Son autogestionados*: El equipo de desarrollo no tiene ningún líder. Ellos mismos determinan cómo convertir los elementos del *product backlog* en incrementos de funcionalidad potencialmente entregables, y deciden qué se comprometen a obtener con cada *sprint*. Son autónomos al seleccionar sus propias herramientas y métodos que garanticen el éxito del desarrollo. El equipo de desarrollo resuelve los impedimentos por sí mismo y posee la suficiente autoconfianza para seleccionar la cantidad necesaria de trabajo en cada *sprint*.

- *Son multifuncionales*: Todos los miembros del equipo de desarrollo cuentan con todas las habilidades necesarias para construir un incremento de producto. Con ello, optimizan la flexibilidad y productividad. Aunque el equipo de desarrollo incluye una combinación de programadores, diseñadores, *testers* y otros roles, no tienen definidos roles especializados en el equipo, todos son desarrolladores, sin importar las actividades que realicen en un momento dado. Aunque pueden tener habilidades especializadas, la responsabilidad recae en el equipo de desarrollo como un todo.

El tamaño óptimo del equipo de desarrollo es lo suficientemente pequeño para permanecer ágil y lo suficientemente grande como para completar una cantidad de trabajo significativa. El tamaño recomendado del equipo de desarrollo es 6 ± 3 personas.

3.3.2. Artefactos

Scrum contiene tres artefactos principales: *product backlog*, *sprint backlog* y el incremento de producto potencialmente entregable.

3.3.2.1. *Product backlog* (pila del producto)

El *product backlog* representa las necesidades de producto definidas por el cliente conformando los requisitos del producto que serán implementados por el equipo de desarrollo. Es la única fuente de requisitos para cualquier cambio a realizarse en el producto. Evoluciona con el tiempo, a medida que finaliza cada *sprint*; por tanto, es dinámico, cambiando constantemente para entregar un producto adecuado a las condiciones del mercado.

Scrum no especifica un formato específico para el *product backlog*, cada entrada de la lista se denomina “elemento del *product backlog*”. Un ejemplo de un formato específico para los elementos del *product backlog* son las historias de usuario que definen el incremento de valor que los desarrolladores entregarán al cliente.

La gestión del *product backlog* es responsabilidad del *product owner*. Por tanto, es el encargado de añadir, eliminar o actualizar elementos al *product backlog*. Los elementos que están al inicio del *product backlog* tienen la más alta prioridad y son los que entregan mayor valor al cliente.

Las historias de usuario son tarjetas que describen el incremento de valor entregado al cliente. Son escritas por el equipo de desarrollo. Se

recomienda que las historias de usuario tengan el formato presentado en la figura 3.12.

Formato	Ejemplo
<u>Historia de usuario:</u> Como un <u>“tipo de usuario”</u> deseo que el sistema <u>realice “alguna funcionalidad”</u> y así obtener <u>“algún valor de negocio”</u>	<u>Historia de usuario:</u> Como un <u>administrador del sistema</u>, deseo <u>consultar el listado de órdenes de compra para gestionar la compra de insumos de forma más eficiente.</u>

Figura 3.12. Formato de historias de usuario con ejemplo
Fuente: elaboración propia.

Además de esta corta descripción, cada historia de usuario también incluye su priorización, estimación y criterios de aceptación. Los requisitos no funcionales generalmente son capturados como criterios de aceptación de una historia de usuario. Estos criterios de aceptación ayudan a desarrollar pruebas de aceptación, a la vez que contribuyen a definir los requisitos del software.

Una historia de usuario define las funcionalidades y características que debe realizar un determinado rol e identifica el beneficio o valor de negocio que el cliente obtiene con su implementación.

Las historias de usuarios, además de describir requisitos funcionales del producto de software a desarrollar, también pueden ser:

- Solicitudes de cambios de funcionalidades.
- Defectos que han sido detectados y deben ser corregidos.
- Mejoras técnicas relacionadas con el ambiente de desarrollo, sistema operativo, herramientas Computer-Aided Software Engineering (CASE), etc.
- Actividades de gestión del conocimiento o realización de pruebas de conceptos.

Un listado de criterios que permite evaluar si una historia de usuario está bien definida es la colección de criterios Independent, Negotiable, Valuable, Estimable, Small, Testable (INVEST) (Buglione y Abran, 2013):

- ▶ *Independiente (I)*: La historia de usuario debe ser autocontenida, no debe depender de otras historias de usuario.
- ▶ *Negociable (N)*: Hasta que sean parte de un *sprint*, la historia de usuario se puede cambiar y reescribir.
- ▶ *Valiosa (V)*: La historia de usuario debe entregar valor al usuario final.
- ▶ *Estimable (E)*: Se puede realizar una estimación del tamaño de la historia de usuario.
- ▶ *Tamaño apropiado (S)*: La historia de usuario no debe ser grande, de tal manera que pueda ser planificada y priorizada con cierto grado de certeza.
- ▶ *Comprobable (T)*: La historia de usuario debe proporcionar información necesaria para que se le realicen posibles pruebas de desarrollo.

Los elementos de mayor prioridad del *product backlog* son más detallados que el resto de elementos y, generalmente, tienen una mayor granularidad. La definición de las historias de usuario usualmente se trabaja en *workshops*, en que los *stakeholders* más importantes, usuarios finales, analistas de negocios y desarrolladores experimentados, entre otros, participan aportando ideas sobre las historias de usuario. Durante esta sesión, el enfoque está en generar un panorama global del producto y no en los detalles. Este panorama global se denomina “épica”, debido a que son grandes historias de usuario y no se han desglosado aún.

Luego, estas historias épicas se descomponen en historias más pequeñas agrupadas por temas y en un formato gráfico denominado “mapa de historias”, como se observa en la figura 3.13.

Se recomienda que, si un proyecto está conformado por varios equipos *scrum* que trabajan juntos en el mismo producto de software, se utilice un único *product backlog*.

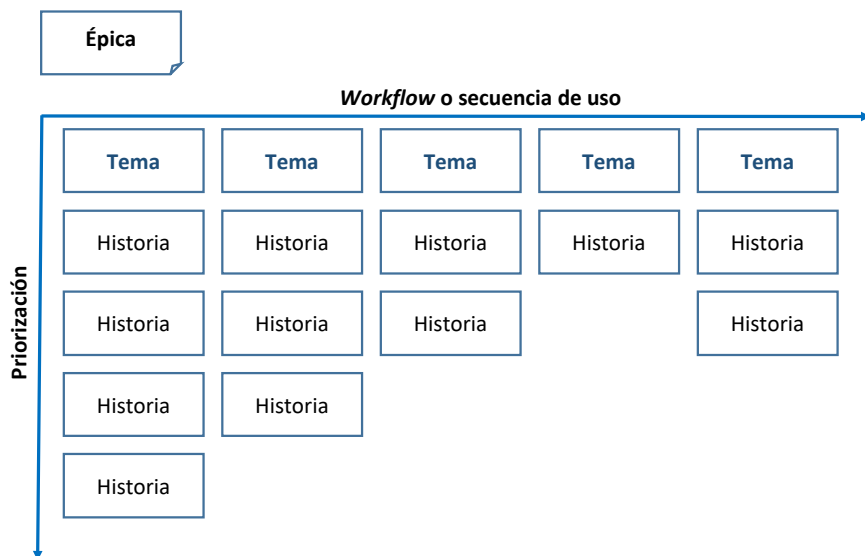


Figura 3.13. Mapa de historias

Fuente: elaboración propia.

3.3.2.2. *Sprint backlog* (pila del *sprint*)

El *sprint backlog* es una lista del conjunto de elementos del *product backlog* seleccionados en un determinado *sprint*, junto con una lista de trabajo que debe realizar el equipo de desarrollo para construir los elementos seleccionados. Con estos dos listados, se obtiene una predicción sobre la funcionalidad que se desarrollará en el próximo incremento y del trabajo necesario para construirlo.

El equipo de desarrollo crea el *sprint backlog* durante el evento “planificación del *sprint*”. Cuando se han seleccionado las historias de usuario a desarrollarse en un *sprint*, el equipo de desarrollo divide las historias de usuario en tareas, las cuales son porciones de trabajo que son realizadas por un único miembro del equipo de desarrollo. Las tareas son desplegadas en un tablero de tareas denominado tablero *kanban*, visible a la organización entera. Las tareas pueden ser de varios tipos: técnicas, como implementar bases de datos o interfaces gráficas de usuario, realización de pruebas, diseño y codificación, o de gestión, como reuniones, revisiones o inspecciones.

A medida que el equipo de desarrollo trabaja y aprende sobre cómo desarrollar el producto, el *sprint backlog* puede cambiar, añadiendo nuevas tareas o eliminando tareas innecesarias, actualizando la estimación de trabajo restante. El *sprint backlog* muestra siempre el estado exacto del *sprint* y la cantidad de trabajo que se planifica realizar durante el *sprint*. Al finalizar un *sprint*, el *sprint backlog* se deja vacío. Cualquier historia de usuario no completada vuelve al *product backlog*, en que es priorizada de nuevo. Únicamente, el equipo de desarrollo puede cambiar el *sprint backlog* durante un *sprint*.

3.3.2.3. Incremento de producto potencialmente entregable

Scrum obliga a obtener un incremento de producto al final de cada *sprint*. Por tanto, el incremento está conformado por todos los elementos del *product backlog* terminados en los *sprints* que se vayan realizando en el proyecto. Cabe resaltar que el incremento al finalizar un *sprint* debe estar “terminado, *done*”, que significa que está en condiciones de ser utilizado por los clientes. La decisión de utilizarse o no es una decisión de negocio que la debe tomar el *product owner*.

3.3.3. Eventos

Las reuniones de *scrum* se denominan eventos y todos son obligatorios. Estos eventos son de corta duración y tienen un marco de tiempo definido sin superar el tiempo máximo permitido.

3.3.3.1. *Sprint*

El evento principal de *scrum* es el *sprint*, el cual es una iteración con una duración máxima de cuatro semanas. Aunque no hay una duración mínima, en la práctica generalmente es una semana. Sin embargo, las duraciones más comunes son dos o tres semanas. Se recomienda que la duración del *sprint* sea consistente a lo largo del proyecto. Al finalizar la iteración, se obtiene un incremento de producto “terminado”. La corta duración de la iteración permite obtener una retroalimentación inmediata por parte del cliente. Una vez que ha finalizado un *sprint*, un nuevo *sprint* comienza.

El *sprint* tiene un objetivo, el cual es una meta que debe ser alcanzada mediante la implementación del *product backlog*. Durante la ejecución del *sprint*, no se pueden realizar cambios que afecten el objetivo del *sprint*, pero el alcance del proyecto puede ser renegociado con el *product owner* y el equipo de desarrollo.

3.3.3.2. Planificación del *sprint*

Es el primer evento que inicia el *sprint*. La planificación del *sprint* tiene dos objetivos. Primero, alcanzar un acuerdo entre el *product owner* y el equipo de desarrollo sobre la funcionalidad que se entregará en el *sprint*. Segundo, realizar una planificación por parte del equipo de desarrollo sobre las tareas para desarrollar la funcionalidad esperada, es decir, el *sprint backlog*. Un resultado adicional de la planificación del *sprint* será la definición del objetivo del *sprint*. El marco de tiempo recomendado para la planificación del *sprint* es de 8 horas para un *sprint* de un mes. Para realizar la planificación, se puede utilizar la técnica de *planning poker*.

3.3.3.3. *Planning poker*

Planning poker es un juego para estimar los tamaños relativos de las historias de usuario, que anima a los miembros del equipo a evaluar la complejidad de una historia de usuario en relación con otras historias. Cada miembro del equipo de desarrollo recibe un conjunto de cartas que tienen los números 1, 2, 3, 5, 8, 13, 21, 40, 100 y ? La última carta representa que se tiene una pregunta para realizar la estimación.

El evento inicia con una lectura de la historia de usuario por parte del *product owner* y su explicación al equipo, el cual realiza preguntas para aclarar su significado. Luego, el *scrum master* solicita que cada miembro determine privadamente un número que represente la complejidad de la historia. Una vez cada miembro haya asignado el número correspondiente, revela su carta, todos al mismo tiempo. Si todos los miembros seleccionaron el mismo número, este número es asignado a la historia de usuario y se pasa a la siguiente. Si los números no concuerdan, los miembros con los valores más bajos y altos explican la razón de su selección. Después de la discusión, se realiza otra ronda de asignación de valores numéricos. Este proceso se repite hasta que haya unanimidad en la determinación del valor numérico. Otras formas de determinar el valor son seleccionar la carta más alta por promedio de valores o por frecuencia de valores. Las estimaciones se realizan comparando el tamaño con la estimación de elementos estimados y de similar esfuerzo. Las estimaciones realizadas también se denominan puntos de historia de usuario.

3.3.3.4. Determinación de la capacidad y velocidad del equipo

Para convertir las estimaciones realizadas en el *planning poker* en tiempo, se requiere conocer la capacidad del equipo. Para ello, se necesita la duración del *sprint*, cuántos días laborables están disponibles en el *sprint* y cuántos días trabajarán los miembros en el *sprint*.

Una vez determinada la capacidad del equipo, se necesita calcular la velocidad del equipo, el cual es el número de puntos de historias de usuario que el equipo puede manejar en un *sprint*. Inicialmente, es imposible conocer esta velocidad. Para ello, el equipo de desarrollo:

- ▶ Estima los puntos de la primera historia de usuario con la técnica de *planning poker*.
- ▶ Descompone la historia de usuario en tareas que necesitan realizarse para entregar las historias.
- ▶ Estima las horas de cada actividad y luego totaliza el tiempo.
- ▶ Calcula el tiempo disponible del equipo en el *sprint*.
- ▶ Si aún resta tiempo para alcanzar la capacidad del equipo, toma más historias de usuario y repite el proceso hasta que se cubra completamente la capacidad del equipo.
- ▶ Totaliza el número total de puntos de historia de usuario para el *sprint*. Este valor final será la velocidad del equipo.

3.3.3.5. *Scrum* diario

Este evento se realiza todos los días durante el *sprint*, participan el equipo de desarrollo, el *scrum master* y el *product owner*. Tiene una duración corta de aproximadamente 15 minutos, en que todos los participantes se reúnen de pie para garantizar su agilidad. El *scrum* diario se realiza para que el equipo *scrum* sincronice sus actividades y pueda crear un plan para las siguientes 24 horas. Para ello, se inspecciona el trabajo realizado desde el último *scrum* diario y se realiza una proyección del trabajo que podría completarse antes del siguiente. El *scrum* diario trata de responder las siguientes preguntas en forma individual por cada miembro del equipo de desarrollo:

- ▶ ¿Qué ha finalizado desde la última reunión diaria?
- ▶ ¿Qué ha planificado hacer antes del siguiente *scrum* diario?
- ▶ ¿Qué obstáculos o impedimentos no le están permitiendo avanzar?

El *scrum* diario es un mecanismo para reducir el riesgo del proyecto, su corta duración y ejecución constante y periódica permite identificar desviaciones y corregirlas rápidamente sin causar grandes impactos en el proyecto. Si surge la necesidad de discutir algún tema, este se posterga para después de la reunión.

3.3.3.6. Ejecución del *sprint*

El equipo de desarrollo presenta las actividades del *sprint backlog* en un tablero, denominado tablero *kanban*, para que sea visible a toda la organización. Los miembros del equipo de desarrollo van tomando elementos del *sprint backlog* y ejecutando estas actividades. Cuando terminan una actividad, seleccionan otra. El equipo es responsable de decidir la mejor asignación de actividades basándose en habilidades y disponibilidad de cada miembro.

Product backlog	Pendiente	En curso	Terminado
Historia 1	Tarea 1.6	Tarea 1.4	Tarea 1.1
Historia 2	Tarea 2.7	Tarea 2.4	Tarea 2.1
Historia 3	Tarea 3.3	Tarea 3.1	
Historia 4	Tarea 4.3	Tarea 4.2	Tarea 4.1

Figura 3.14. Tablero *kanban*
Fuente: elaboración propia.

El tablero *kanban*, que se muestra en la figura 3.14, está conformado por cuatro columnas. En la primera, aparecen las historias de usuario a ser desarrolladas en el *sprint* actual. En la segunda columna, están las actividades pendientes del *sprint backlog*. En la tercera columna, se presentan las

actividades en curso que los miembros del equipo están desarrollando actualmente. Por último, en la cuarta columna, se colocan las actividades que van finalizando. Por tanto, las tarjetas que representan las actividades del *sprint backlog* se van trasladando de izquierda a derecha, representando su iniciación, desarrollo y finalización, hasta completar cada historia de usuario.

Durante la ejecución del *sprint* también se elabora y completa una gráfica denominada *burndown chart*, la cual es una representación visual sobre cómo el *sprint* está progresando. La gráfica muestra la cantidad de trabajo restante (en puntos de historia de usuario) para un determinado día del *sprint*, como se observa en la figura 3.15.

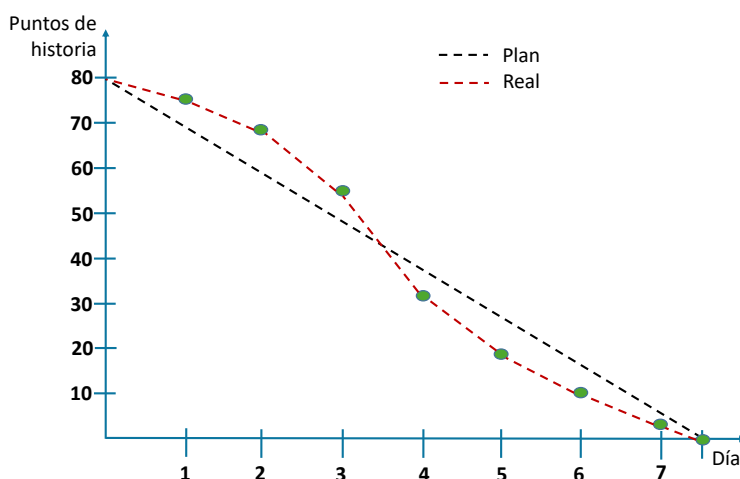


Figura 3.15. *Burndown chart*

Fuente: elaboración propia.

El *burndown chart* también sirve como un indicador temprano que permite detectar problemas en el *sprint* y si el equipo puede cumplir los compromisos pactados para desarrollar el *sprint*.

3.3.3.7. Revisión del *sprint*

Al finalizar el *sprint*, se realiza el evento revisión del *sprint*. A este evento asisten el *product owner*, el *scrum master*, el equipo de desarrollo, involucrados externos (gerentes, directores de mercadeo, usuarios internos y expertos funcionales) e involucrados internos (usuarios externos y socios). Este

evento consiste en presentar únicamente el incremento potencialmente entregable que fue desarrollado durante el *sprint*. La revisión es una demo informal del software desarrollado que permite obtener retroalimentación de todos los participantes y, en especial, de los clientes.

La duración recomendada de la revisión del *sprint* es de cuatro horas para un *sprint* de un mes. Durante la revisión, el incremento presentado es evaluado con respecto al objetivo del *sprint*. Si algunos elementos del *product backlog* no fueron finalizados, se explica su no terminación y son colocados nuevamente en el *product backlog*, luego deben ser priorizados para el siguiente *sprint*.

Durante la revisión del *sprint*, el *product backlog* puede tener ajustes generales para enfocarse en nuevas oportunidades provenientes del mercado o del uso potencial del producto.

3.3.3.8. Retrospectiva del *sprint*

Este evento se realiza una vez un *sprint* ha finalizado y es el último evento antes de comenzar otro *sprint*. Durante este, el proceso es inspeccionado y se identifican medidas para mejorar el proceso. Solo el equipo *scrum* debe estar presente en la retrospectiva del *sprint*. La duración recomendada es de tres horas para un *sprint* de un mes.

La retrospectiva del *sprint* es una oportunidad para que el equipo reflexione y aprenda de sus errores, inspeccionando las actividades que realizó durante el *sprint*. Se inspecciona el trabajo de personas, relaciones, procesos y herramientas. A partir de las mejores prácticas que hayan sido desarrolladas y lecciones aprendidas obtenidas en el transcurso del *sprint*, el equipo *scrum* realiza sesiones de lluvias de ideas para identificar oportunidades de mejora del proceso.

El *scrum master* entrega a cada miembro del equipo un conjunto de notas adhesivas en tres colores. Un color está diseñado para identificar aspectos que se realizaron bien durante el *sprint*. Otro color para identificar aspectos que se hicieron mal durante el *sprint*. El tercer color es para definir aspectos que causaron confusión o que se pueden mejorar o reparar durante el *sprint*. Las notas adhesivas son colocadas en un tablero o pared. Luego, el *scrum master* lee las notas y las discute con el equipo. Al finalizar, este obtiene un plan para implementar las mejoras propuestas y asigna miembros que van a ser responsables de asegurar que las oportunidades de mejora detectadas van a implementarse en próximos *sprints*. Este evento está alineado con los principios de mejora continua de prácticas y procesos.

3.3.3.9. Refinamiento del *product backlog*

Este evento es un proceso continuo en el que se agregan detalles, estimaciones y orden a los elementos del *product backlog*. El evento es realizado por el *product owner* y el equipo de desarrollo. El equipo *scrum* decide cómo y cuándo se realizará el refinamiento, generalmente esta actividad no consume más del 10% de la capacidad del equipo de desarrollo.

3.3.4. Otros conceptos

A continuación, se describirán otros conceptos importantes relacionados con el proceso *scrum*, entre ellos la definición de “terminado”, la deuda técnica y la planificación multinivel.

3.3.4.1. Definición de “terminado”

La definición de “terminado” es un acuerdo entre el *product owner* y el equipo de desarrollo sobre las medidas recolectadas de los elementos del *product backlog* para garantizar la calidad del producto y evaluar cuándo se ha completado el trabajo sobre el incremento. Si un elemento del *product backlog* no cumple estas medidas, el incremento de producto se considera “no terminado” y es rechazado por el *product owner*. La definición de “terminado” es el principal documento de calidad; si no se cumple, no se está entregando un producto con calidad.

Algunos elementos típicos, por ejemplo, para considerar que un elemento del *product backlog* se considere “terminado”, son:

- ▶ Cumple estándares de codificación.
- ▶ Ha pasado por revisiones de pares.
- ▶ Se le han realizado con éxito pruebas unitarias.
- ▶ Ha pasado todas las pruebas de aceptación.
- ▶ La historia de usuario ha sido revisada y aprobada por el *product owner*.
- ▶ El código ha sido integrado.
- ▶ El código está incluido en un repositorio de control de versiones.
- ▶ Se ha refactorizado el código.
- ▶ La documentación ha sido actualizada y verificada.
- ▶ El manual de usuario ha sido actualizado.

La definición de “terminado” puede incluir cualquier otra definición establecida por el equipo de desarrollo que considere apropiada para el producto. Los equipos experimentados deben tener una definición de “terminado” más amplia que incluya criterios más rigurosos para una mejor calidad.

3.3.4.2. Deuda técnica

Desarrollar software de manera ágil conlleva implicaciones en la calidad del código desarrollado. Mejorar la calidad del código requiere una cantidad de retrabajo para colocar las piezas de software en el estado en que deberían haberse encontrado desde el comienzo. Si este no es realizado, el software quedará en un estado por debajo de los estándares. Ward Cunningham acuñó el término *deuda técnica* desde esa perspectiva, pero después incluyó aspectos que quedan por fuera del presupuesto, características como rutinas de manejo de errores, condiciones de excepción, verificaciones de seguridad, procedimientos de copias y recuperación y documentos esenciales, como el diseño de la arquitectura, guías de usuario, modelado de datos y especificaciones de requisitos actualizadas (Sneed, 2014).

La deuda técnica incluye diseño no ajustado a las necesidades actuales, defectos que aún permanecen en el software, cubrimiento insuficiente de pruebas, pruebas manuales excesivas en lugar de automatizadas, integración y gestión de entregas pobres.

El equipo de desarrollo debe “pagar la deuda técnica”, es decir, actualizar continuamente el incremento de producto. Por ejemplo, se puede reducir la deuda técnica mediante la utilización de prácticas técnicas en que se mejora la calidad del código aplicando técnicas de refactorización de código o utilizando una definición fuerte de “terminado”.

3.3.4.3. Planificación multinivel

En *scrum*, se realiza una planificación en múltiples niveles, como se observa en la figura 3.16, y se resumen los seis tipos de planificación en la tabla 3.2.

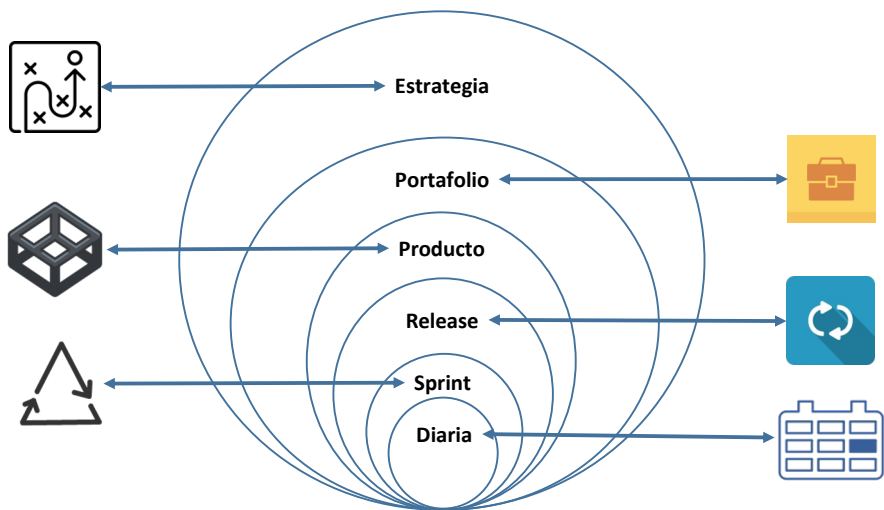


Figura 3.16. Planificación multinivel

Fuente: elaboración propia con base en Rubin (2014).

Tabla 3.2. Niveles de planificación en scrum

Nivel	Horizonte	Roles	Entregables
Planificación de la estrategia	Varios años	Stakeholders	Estrategia organizacional
Planificación del producto	Un año o más	Stakeholders y product owners	Backlog del portafolio con un conjunto de productos en desarrollo
Visión del producto	Muchos meses	Stakeholders y product owners	Visión del producto y características de alto nivel
Planificación de la entrega	De tres a nueve meses	Equipo scrum y stakeholders	Plan de entregas
Planificación del sprint	Cada iteración	Equipo scrum	Objetivo del sprint y sprint backlog
Reunión diaria	Cada día	Equipo de desarrollo y scrum master	Inspección del trabajo actual y organización del trabajo diario

Fuente: elaboración propia con base en Rubin (2014).

3.3.5. Beneficios, desventajas y restricciones

Los beneficios de *scrum* se muestran en la figura 3.17. Las organizaciones que implementan *scrum* tienen clientes satisfechos por estar entregándoles lo que desean, y no solo las características que definieron en etapas tempranas cuando no conocían sus necesidades verdaderas. También obtienen un retorno de inversión mejorada realizando entregas más pequeñas y frecuentes. Además, entregando software integrado, probado y con características de valor de negocio en cada iteración, se reducen costos y las entregas son rápidas. Las organizaciones mostrarán confianza en el éxito en un mundo complejo donde pueden responder y adaptarse rápidamente a las acciones de competidores, clientes, usuarios y otros involucrados. Por último, las personas desarrolladoras disfrutan con su trabajo a través de colaboraciones significativas, relaciones personales mejoradas y confianza mutua entre los miembros de los equipos de desarrollo.

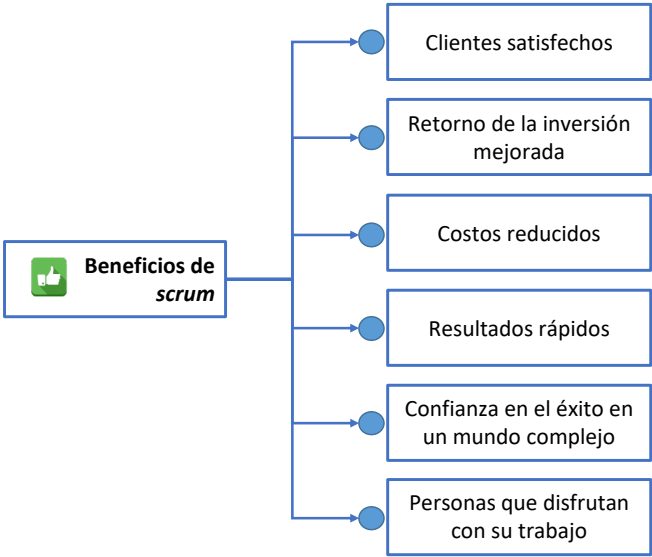


Figura 3.17. Beneficios de *scrum*

Fuente: elaboración propia con base en Rubin (2014).

Entre las desventajas de aplicar *scrum*, se encuentran (Highsmith y Cockburn, 2004; Ionel, 2008):

- ▶ Cuando el proyecto es desarrollado por un cliente externo, este debe estar disponible todo el tiempo y estar involucrado en los entregables periódicos para sugerir o modificar funcionalidades.
- ▶ Si el cliente no tiene un sentido claro de la dirección del producto, el equipo de desarrollo construirá un producto diferente a lo esperado.
- ▶ Implementar *scrum* en proyectos grandes y complejos es viable, pero no es fácil su implementación.
- ▶ La visibilidad de los proyectos por fuera de los *sprints* es baja, es decir, es difícil estimar la duración de un proyecto o su costo.

Las restricciones de alcance, fecha y presupuesto son variables que afectan cómo se logrará el objetivo. La tabla 3.3 presenta diferentes combinaciones de estas variables en las que las restricciones son fijas o flexibles.

Tabla 3.3. Combinaciones de alcance, fecha y presupuesto

Tipo de producto	Alcance	Fecha	Presupuesto
Todo fijo (no recomendado)	Fijo	Fijo	Fijo
Alcance y fecha fijos (no recomendado)	Fijo	Fijo	Flexible
Alcance fijo	Fijo	Flexible	Fijo (no realmente)
Fecha fija	Flexible	Fijo	Fijo

Fuente: elaboración propia con base en Rubin (2014).

Como se observa en la tabla 3.3, los proyectos con todo fijo, es decir, alcance, fecha y presupuesto fijos, no se recomiendan. La opción de alcance y fecha fijos con presupuesto flexible tampoco se recomiendan debido a que dejar el presupuesto flexible no es fácil y significa agregar más personal o que este trabaje horas extra, lo que contradice el principio de ritmo sostenible.

Se recomiendan dos opciones: alcance fijo y fecha fija. Con el alcance fijo, si no se han terminado de completar las características funcionales, la fecha se puede ampliar; por tanto, el presupuesto no puede ser fijo. La segunda opción es la más recomendada para cumplir con los principios de *scrum*: dejar la fecha y el presupuesto fijo pero el alcance flexible, lo cual permite que el marco de tiempo se establezca y el equipo se concentra en las historias de usuario de más alto nivel.

► 3.4. Otros procesos ágiles

Aunque **xp** y *scrum* son extensamente conocidos y utilizados, otros procesos ágiles han sido propuestos y es necesario conocer sus principios y conceptos fundamentales. A continuación, se describirán brevemente los procesos ágiles: Lean Software Development (**LSD**), *crystal*, Dynamic System Development Method (**DSDM**), Feature-Driven Development (**FDD**), Adaptive Software Development (**ASD**) y Test-Driven Development (**TDD**).

3.4.1. Lean Software Development (**LSD**)

Lean development es una práctica de manufactura y producción que considera que el gasto de recursos para cualquier otro objetivo diferente a la creación de valor al cliente es un desperdicio y se debe eliminar (Walton, 1999). Su enfoque orientado al desarrollo de software se denomina **LSD** que se esfuerza por maximizar la creación de valor para el cliente eliminando las tareas innecesarias o que derrochan tiempo y organizando los procesos como un todo. Este proceso de desarrollo tiene su origen en la industria automotriz japonesa (Womack *et al.*, 2007) y busca métodos eficientes sobre la base de maximizar el valor, reducir los desperdicios y abarcar una mentalidad de mejora continua. Los cinco principios fundamentales de **LSD** son (Womack y Jones, 2003):

- *Valor*: Todo lo que un cliente tiene voluntad de pagar. El desperdicio es su opuesto y significa que es todo lo que absorbe recursos, pero no aporta valor.
- *Flujo de valor*: Es una colección optimizada de acciones para llevar el producto al cliente.
- *Flujo*: La organización de actividades como un flujo continuo que elimina discontinuidades y pasos innecesarios en el flujo de valor.
- *Atracción y perfección*: Producir solo lo que es necesario para hacer que las necesidades del cliente sean el conductor principal de las decisiones.

Poppendieck y Poppendieck (2006) compilaron siete principios ampliamente reconocidos en el área de aplicar los conceptos de **LSD** al desarrollo de software. Estos principios son:

- *Eliminar desperdicios*: Enfocarse en las tareas que agregan valor al cliente sin retrasos. Ejemplos de desperdicios son características funcionales innecesarias, conocimiento que se ha perdido, trabajo

parcialmente hecho, reaprendizaje, traspasos de esfuerzos, cambios de tareas, retrasos y defectos.

- ▶ *Aprendizaje constante*: Utilizar ciclos frecuentes de retroalimentación del cliente para tomar decisiones efectivas sobre el producto.
- ▶ *Retrasar compromisos*: Se debe retrasar la toma de decisiones críticas para realizarla con base en el mejor conocimiento disponible. Este aprendizaje continuo minimizará el esfuerzo de desarrollar características que los clientes no encuentren valiosas.
- ▶ *Entrega rápida*: Se responde a las necesidades a medida que el software es diseñado, desarrollado y entregado a través de un flujo de cambios pequeños.
- ▶ *Potenciar al equipo*: El desarrollo está localizado en unidades organizacionales alineadas con el negocio. Se proporciona una fuerza de trabajo experta y se delegan responsabilidades a los desarrolladores de acuerdo con su nivel.
- ▶ *Construcción de calidad*: Se enfoca en la calidad desde el inicio del proyecto y se evita que se corrijan defectos en la etapa de pruebas al final del proceso. Se aplican técnicas basadas en pruebas automatizadas y refactorización.
- ▶ *Ver el todo*: Se enfoca en lograr los objetivos generales que permitan que el software desarrollado esté fundamentado en el dominio del cliente y entregue valor en un contexto más amplio que incluya la resolución de necesidades organizacionales, diseño, implementación y mantenimiento.

La retroalimentación del cliente es central al proceso **LSD**, como se observa en la figura 3.18, que presenta las principales actividades del proceso: investigación, planificación, diseño, prototipos, desarrollo, pruebas, instalación y mantenimiento. Este proceso es adecuado para proyectos de pequeña, mediana y gran escala.

En cuanto a la duración de las iteraciones, deben ser lo suficientemente largas para soportar un ciclo de diseño, desarrollo, pruebas, pero también lo suficientemente cortas para proporcionar retroalimentación frecuente de los clientes. La duración ideal del proceso varía mucho: algunos opinan sobre un marco de tiempo de un mes, otros de un par de semanas y otras compañías recomiendan de 6 a 10 semanas (Poppendieck y Poppendieck, 2003).

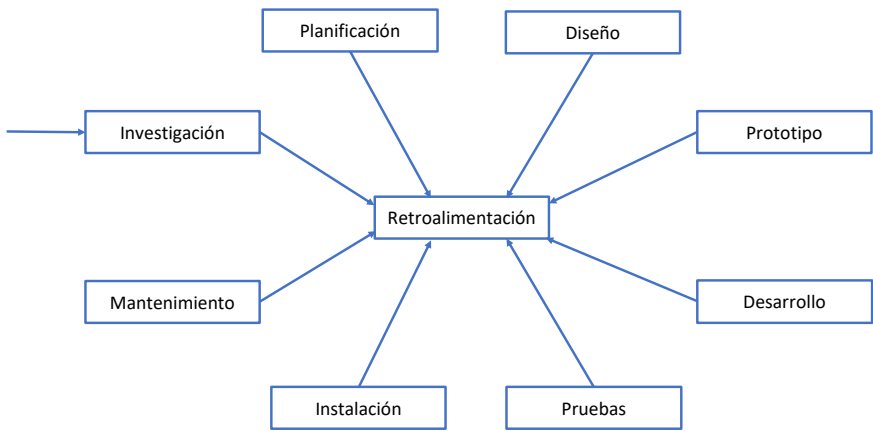


Figura 3.18. Proceso LSD

Fuente: elaboración propia con base en Despa (2014).

3.4.2. Métodos *crystal*

Crystal es un enfoque ligero y adaptable para desarrollar software propuesta por Alistair Cockburn a mediados de la década de 1990. Comprende una familia de procesos ágiles que se pueden aplicar a diferentes proyectos según el tamaño y la complejidad del proyecto (Cockburn, 2004). Su nombre se refiere a las diferentes facetas de una piedra preciosa donde cada una de ellas representa una vista diferente de principios y valores.

El marco de trabajo de *crystal* se presenta en la figura 3.19 que incluye los factores que influyen en la selección de un proceso en particular. El eje X representa el tamaño del personal y el eje Y la criticidad del sistema. *Crystal* define cuatro niveles de criticidad (Abrahamsson *et al.*, 2002):

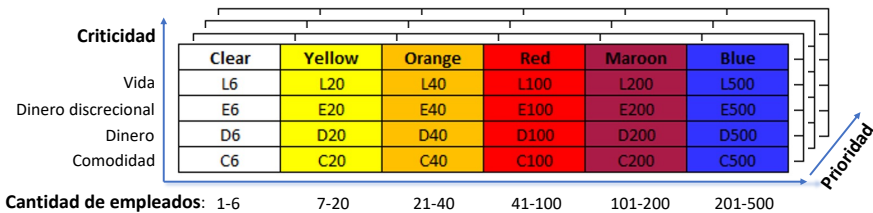


Figura 3.19. Métodos *crystal*

Fuente: elaboración propia con base en Abrahamsson *et al.* (2002).

- ▶ *Vida (L)*: Una falla del sistema es crítico y puede causar pérdida de vidas.
- ▶ *Dinero esencial (E)*: Una falla del sistema puede causar pérdida de dinero.
- ▶ *Dinero discrecional (D)*: Una falla del sistema puede causar pérdida de dinero, pero puede ser corregida consultando el manual de usuario del sistema.
- ▶ *Comodidad (C)*: Una falla del sistema puede causar pérdida de comodidad del cliente.

A cada proceso se le asigna un color diferente que representa su complejidad y criticidad:

- ▶ *Crystal clear*: Es el proceso más ligero con equipos pequeños de dos a seis personas que trabajan en el mismo sitio en sistemas no críticos. Además, la documentación es ligera.
- ▶ *Crystal yellow*: Equipos entre 7 y 20 miembros con fácil comunicación entre el personal de desarrollo, incrementos mensuales, se trabaja con usuarios reales, se define una sentencia de misión crítica, se definen áreas de código con claros propietarios y se incorporan pruebas automatizadas.
- ▶ *Crystal orange*: Equipos entre 21 y 40 miembros con habilidades interfuncionales. Incluye desarrollo incremental y entregas cada 3-4 meses.
- ▶ *Orange web*: Equipos entre 21 y 40 miembros. Incluye trabajo regular, procesos básicos, progreso máximo con mínimas distracciones y maximización de la eliminación de defectos. Orientado al desarrollo de aplicaciones web (no se muestra en la figura 3.20).
- ▶ *Crystal red*: Con equipos entre 41 y 100 miembros. Los equipos se forman y se dividen según el trabajo requerido.
- ▶ *Crystal maroon*: Con equipos entre 101 y 200 miembros. Para proyectos de gran tamaño. Además, los métodos definidos son diferentes y según la necesidad del software.
- ▶ *Crystal blue*: Con equipos entre 201 y 500 miembros.

Cada proceso cuenta con un conjunto de políticas estándar, artefactos, procedimientos (llamados asuntos locales), herramientas, roles, actividades y estándares. Los métodos *crystal* comparten siete propiedades comunes (Abrahamsson *et al.*, 2002):

1. *Entrega frecuente*: Se realizan entregas desde semanales hasta trimestrales, de tal manera que se obtiene retroalimentación rápida del cliente y se puedan detectar problemas tempranamente.
2. *Mejora reflexiva*: Por medio de la realización de talleres periódicos se detectan problemas y se proponen mejoras al proceso.
3. *Comunicación cerrada u osmótica*: Consiste en que el personal de desarrollo debe estar en un mismo espacio de trabajo para que la comunicación fluya rápidamente, se respondan preguntas inmediatamente y se conozca el trabajo de los demás.
4. *Seguridad personal*: Las personas tienen libertad para opinar y sugerir ideas, lo cual permite la generación de confianza para hablar de problemas o sucesos que ocurran durante el proyecto.
5. *Enfoque*: El enfoque tiene dos puntos de vista. En el primero, el personal se debe enfocar en tareas largas durante un proyecto para que se pueda medir su progreso. Se tienen dos recomendaciones: trabajar durante dos horas seguidas sin interrupciones y se asignan desarrolladores a un proyecto al menos durante dos días antes que sea cambiado a otra actividad. El segundo punto de vista se centra en tener una definición clara de los objetivos del proyecto que los desarrolladores conozcan para enfocarse en áreas particulares.
6. *Fácil acceso a usuarios expertos*: El proyecto cuenta con un experto quien también es un usuario final, con el cual se reúne dos horas a la semana para responder preguntas y sugerir mejoras y soluciones.
7. *Ambiente técnico con pruebas automatizadas, gestión de configuración e integración frecuente*: Con la utilización de estas técnicas, los cambios frecuentes que se realicen en el producto son rápidamente verificados, los problemas son resueltos tempranamente y tienen menos probabilidades de crecer.

3.4.3. Dynamic Systems Development Method (DSDM)

El **DSDM** surge de los trabajos de Jennifer Stapleton en la década de 1990 en el Reino Unido (Stapleton, 1997). El Consorcio **DSDM** es el propietario y responsable de este proceso. El **DSDM** utiliza un ciclo de vida iterativo en el que se fragmenta el proyecto en periodos cortos. Se recomienda su utilización en proyectos con requisitos de tiempo fijo y con funcionalidad varia-

ble según las necesidades de los *stakeholders*. Los principios fundamentales de **DSDM** son (**DSDM Consortium**, 2008):

- ▶ El involucramiento activo de los usuarios es obligatorio.
- ▶ Los equipos **DSDM** están facultados para tomar decisiones.
- ▶ El foco del proceso está en la entrega frecuente de productos.
- ▶ La aptitud para el propósito del negocio es el criterio esencial para la aceptación de entregables.
- ▶ El desarrollo iterativo e incremental es necesario para converger en una solución de negocio exacta.
- ▶ Todos los cambios durante el desarrollo son reversibles.
- ▶ Los requisitos son la línea base a un alto nivel.
- ▶ Las pruebas se integran a lo largo del ciclo de vida.
- ▶ Un enfoque cooperativo y colaborativo entre los *stakeholders* es esencial.

El proceso **DSDM** divide el ciclo de vida de desarrollo en las siguientes fases, como se observa en la figura 3.20 (**DSDM Consortium**, 2008):

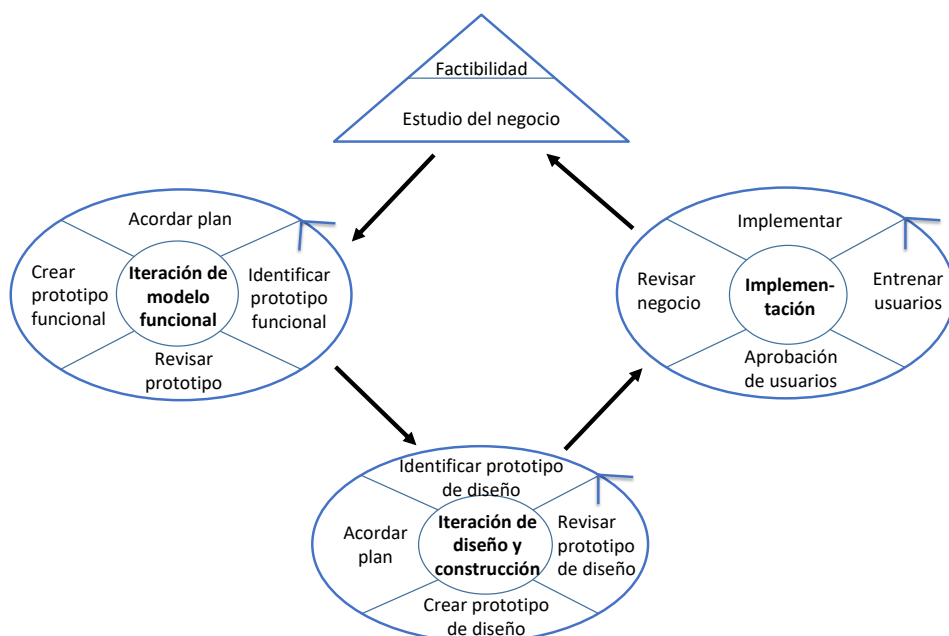


Figura 3.20. Proceso DSDM

Fuente: elaboración propia con base en **DSDM Consortium** (2008).

- *Preproyecto*: El proyecto es iniciado apropiadamente.
- *Estudio de factibilidad*: Se evalúa si el **DSDM** es el enfoque apropiado para el proyecto. Si se decide por este, se analiza la factibilidad técnica, de costos y riesgos.
- *Estudio del negocio*: Se enfoca en el entendimiento de los requisitos de negocio y restricciones técnicas asociadas con el proyecto.
- *Iteración de modelo funcional*: Involucra la creación de un modelo funcional, desarrollo de prototipos y un modelo de análisis.
- *Iteración de diseño y construcción*: Se refina y prueba el prototipo funcional creado. El diseño y la construcción son iterativos.
- *Implementación*: Se transfiere el sistema al ambiente de operación para que sea probado por los usuarios y se proporciona el entrenamiento requerido a los usuarios.
- *Posproyecto*: Asegura el mantenimiento efectivo de la solución entregada.

Las fases de iteración de modelo funcional e iteración de diseño y construcción comprenden ciclos de cuatro actividades:

- Identificar lo que se va a producir.
- Estar de acuerdo sobre cómo y cuándo hacer el producto.
- Crear el producto.
- Verificar que el producto se ha producido correctamente mediante revisión de documentos, presentación de prototipos o pruebas de partes del producto.

El **DSDM** especifica hasta quince roles entre usuarios y desarrolladores, entre algunos de ellos están coordinador técnico (responsable de la arquitectura y calidad técnica), desarrolladores, desarrolladores sénior, usuario “embajador” (proporciona conocimiento del dominio), asesor de usuarios, visionario (usuario que participa en el proyecto) y patrocinador ejecutivo (toma las decisiones finales).

La priorización de los requisitos se realiza utilizando las reglas MoSCoW (Voigt *et al.*, 2004):

- *Must have*: Todas las características en este grupo deben implementarse obligatoriamente. Si no se entregan, el producto no funcionará.
- *Should have*: Las características son importantes y aportan un valor significativo, pero pueden omitirse si las limitaciones de tiempo ponen en peligro la entrega de cualquier característica “*Must have*”.

- ▶ *Could have*: Estas características mejoran el sistema con elementos funcionales, que pueden ser fácilmente reasignados a un tiempo posterior.
- ▶ *Wan't to have*: Usualmente estas características solo sirven a un grupo limitado de usuarios y son de poco valor.

Los proyectos **DSDM** consisten de uno o dos equipos, en los que el segundo equipo puede tener la responsabilidad de ejecutar pruebas. Se sugiere que un equipo tenga cinco personas (que incluye expertos externos). Si el trabajo a entregar es mayor, se requiere una estructura multiequipo (Voigt *et al.*, 2004).

3.4.4. Feature Driven Development (FDD)

El **FDD** es un proceso iterativo e incremental basado en iteraciones cortas. Fue publicado en 1999 después de una aplicación exitosa en un banco internacional en Singapur en 1997 (Coad *et al.*, 1999). Está compuesto por cinco procesos, como se observa en la figura 3.21.

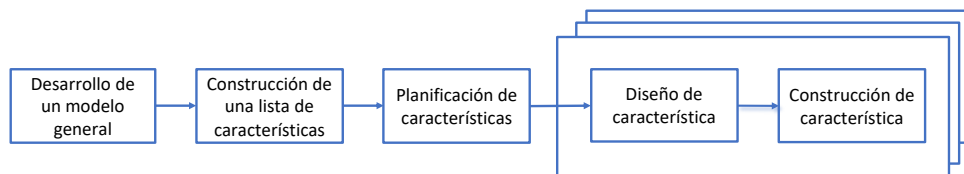


Figura 3.21. PROCESO FDD

Fuente: elaboración propia con base en Palmer y Felsing (2002).

- ▶ *Desarrollo de un modelo global*: Se desarrolla un modelo de objetos de dominio por parte de los desarrolladores y expertos del dominio que captura el alcance y contexto de cada área del dominio del problema, sus abstracciones clave y sus relaciones en el sistema.
- ▶ *Construcción de una lista de características*: La lista creada contiene características que describen un objeto o sus métodos en el modelo de dominio. Una característica es definida como una pieza pequeña de funcionalidad de un sistema, entregable y de valor para el cliente. Las características se agrupan en conjuntos de características y estos en áreas sujeto o dominios sujeto. Cada uno de estos conjuntos tiene sus propios formatos de descripción.

- *Planificación por característica:* Se realizan estimaciones de las características definidas. Los conjuntos de características son ordenados de acuerdo con prioridades del cliente, estimaciones de tiempo y dependencias técnicas. Luego, son asignados a equipos de características individuales que están conformados por desarrolladores individuales. Cada desarrollador es responsable de crear y mantener sus características en el modelo de dominio.
- *Diseño y construcción por característica:* El diseño y la construcción de cada conjunto de características es iterado en un paquete de trabajo por un equipo de características. Luego, los desarrolladores individuales implementan e integran sus partes en el sistema.

Con respecto a los roles implicados en el **FDD**, existen tres categorías en las que un miembro del equipo puede ejecutar múltiples roles y un único rol puede ser compartido por varios miembros del equipo. A continuación, se describen brevemente las categorías de roles (Palmer y Felsing, 2002).

- *Roles clave:* Administrador de proyectos, arquitecto jefe, administrador de desarrollo, programador jefe (líder de equipo de características que prepara paquetes de trabajo), propietario de partes y experto de dominio.
- *Roles de soporte:* Administrador de entregas, gurú de lenguaje de programación, ingeniero de construcción, administrador del sistema y de herramientas.
- *Roles adicionales:* testers, implementadores (mantienen la compatibilidad de los datos y preparan nuevas entregas) y escritores técnicos (preparan la documentación del usuario).

Por último, las prácticas recomendadas por el **FDD** son modelado de objetos de dominio, desarrollo por características, propiedad del código, equipos de características, inspecciones, construcción regular de la planificación, gestión de configuración y reportes y visibilidad de los resultados.

3.4.5. Adaptive Software Development (ASD)

El **ASD** fue introducido por James A. Highsmith en 2000 (Chowdhury y Huda, 2011). Se fundamenta en la teoría de sistemas adaptativos complejos. Por tanto, los proyectos de software se consideran sistemas adaptativos complejos compuestos de agentes (*stakeholders*), entornos (organizacional y tecnológico) y salidas (producto de software) (Riehle, 2000). Es un proceso dedicado al aprendizaje continuo y orientado al cambio, la reevaluación,

la contemplación de un futuro incierto y una colaboración intensa entre desarrolladores, gestión y clientes (Highsmith, 2002).

El proceso **ASD** se compone de fases que se caracterizan por estar enfocadas en la misión, están basadas en características, son iterativas, tienen marcos de tiempo específicos, están orientadas por los riesgos y son tolerantes a los cambios. El proceso **ASD**, ilustrado en la figura 3.22, se compone de las siguientes fases (Highsmith, 2013):

- *Investigación*: La misión y los objetivos del proyecto son definidos entendiendo las restricciones, estableciendo la organización del proyecto, realizando estimaciones iniciales de tamaño y alcance, e identificando riesgos clave. Luego, se planifica la cantidad de iteraciones del proyecto asignando un marco de tiempo y objetivos a cada una de las iteraciones. Cada iteración debe entregar un conjunto de características al cliente.
- *Colaboración*: Se desarrollan componentes en forma concurrente. Estos son refinados en un proceso iterativo. La colaboración se considera un acto de creación compartida fomentado por la confianza y el respeto del equipo de desarrollo, los clientes, los consultores externos y los proveedores. Los equipos deben colaborar en problemas técnicos, requisitos empresariales y toma rápida de decisiones.
- *Aprendizaje*: Se capturan lecciones aprendidas sobre la calidad del resultado desde una perspectiva del cliente, una perspectiva técnica (revisiones técnicas y programación en parejas), la funcionalidad entregada y las prácticas utilizadas por los miembros del equipo, y el estado del proyecto (con respecto al plan, y así enfocar esfuerzos de replanificación).

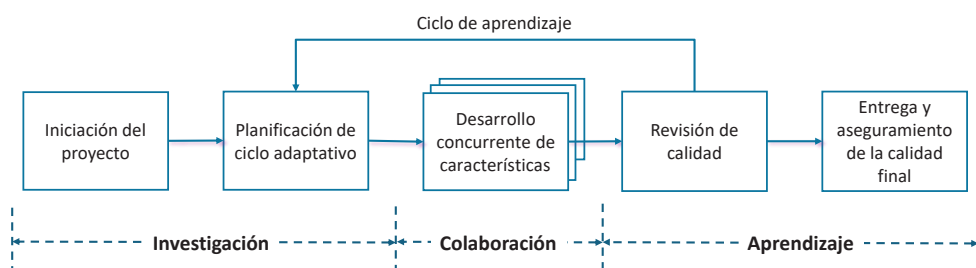


Figura 3.22. PROCESO ASD

Fuente: elaboración propia con base en Highsmith (2013).

3.4.6. Test Driven Development (TDD)

El **TDD** es una técnica de desarrollo de software incluida en **XP** que conduce a un desarrollo evolutivo de software en pequeños pasos (Beck, 2000). Se centra en tres pilares fundamentales (Beck, 2002):

- La implementación de solo las funciones que el cliente necesita.
- La minimización del número de defectos que llegan a la fase de producción.
- La producción de software modular, altamente reutilizable y preparado para el cambio.

El proceso de desarrollo del **TDD** está conformado por tres actividades (Beck, 2002):

- *Prueba*: Escribir la especificación del requisito como una prueba.
- *Aplicación*: Implementar el código para que pase las pruebas.
- *Refactorización*: Refactorizar el código para eliminar duplicidades y realizar mejoras.

El ideal es que cada una de estas actividades corresponda a un ciclo, el cual es el marco de tiempo entre dos ejecuciones sucesivas de pruebas unitarias. Las tres actividades componen una iteración ideal **TDD**. El orden de las tareas es importante para la conformidad del proceso. Finalmente, el proceso **TDD** consiste de una secuencia de iteraciones, como se observa en la figura 3.23.

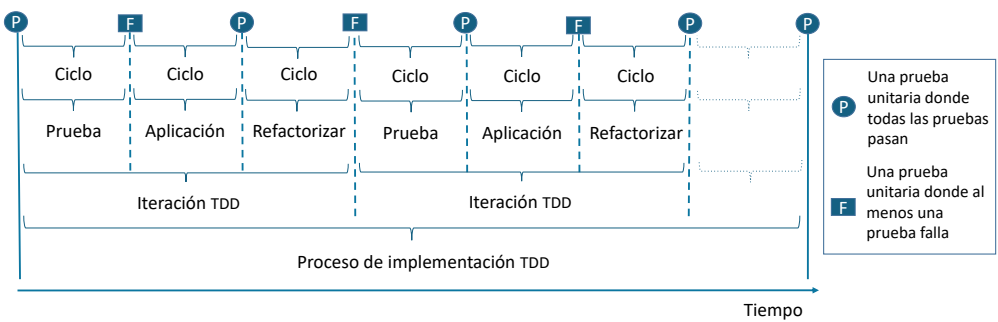


Figura 3.23. Proceso TDD

Fuente: elaboración propia con base en Müller y Höfer (2007).

En el **TDD**, las pequeñas iteraciones frecuentes son utilizadas para desarrollar software y cada microiteración puede tomar solo unos pocos

minutos. De esta manera, se obtiene un diseño del sistema global que permite al cliente obtener retroalimentación frecuente que posibilita cumplir las necesidades actuales del sistema en un contexto organizacional. El **TDD** anima la obtención de un producto viable mínimo tan rápido como sea posible. La refactorización promueve escribir código compacto y pequeño que sea fácilmente probado.

► 3.5. Comparación entre procesos ágiles y tradicionales

A continuación, se realiza una comparación entre los diferentes procesos ágiles presentados en este capítulo y una comparación entre los procesos ágiles y los procesos tradicionales descritos en el capítulo anterior.

3.5.1. Comparación entre procesos ágiles

El criterio principal de los procesos ágiles es el desarrollo de software que logre la satisfacción del cliente por medio de entrega continua de software (Mahmud y Abdullah, 2015).

Aunque los diferentes procesos ágiles pueden diferir ampliamente en la forma en que enfocan el desarrollo de software, comparten características clave, como la colaboración estrecha y cara a cara entre los equipos desarrolladores y equipos de negocio (Cugola y Ghezzi, 1998; Nerur *et al.*, 2005). Sin embargo, la selección del proceso ágil adecuado no es fácil, se deben estudiar las características diferenciadoras para escoger el proceso apropiado a las necesidades de los *stakeholders* (Abrahamsson *et al.*, 2003; Coram y Bohner, 2005; Cugola y Ghezzi, 1998).

En la tabla 3.4, se presenta una comparativa entre los diferentes procesos ágiles presentados en este capítulo.

Los procesos ágiles difieren significativamente de los procesos tradicionales. En la tabla 3.5, se presenta una comparativa entre estos procesos basada en diferentes criterios.

Tabla 3.4. Diferencias entre procesos ágiles

Criterio	XP	Scrum	Lean	Crystal	DSDM	FDD	ASD	TDD
Tipo de proceso	Desarrollo	Gestión	Desarrollo	Desarrollo	Desarrollo	Desarrollo	Gestión	Desarrollo
Tamaño de equipo	6-10	5-10	No especificado	Tamaños diverso	Máximo 6	Máximo 800 con 3-5 por iteración	No especificado	No especificado
Gestión del equipo	Autoorganizado	Autoorgani- zado	Autoorganizado	Autoorganizado	Autoorganizado	Controlado o centralizado	Autoorganizado	Autoorganizado
Equipos distribuidos	No	Sí	No	No	Sí	Sí	Sí	Sí
Recolección de requisitos	Historias de usuario	<i>Product backlog</i>	Requisitos de valor máximo	Casos de uso	Estudio del negocio	Características	Características	Requisito como pruebas
Modelado de dominio	No	No	No	Opcional	Sí	Sí	No	No
Prioridad de tareas	Cliente	<i>Product Owner</i>	Usuario	Usuario	Técnica MoSCoW	Experto de dominio	Cliente	Cliente
Tiempo de iteración	1-4 semanas	4 semanas	2-10 semanas	Tiempos varios	Máximo 6 semanas	2 semanas máximo	2-8 semanas	Pocos minutos
Manejo del cambio	Cambios menores: Oralmente Cambios mayores: Documentos	En el <i>backlog</i> en un nuevo <i>sprint</i>	Orientado a eliminar desperdicios	Cambios mayores según el color del método	Todos los cambios son reversibles	Cambios menores: Conjunto de características Cambios mayores: Modelo de dominio	En todas las etapas del proyecto	Una vez una prueba falla, se cambia el código

Criterio	XP	Scrum	Lean	Crystal		DSDM	FDD	ASD	TDD
Interacción con clientes	Fases de exploración, iteración y entrega	<i>Product backlog</i> , planificación y revisión del <i>sprint</i>	Continuamente durante todo el proyecto	Fase de recolección de requisitos		Fase de estudio del negocio y fase implementación	Fase inicial y fase de planificación de características	Fases de especulación colaboración aprendizaje	Definición de pruebas y refactorización
Desarrollo	Enfoque oo	No especificado	No especificado	Enfoque oo	No especificado	Enfoque oo	No especificado	No especificado	No especificado
Pruebas	En componente	En <i>sprint</i>	En unidad	En unidad	En unidad	En unidad e iteración	En componente	En unidad	
Aseguramiento de la calidad	Inspecciones por pares e inspecciones de equipo	Revisión del <i>sprint</i>	Retroalimentación continua del cliente	Revisiones e inspecciones	Estándares estrictos de calidad	Inspección de diseño y código	Calidad del cliente, técnica, funcionalidad y estado del proyecto	Alta eliminación de defectos y refactorización	
Fortaleza	Simplicidad	Interacción del equipo	Tiempo de entrega reducido por eliminación de desperdicio	Tamaño y criticidad del proyecto	Incluye implementación posproyecto, documentación completa	Modelado de negocios	Manejo efectivo del cambio y del alcance, permite innovación	Alta calidad del código, centrado en el usuario	

Fuente: elaboración propia con base en Kanwal et al. (2010).

Tabla 3.5. Diferencias entre procesos ágiles y procesos tradicionales

Criterio	Procesos tradicionales	Procesos ágiles
Origen	Heurísticas de prácticas de producción de código	Estándares seguidos por el entorno de desarrollo
Control	Centrado en el proceso	Centrado en las personas
Estilo de gestión	Comando y control	Liderazgo y colaboración
Gestión del conocimiento	Explicito	Tácito
Asignación de roles	Individual, favorece la especialización	Equipos autoorganizados, anima el intercambio de roles
Comunicación	Formal y solo cuando es necesaria	Informal y continua
Involucramiento del cliente	Durante el análisis del proyecto, en reuniones periódicas y durante las pruebas de aceptación	Continuo debido a que el cliente forma parte del equipo de desarrollo
Ciclo de proyecto	Guiado por tareas o actividades	Guiado por características del producto
Modelo de desarrollo	Modelos del ciclo de vida	Modelos de entrega evolutivos
Estructura organizacional	Mecanicista (burocrático con alta formalización)	Orgánico (flexible y participativo)
Tecnología	Sin restricciones	Favorece la tecnología oo
Localización del equipo	Predominantemente distribuido	Predominantemente en el mismo sitio
Tamaño del equipo	Grupos grandes (mayores que 10)	Grupos pequeños (menores que 10)
Cantidad de artefactos	Muchos artefactos	Pocos artefactos
Cantidad de roles	Muchos roles	Pocos roles
Arquitectura de software	La arquitectura de software es esencial y se expresa mediante modelos	Menos énfasis en la arquitectura de software
Aprendizaje continuo	Frecuentemente no se incluye	Adoptado
Planificación del proyecto	Al comienzo	Continua
Mecanismos de retroalimentación	No son fáciles de obtener	Numerosos mecanismos disponibles
Gestión del cambio	Cierta resistencia a los cambios	Especialmente preparados para cambios durante el proyecto
Cuestiones contractuales	Existe un contrato prefijado	No existe contrato tradicional o al menos es flexible
Documentación	Substantial	Mínima

Fuente: elaboración propia con base en Canós *et al.* (2003), Nerur *et al.* (2005) y Schuh (2004).

3.5.2. Comparación entre procesos ágiles y procesos tradicionales

Los procesos ágiles ofrecen muchos beneficios tangibles sobre los procesos tradicionales, entre ellos, el tiempo de lanzamiento de productos al mercado se mejora, hay una mejor productividad y calidad de software, y se reducen los costos de desarrollo (Barnett y Schwaber, 2004; Fitzgerald *et al.*, 2006; Reifer, 2002).

Una diferencia clara entre los procesos ágiles y los tradicionales es el involucramiento de los usuarios. Los procesos tradicionales requieren un alto nivel de involucramiento al comienzo y al finalizar el proceso, en concreto, en la fase de definición de requisitos y de pruebas de aceptación. Mientras en los procesos ágiles la colaboración se mantiene a largo del proyecto, como se presenta en la figura 3.24.

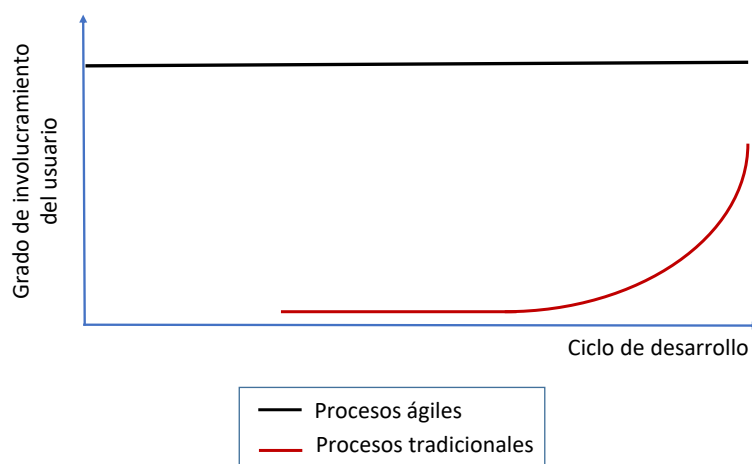


Figura 3.24. Involucramiento del usuario en procesos ágiles y tradicionales

Fuente: elaboración propia con base en Bessam *et al.* (2009).

► 3.6. Escalamiento ágil

La mayoría de las grandes organizaciones desarrollan proyectos a gran escala por equipos de equipos o equipos de equipos de equipos. Las grandes

organizaciones están interesadas en incorporar procesos ágiles en equipos más grandes, que incluye coordinación, comunicación entre equipos y clientes, y supervisión.

El éxito de los métodos ágiles para equipos pequeños inspiró a las grandes organizaciones a aplicarlos a gran escala mediante el uso de marcos ágiles de escalado. Los procesos ágiles más reconocidos como *XP* y *scrum* deben modificarse antes de que se empleen en grandes organizaciones. Para lograr esta tarea, las grandes organizaciones necesitan elegir entre diferentes métodos ágiles de escalado para adaptarse a una visión única en equipos grandes y múltiples (Alqudah y Razali, 2016).

Los diferentes métodos de escalado fueron desarrollados para resolver el problema del tamaño del equipo y otros problemas asociados con la participación del cliente, las limitaciones del proyecto, la aprobación de casos de negocio, la interacción con socios y usuarios finales, y la realización de los beneficios del proyecto (Vaidya, 2014).

El 17.º Reporte del Estado Ágil muestra los métodos y enfoques más populares para el escalamiento ágil (figura 3.25). SAFe es el método más ampliamente utilizado. SAFe supera significativamente al siguiente método de escalado más cercano, *scrum* de *scrums*.

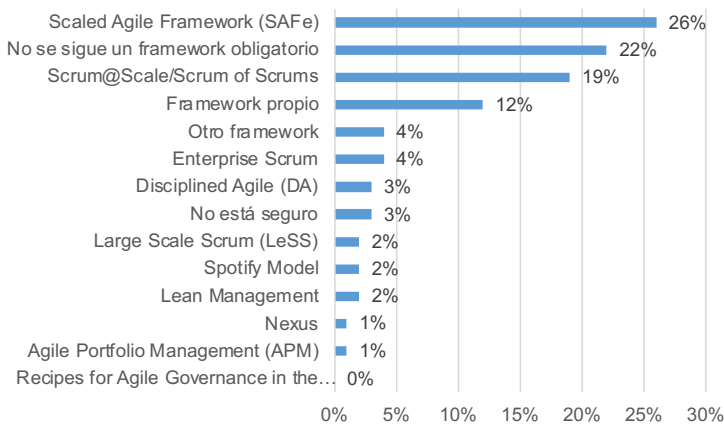


Figura 3.25. Métodos y enfoques de escalamiento ágil
Fuente: elaboración propia con base en Digital.ai (2023).

A continuación, se describirán las principales características, los roles y las prácticas de los métodos SAFe, *scrum* de *scrums*, *DAD* y LeSS.

3.6.1. *Scrum de scrums*

Scrum de scrums son varios equipos *scrum* que trabajan y se coordinan en el mismo producto o hacia el mismo objetivo. Si el producto es complejo, se pueden organizar varios equipos *scrum* y el equipo de más alto nivel se denomina *scrum de scrums*, como se observa en la figura 3.26. Cada equipo tiene su propio *scrum master*. Cada equipo determina cuál miembro participará en el *scrum de scrums*, puede ser el *scrum master*, pero no necesariamente. La persona elegida puede cambiar con el tiempo, de acuerdo con el momento en que se encuentre el desarrollo.

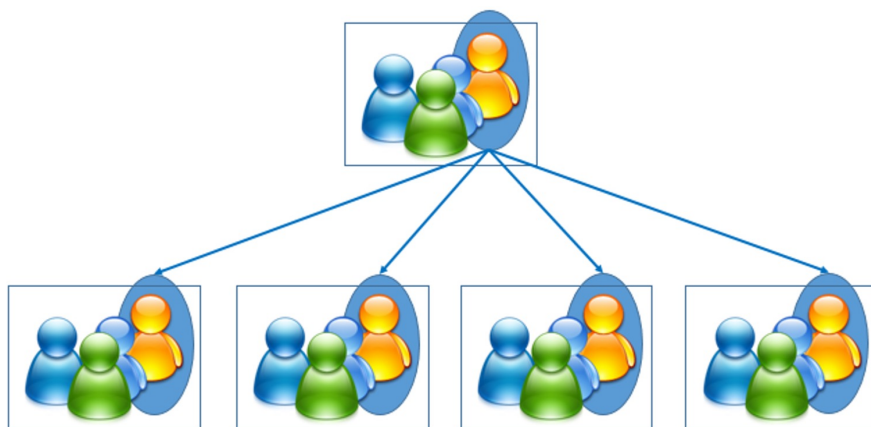


Figura 3.26. *Scrum de scrums*

Fuente: elaboración propia con base en Rubin (2014).

También puede usar *scrum de scrums* si el equipo de producto está en varias ubicaciones y cada ubicación tiene equipos *scrum* completamente funcionales. Esto hace que la ejecución del producto en múltiples ubicaciones sea manejable (Mundra *et al.*, 2013).

Los equipos *scrum de scrums* se coordinan a través de una reunión en la que participa un representante de cada equipo. Esta reunión es una práctica de coordinación interequipos. El formato básico de la reunión se parece a la reunión diaria de *scrum*, excepto que trata con equipos en lugar de miembros del equipo. El tiempo de reunión es de 15 a 30 minutos y su frecuencia puede ser diaria o esporádica con un mínimo de una vez a la semana.

Las tres preguntas respondidas en la reunión diaria de *scrum* pueden cambiarse ligeramente: 1) ¿Qué hizo el equipo desde la reunión anterior que sea relevante para algún otro equipo? 2) ¿Qué hará el equipo en la próxima reunión que sea relevante para otros equipos? 3) ¿Qué obstáculos tiene el equipo que afectan a otros equipos o requieren ayuda de ellos? (Paasivaara *et al.*, 2012).

Con respecto al *product backlog* en una estructura de varios equipos *scrum*, se recomienda que se trabaje con único *product backlog*.

3.6.2. Marco de referencia ágil escalado

Dean Leffingwell diseñó el marco de referencia ágil escalado (Scaled Agile Framework [SAFe]) para el escalamiento en grandes empresas (Alqudah y Razali, 2016; Vaidya, 2014). Incorpora prácticas de *scrum*, **XP**, *kanban* y *lean*. Los valores centrales de SAFe son construir con calidad, transparencia, alineación y ejecución del programa. La figura 3.27 ofrece una descripción general del marco SAFe.

SAFe ofrece cuatro niveles (Leffingwell, 2011):

- *Equipo*: Adopta *scrum* con prácticas de ingeniería **XP**, pero también es posible usar *kanban*.
- *Programa*: Define el concepto de tren de lanzamiento ágil (Agile Release Train [**ART**]), que es la analogía de los *sprints* en equipo, trabajando en un marco de tiempo más lento. Los **ART** siguen las iteraciones Hardening, Innovation and Planning (**HIP**), o endurecimiento, innovación y planificación, para desarrollar incrementos potenciales de envío o incrementos de programa. Estos últimos se planifican durante los días de planificación del lanzamiento.
- *Programa*: Contiene roles adicionales, por ejemplo, equipo del sistema, gerente de producto, arquitecto del sistema, ingeniero de trenes de lanzamiento ágil y equipo de gestión de versiones.
- *Portafolio*: La planificación se realiza como épicas que definen grandes iniciativas de desarrollo.
- *Flujo de valor opcional*: Apoya el desarrollo de soluciones grandes y complejas que requieren **ART** múltiples y sincronizados.

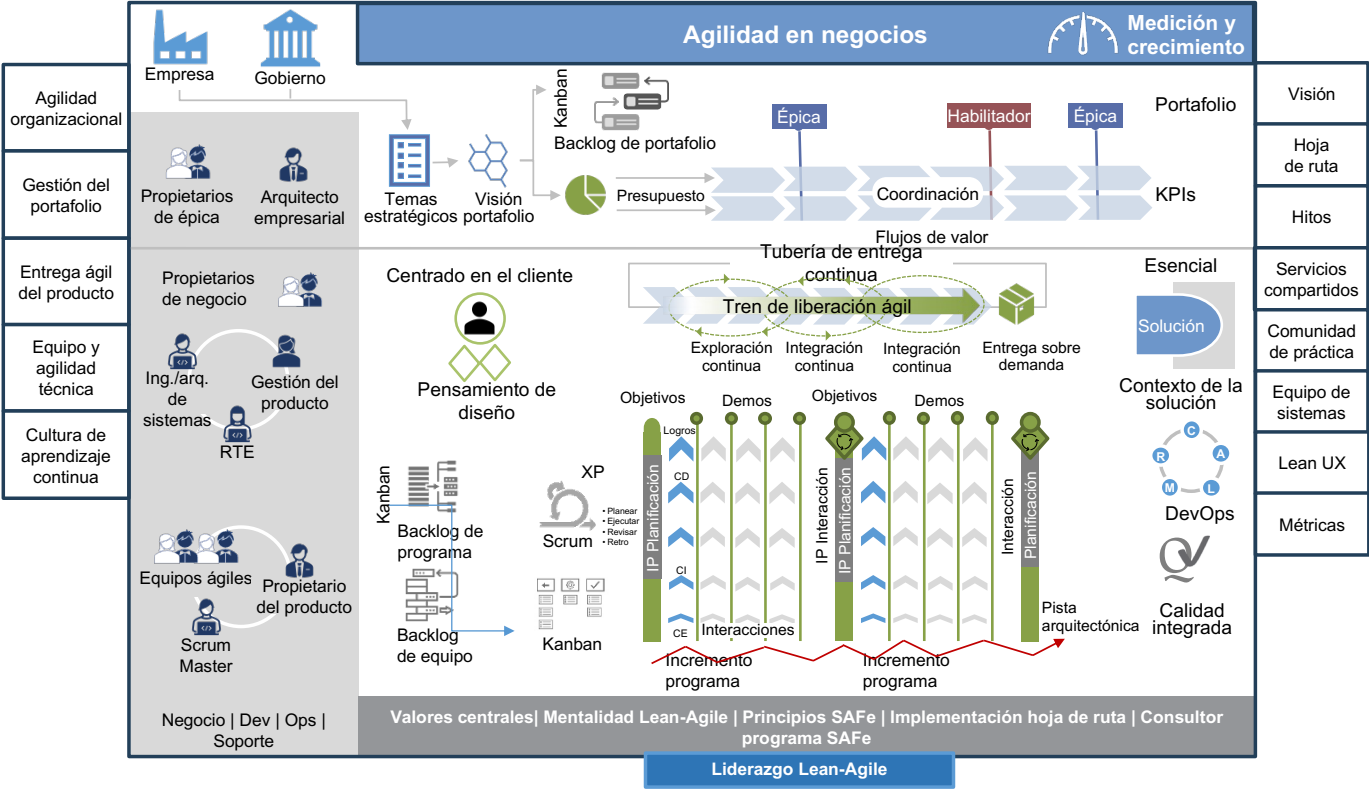


Figura 3.27. Framework SAFe

Fuente: elaboración propia con base en Scale Agile Framework (2020).

Para adoptar SAFe, el documento técnico de SAFe 4.0 sugiere el patrón de “Implementación de SAFe 1-2-3”, que incluye los siguientes tres pasos:

1. Capacitar a los implementadores y agentes de cambio *lean-agile*.
2. Capacitar a todos los ejecutivos, gerentes y líderes.
3. Capacitar equipos y lanzar trenes de lanzamiento ágiles.

3.6.3. Entrega disciplinada ágil

Disciplined Agile Delivery (**DAD**) es un proceso híbrido desarrollado originalmente por **IBM** que extiende el ciclo de vida de *scrum* con estrategias probadas de muchos métodos ágiles (Ambler y Lines, 2013). Es un enfoque ágil híbrido orientado al aprendizaje y a las personas para la entrega de soluciones de la tecnología de la información (**TI**). Tiene varios ciclos de vida, se basa en objetivos y es escalable.

DAD adopta los roles de *scrum* (*product owner* y equipos *scrum*). Además, incluye un rol denominado “líder del equipo” similar al *scrum master* de *scrum* y un propietario de arquitectura (proveniente del método *agile modeling*). Otros roles secundarios son especialista, *tester* independiente, experto de dominio, experto técnico e integrador, incluidos para abordar problemas de escala que se pueden organizar de forma temporal.

El marco **DAD** tiene cuatro ciclos de vida: ágil/básico, avanzado/*lean*, entrega continua y exploratorio (Ambler y Lines, 2013).

El ciclo de vida ágil/básico está orientado a equipos que trabajan de una manera ágil basada en *scrum*. Tiene tres fases, como se puede ver en la figura 3.28:

- *Inicio*: Consiste en comprender el alcance de lo que se desea desarrollar especialmente cuando un equipo está trabajando en el primer lanzamiento de nuevos productos. También en esta fase se forma el equipo inicial, se identifica la estrategia técnica, se desarrolla un plan inicial y se identifican riesgos.
- *Construcción*: Se produce una solución que proporcione valor de negocio y trate cualquier problema en el desarrollo temprano del producto mejorando la calidad y probando la arquitectura.
- *Transición*: Se organiza la solución en producción mediante una planificación y coordinación cuidadosas entre varios equipos.

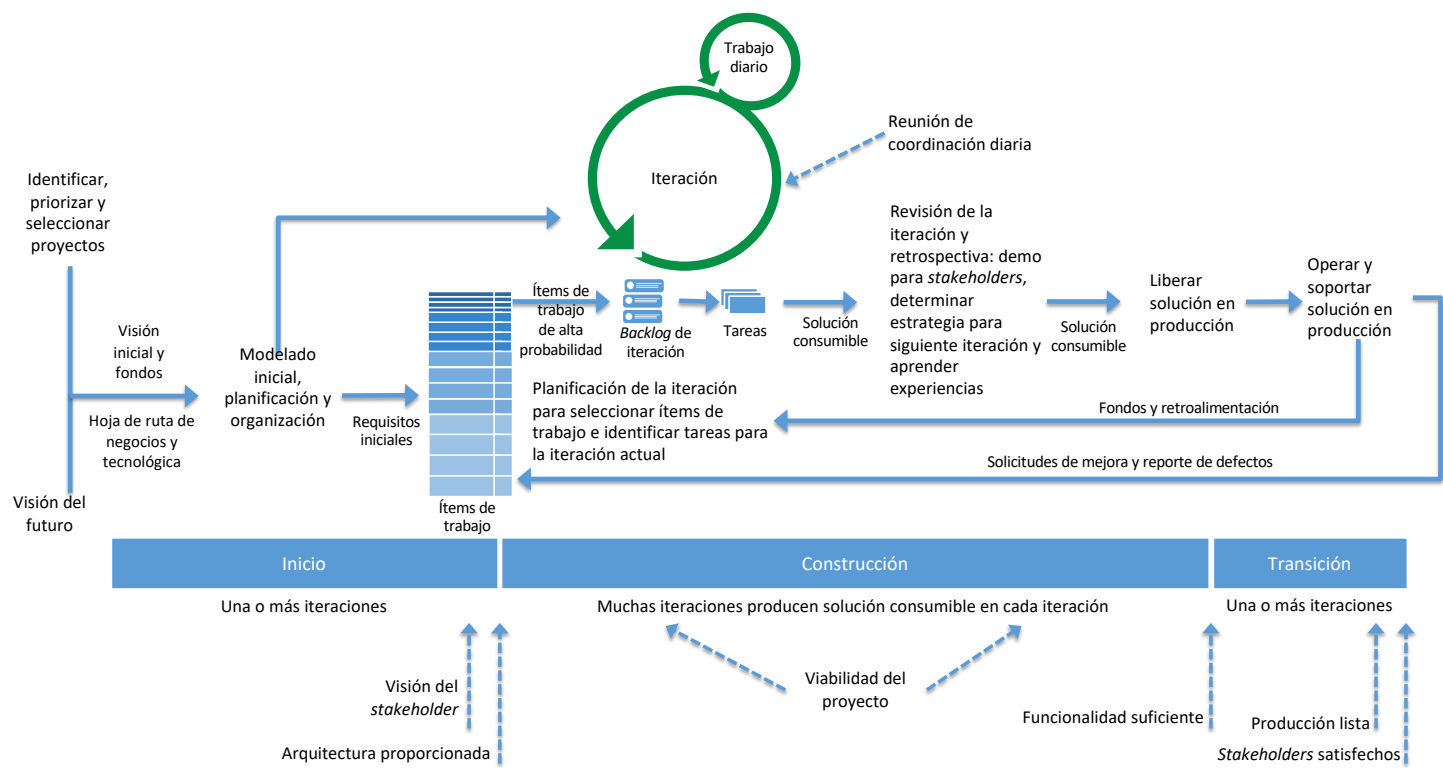


Figura 3.28. DAD

Fuente: elaboración propia con base en Ambler y Lines (2013).

El segundo ciclo de vida, avanzado/*lean*, está más cerca de *kanban* en su fase de construcción. Al igual que el primer ciclo de vida, la fase de inicio se utiliza para almacenar un grupo de elementos de trabajo que se organizan a lo largo de valores de negocio, fecha de entrega fija, acelerada o algunas otras categorías intangibles. Por otro lado, a diferencia de ágil/básico, la planificación, retrospectiva, demos y otras actividades se llevan a cabo según sea necesario, incluida la preparación para la implementación durante la fase de transición. Mientras tanto durante la fase de inicio se llevan a cabo algunos modelos arquitectónicos iniciales y una visión similar al primer ciclo de vida (Ambler y Lines, 2012).

En el ciclo de vida de entrega continua, la fase de inicio es explícita y tiene un periodo de transición breve. En este ciclo de vida, un producto se compra en producción o en el mercado de forma regular. Sin embargo, también son comunes las actividades semanales y mensuales (Ambler y Lines, 2012).

El ciclo de vida exploratorio tiene como objetivo incentivar a los equipos a encontrarse en situaciones de inicio o investigación en las que las partes interesadas tienen ideas claras para un nuevo producto mientras no comprenden las necesidades de sus usuarios. Este ciclo de vida consta de seis actividades: visualizar, crear prototipos, implementar, observar y medir, y basándose en la retroalimentación, cancelar o producir ideas (Ambler y Lines, 2012).

3.6.4. *Scrum* de gran escala

Large-Scale Scrum (LeSS) fue lanzado en 2008 basado en los trabajos de Larman y Vodde (2008). LeSS extiende *scrum* con reglas y pautas de escalado sin perder de vista los objetivos originales de *scrum* (figura 3.29). Tiene como objetivo facilitar la coordinación entre múltiples equipos *scrum* al tener un propietario de producto responsable de un *backlog* central y varios equipos. La coordinación entre equipos se realiza de manera similar a *scrum*, en que realizan una planificación y revisión de *sprints*. Para productos más pequeños, todos los miembros del producto se unen a la misma planificación y revisión del *sprint*. Para productos más grandes, se debe enviar un representante del equipo a las reuniones.

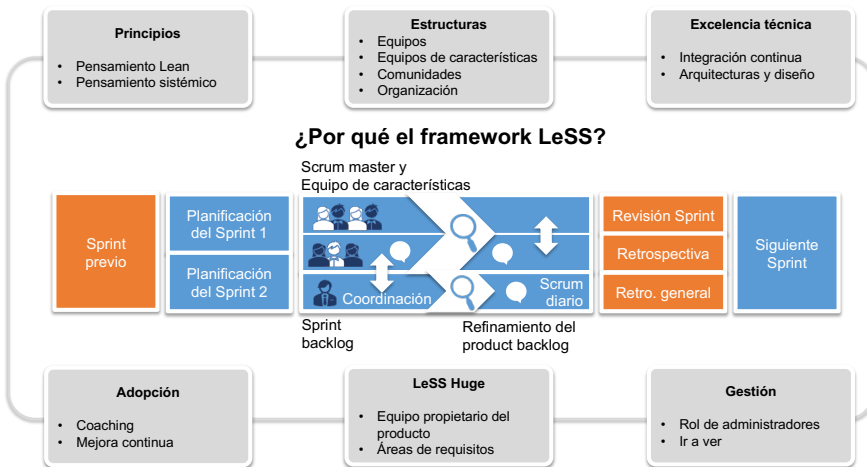


Figura 3.29. Framework LeSS

Fuente: elaboración propia con base en Scaled Agile Framework (2020).

LeSS comprende los siguientes cuatro componentes (Larman y Vodde, 2016):

- ▶ **Reglas:** El enfoque se centra en la estructura de los equipos, los roles dentro del equipo, la definición de los requisitos del producto y el proceso de desarrollo.
- ▶ **Principios:** Proporcionan respuestas sobre cómo aplicar LeSS en contextos empresariales específicos.
- ▶ **Guías:** Apoyan la adaptación de las reglas y un subconjunto de los experimentos al proporcionar consejos y mejores prácticas.
- ▶ **Experimentos:** Animan a los equipos a experimentar, fallar y aprender nuevos conceptos.

LeSS está conformado por dos marcos de referencia:

- ▶ **LeSS:** Para 2-8 equipos.
- ▶ **LeSS Huge:** Para más de ocho equipos.

Si las organizaciones o proyectos se escalan a ocho o más equipos, se debe utilizar LeSS Huge. LeSS Huge introduce elementos adicionales, como el concepto de áreas de requisitos. Estas se organizan en torno a requisitos centrados en el cliente, siguen el mismo ritmo del *sprint* y apuntan a la integración continua en todo el producto.

Surge un nuevo rol, un propietario del producto de área de requisitos que se centra en un área de requisitos y es responsable de una acumulación de productos de área. El propietario de producto global más todos los propietarios de productos de área forman el equipo propietario del producto. Un conjunto de tres a diez equipos (equipos de área) están dedicados a un propietario de producto de área, todos los cuales se especializan en un área de requisitos (figura 3.30). Cada equipo es multifuncional realizando desarrollo de funciones centradas en el cliente final.

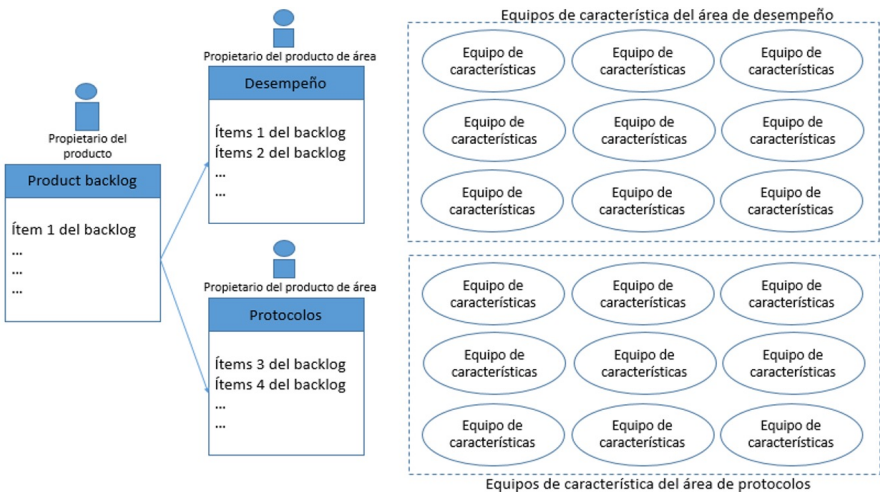


Figura 3.30. Propietario del producto por área de requisitos
Fuente: elaboración propia con base en Larman y Vodde (2013).

3.6.5. Comparativa de métodos de escalamiento ágil

En la tabla 3.6, se presenta una comparativa entre los cuatro métodos de escalado ágil descritos en este apartado. DAD, SFAFe, LeSS y scrum de scrums son los principales métodos de escalado ágil para abordar el desafío de aplicar las prácticas ágiles en grandes empresas. Por un lado, estos métodos ágiles parecen ser similares, ya que todos están orientados a resolver los problemas en grandes proyectos. Por otro lado, las discrepancias entre ellos toman la forma de tamaño del equipo, capacitación y certificación, métodos y prácticas adoptadas, prácticas técnicas requeridas y tipo de organización (Alqudah y Razali, 2016).

Tabla 3.6. Comparativa de métodos de escalamiento ágil

Criterio	SAFe	Scrum de scrums	LeSS	DAD
Tamaño del equipo	Grandes empresas con trenes de entrega	4 o 5 equipos <i>scrum</i> , cada uno de 3 a 9 personas	LeSS básico: hasta 70 personas o 10 equipos <i>scrum</i> Less Huge: más de 1000 personas	200 o más personas
Entrenamiento y certificación	Se requiere certificación	Certificaciones de <i>scrum master</i> escalado	7 compañías en 6 países ofrecen certificación	Talleres y libro sobre DAD
Métodos y prácticas	<i>Scrum</i> , <i>lean</i> , <i>kanban</i> , DevOps y XP	Prácticas <i>scrum</i>	<i>Scrum</i> con prácticas adicionales para grandes proyectos	<i>Kanban</i> , <i>scrum</i> , <i>agile modeling</i> , proceso unificado, XP
Prácticas técnicas requeridas	Media utilizando herramientas de gestión de portafolio	Baja para adaptadores de <i>scrum</i>	Media y baja para adaptadores de <i>scrum</i>	Alta requiriendo habilidades técnicas superiores
Tipo de organización	Empresas y nivel de portafolio	Grandes empresas tradicionales	Grandes empresas tradicionales	Organizaciones y empresas múltiples

Fuente: elaboración propia con base en Alqudah y Razali (2016).

► Desafíos

Los procesos ágiles, a pesar de tener muchos años en la industria, aún presentan desafíos por resolver. Tanto el desarrollo de software tradicional como el ágil tienen sus puntos fuertes y sus limitaciones según el contexto, y aplicarlos lejos de su “punto óptimo” siempre es un desafío. Entre algunos de los desafíos, se encuentra que, mientras muchas organizaciones están interesadas en adoptar métodos ágiles apropiados a ambientes de desarrollo grandes y complejos, hay pocas guías disponibles sobre cómo hacerlo (Sidky, 2007). Las organizaciones necesitan procedimientos y métodos para apoyar una selección y despliegue sistemáticos de nuevas prácticas ágiles y adaptarlas a contextos organizacionales específicos (Pikkariainen *et al.*, 2005).

Meyer (2014) resalta que los procesos ágiles desestiman las actividades iniciales de requisitos y diseño, la utilización de historias de usuario como base para los requisitos no es lo suficientemente rigurosa, ignoran las dependencias entre características, se resisten a aplicar herramientas de seguimiento de dependencias, rechazan las tareas de gestión tradicionales y la documentación, y a su vez la idea de incorporar el cliente al equipo no funciona bien en la práctica.

Los procesos ágiles poseen varios desafíos específicos para las prácticas convencionales de aseguramiento de la calidad (Marrington *et al.*, 2005):

- *Los procesos ágiles generan pocos artefactos del proceso:* Los procesos tradicionales ofrecen una trayectoria de registro y auditoría más elaborada que el enfoque ágil, y así proporcionan una base para la revisión de prácticas y artefactos. La doctrina ágil, en su forma extrema, afirma que toda la documentación está incorporada en el código, asegurando que la extracción del material de revisión sea más difícil.
- *Los procesos ágiles pretenden estar más orientados a las personas que a los procesos:* Existe una percepción de que el proceso en general y las actividades de aseguramiento de la calidad orientadas a la gestión en particular son mucho menos críticas que la visión técnica del desarrollador.
- *Las metodologías ágiles son adaptables y, por tanto, permiten la toma de decisiones críticas a lo largo del proceso:* Esta preocupación refleja la naturaleza iterativa del enfoque ágil y la adaptación del proyecto a los cambios en los requisitos.

Otros desafíos los plantea Cohn y Ford (2003) relacionados con las actividades realizadas por los desarrolladores, la administración superior y los recursos humanos:

- *Desarrolladores:* Responden a la introducción de procesos ágiles con una combinación de escepticismo, entusiasmo y optimismo cauteloso. Sin embargo, otros desarrolladores se resisten al cambio o se lanzan exageradamente al proceso ágil sin suficiente previsión. Pueden presentar resistencia al cambio en la transición de un proceso tradicional a ágil, ven el proceso ágil como una microgestión y se requiere personal con mucho talento.
- *Administración superior:* Muchos administradores de alto nivel son reacios a renunciar a los diagramas clásicos de planificación y otros artefactos similares. Se preocupan por que un proceso ágil

no pueda gestionar compromisos y que no implique garantías sobre la fecha de entrega, costo y funcionalidad. Además, piensan que no hay un buen seguimiento del progreso de los proyectos. Por último, la alta gerencia comúnmente teme que un proyecto se prolongue para siempre.

- *Recursos humanos*: El personal se puede quejar de que no les gusta la programación en parejas, la forma en que un proyecto es gestionado o que están tan familiarizados con los procesos tradicionales y que cualquier otro proceso les parece caótico.

Boehm y Turner (2005) enseñan tres áreas que presentan desafíos críticos a los administradores de software de grandes organizaciones para aplicar enfoques ágiles en sus proyectos:

- *Conflictos del proceso de desarrollo*: ¿Cómo mezclar procesos ágiles con procesos industriales estandarizados sin eliminar la agilidad o sin menoscabo de años gastados en la definición y el refinamiento de sus activos de procesos? En esta área, se han manifestado dificultades de su aplicación en la gestión de la variabilidad en subsistemas y equipos, en el trabajo con diferentes ciclos de vida, en su aplicación en sistemas heredados y en el desarrollo de requisitos.
- *Conflictos del proceso de negocio*: En la forma en que cada día el negocio es realizado. Aspectos como la estimación, la carga de recursos y los cálculos laxos pueden variar significativamente. Pueden surgir problemas de recursos humanos (como registro de tiempos, orientación al equipo frente a recompensas individuales y habilidades requeridas), medición del progreso (contratos, hitos y otras técnicas) que puede ser inadecuado y adaptación a evaluación de estándares de procesos.
- *Conflictos de personal*: Las cuestiones de personal son más críticas en la mejora de la gestión de personal. Se presentan problemas en la migración de actitudes de gestión, en cuestiones logísticas, en la gestión de proyectos piloto exitosos y en la gestión del cambio organizacional.

Boehm y Turner (2005) no creen que estos conflictos sean insuperables y confían en que las organizaciones puedan vencerlos con diligencia, paciencia y trabajo.

► Ejemplos

Para un proyecto de software de una librería *online*, se ha aplicado el proceso ágil *scrum*. Se han definido los elementos del *product backlog* que se presentan en la tabla 3.8. Esta tabla presenta tantos elementos que son requisitos funcionales (en el formato recomendado) como no funcionales, así como resolución de defectos y búsqueda de conocimiento.

Tabla 3.8. Elementos del *product backlog*

Prioridad	Elemento	Estimación inicial
1	Como comprador, quiero colocar un libro en un carrito de compras.	5
2	Como comprador, quiero eliminar un libro en un carrito de compras.	2
3	Mejorar el rendimiento del procesamiento de transacciones.	13
4	Investigar soluciones para acelerar la validación de tarjetas de crédito.	20
5	Actualizar todos los servidores a Apache 2.2.3.	13
6	Diagnosticar y corregir los errores de <i>script</i> de procesamiento de pedidos (ID Bugzilla 14823).	3
7	Como comprador, quiero crear y guardar una lista de deseos.	40
8	Como comprador, quiero agregar o eliminar artículos en mi lista de deseos.	20

Fuente: elaboración propia con base en Deemer *et al.* (2012).

A partir del *product backlog*, se definió el correspondiente *sprint backlog* para implementar las historias de usuario (tabla 3.9). Solo se presentan las tareas definidas para completar dos historias de usuario.

A medida que va avanzando la ejecución del *sprint* se va actualizando el *sprint backlog* con datos reales de la realización de cada tarea, como se observa en la tabla 3.10.

Tabla 3.9. *Sprint backlog*

Elemento del <i>product backlog</i>	ID tarea	Tarea del <i>sprint</i>	Estimación inicial de esfuerzo
Como comprador quiero colocar un libro en un carrito de compras	T1	Modificar base de datos	5
	T2	Crear página web (interfaz gráfica de usuario)	8
	T3	Crear página web (JavaScript)	13
	T4	Crear pruebas de aceptación automatizadas	13
	T5	Actualizar página web de ayuda al comprador	3
Mejorar el rendimiento del procesamiento de transacciones	...		
	T6	Mezclar código y completar pruebas en capas	5
	T7	Completar orden de máquina para herramienta PRANK	8
	...		

Fuente: elaboración propia con base en Deemer *et al.* (2012).

Tabla 3.10. Actualización del *sprint backlog*

ID tarea	Tarea del <i>sprint</i>	Estimación inicial de esfuerzo	Responsable	Esfuerzo restante				
				1	2	3	4	5
T1	Modificar base de datos	5	Persona 1	4	3	0	0	0
T2	Crear página web (GUI)	8	Persona 2	3	3	2	0	0
T3	Crear página web (JavaScript)	13	Persona 3	2	2	2	1	0
T4	Crear pruebas de aceptación	13	Persona 4	5	5	5	5	0
T5	Actualizar página web de ayuda al comprador	3	Persona 5	3	3	3	3	0
...								
T6	Mezclar código y completar pruebas en capas	5	Persona 6	5	5	5	5	5
T7	Completar orden de máquina para PRANK	8	Persona 7	3	8	8	8	8

Fuente: elaboración propia con base en Deemer *et al.* (2012).

A medida que avanza el proyecto, se va actualizando el *burndown chart*, como se observa en la figura 3.31.

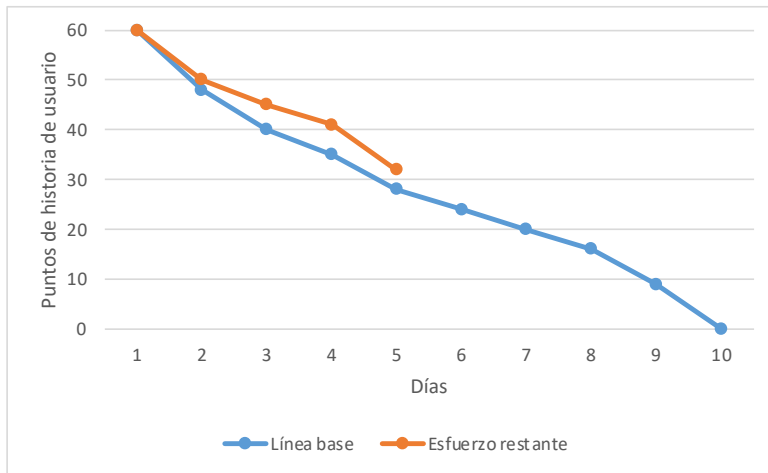


Figura 3.31. *Burndown chart*

Fuente: elaboración propia con base en Deemer *et al.* (2012).

► Resumen

Los procesos ágiles son procesos ligeros (con pocos artefactos, roles y eventos) que se basan en el desarrollo de ciclos rápidos, entrega frecuente, marcos de tiempos cortos y bien definidos, son concurrentes y asincrónicos, con un enfoque iterativo e incremental y centrados en las personas con la necesidad de una comunicación completa entre el equipo de desarrollo y el cliente, además de una respuesta efectiva al cambio.

Los equipos ágiles se caracterizan por ser autogestionados (no tienen líderes específicos y ellos mismos organizan su trabajo) y multifuncionales (tienen la experiencia necesaria para realizar el trabajo sin necesidad de especificar roles concretos y especializados).

Aunque su origen data de los trabajos iniciales del ciclo **PDCA** y manufactura *lean*, los procesos ágiles de manera formal se consolidaron en la redacción del manifiesto ágil de 2001 y hoy son ampliamente utilizados en la industria. Los procesos ágiles más conocidos y utilizados son **XP** y *scrum*.

XP se basa en los valores del respeto, la comunicación, la simplicidad, la retroalimentación y el coraje. Incluye la utilización de doce prácticas categorizadas en prácticas técnicas, planificación, coordinación y comunicación. A su vez, el proceso está conformado por las fases de exploración, planificación, iteraciones, producción, mantenimiento y cierre.

Scrum es un marco de trabajo compuesto por roles, artefactos y actividades para el desarrollo de software. El núcleo de *scrum* es el *sprint*, el cual es una iteración de duración corta en la que se desarrollan las principales funcionalidades del sistema obteniéndose un incremento de producto potencialmente entregable. El *product owner* es el encargado de definir las funcionalidades con mayor valor de negocio y que deben ser producidas por el equipo de desarrollo, el cual es orientado por un *scrum master* en la correcta aplicación de las prácticas *scrum*. Se recomienda la aplicación de *scrum* en proyectos con fecha y presupuesto fijo, pero con alcance flexible.

Otros procesos ágiles son **LSD**, los métodos *crystal*, **DSDM**, **FDD**, **ASD** y **TDD**.

La selección del proceso ágil adecuado dependerá de muchos criterios y características del proyecto a desarrollar.

Aunque los procesos ágiles han sido ampliamente acogidos por la comunidad desarrolladora de software, aún tienen muchos desafíos por resolver, entre ellos, una adopción apropiada en ambientes grandes y complejos, incorporar documentaciones formales y necesarias que orienten y guarden registro del proceso, hacer énfasis en arquitectura de software y realizar estimaciones fiables y rigurosas.

Los equipos de desarrollo de software en empresas a gran escala se enfrentan actualmente a los desafíos de entregar software de alta calidad con arduas limitaciones de tiempo. El desafío de adoptar métodos ágiles para desarrollar software aumenta aún más con la necesidad de escalar estas prácticas a entornos más grandes. Entre los métodos más utilizados para escalar las prácticas ágiles se encuentran **SAFe**, *scrum of scrums*, **LeSS** y **DAD**.

► Ejercicios

1. Repasar los doce principios del manifiesto ágil. ¿Cuáles cree que son los dos puntos más difíciles de implementar en proyectos de desarrollo de software? ¿Por qué?

2. Realizar una tabla en la que verifique el cumplimiento de los doce principios del manifiesto ágil en los diferentes procesos ágiles presentados en este capítulo.
3. ¿Cuál es la estructura que se recomienda para un definir una buena historia de usuario?
4. Determinar cinco historias de usuario para los siguientes productos de software:
 - WhatsApp
 - Facebook
 - Gmail
 - Uber
 - Netflix
5. Investigar en la web diversas herramientas que permitan elaborar en línea un tablero *kanban*.
6. ¿En qué tipos de riesgos incurre un equipo de desarrollo cuando elige un modelo de proceso ágil en lugar de un modelo de proceso prescriptivo?
7. ¿Es iterativo todo proceso ágil? Justificar su respuesta.
8. Determinar dos diferencias y dos similitudes entre *scrum* y el modelo de desarrollo incremental.
9. Responder las siguientes preguntas sobre el proceso *scrum*:
 - ¿En qué situaciones un *scrum master* se comunicará con el propietario de producto en un proyecto *scrum*?
 - ¿Debería el equipo de *scrum* involucrarse en el proceso de descubrimiento de productos? Si la respuesta es afirmativa, ¿cómo?
 - ¿Debería un propietario de producto asignar historias de usuario o tareas a miembros individuales de un equipo *scrum*?
 - ¿Cómo aplicaría el concepto de reuniones diarias de *scrum* en equipos distribuidos?
10. El 15.º Reporte del Estado Ágil identifica al *framework* SAFe como el marco de escalado ágil de mayor aceptación. ¿Por qué cree que SAFe tiene la mayor aceptación en el mercado?

► Referencias

- Abrahamsson, P., Salo, O., Ronkainen, J. y Warsta, J. (2002). *Agile software development methods: Review and analysis*. **VTT Electronics**. <https://doi.org/10.48550/arXiv.1709.08439>
- Abrahamsson, P., Warsta, J., Siponen, M. T. y Ronkainen, J. (2003). New directions on agile methods: A comparative analysis. En *25th International Conference on Software Engineering, 2003. Proceedings* (pp. 244-254). **IEEE**. <https://doi.org/10.1109/ICSE.2003.1201204>
- Abrantes, J. F. y Travassos, G. H. (2011). Common agile practices in software processes. En *2011 International Symposium on Empirical Software Engineering and Measurement* (pp. 355-358). **IEEE**. <https://doi.org/10.1109/ESEM.2011.47>
- Adelyar, S. H. y Norta, A. (2016). Towards a secure agile software development process. En *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)* (pp. 101-106). **IEEE**. <https://doi.org/10.1109/QUATIC.2016.028>
- Agile Manifesto. (2001). *Manifesto for Agile Software Development*. <http://agilemanifesto.org/>
- Alqudah, M. y Razali, R. (2016). A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology*, 6(6), 828-837. <http://dx.doi.org/10.18517/ijaseit.6.6.1374>
- Ambler, S. (2002). *Agile modeling: Effective practices for eXtreme Programming and the unified process*. John Wiley & Sons.
- Ambler, S. W. (2009). *The agile scaling model (ASM): Adapting agile methods for complex environments*. <https://scrummasters.com/wp-content/uploads/2022/02/White-Paper-Adapting-Agile.pdf>
- Ambler, S. W. y Lines, M. (2012). *Disciplined Agile Delivery: A practitioner's guide to agile software delivery in the enterprise*. **IBM Press**.
- Ambler, S. W. y Lines, M. (2013). Going beyond scrum: Disciplined Agile Delivery, disciplined agile consortium. *White Paper Series*. <https://www.classes.cs.uchicago.edu/archive/2016/fall/51205-1/required.reading/BeyondScrum.pdf>
- Ambler, S. W. y Lines, M. (2016). The disciplined agile process decision framework. En D. Winkler, S. Biffl y J. Bergsmann (eds.), *Software Quality: The Future of Systems- and Software Development*. **SWQD 2016. Lecture Notes in Business Information Processing** (pp. 3-14). Springer. https://doi.org/10.1007/978-3-319-27033-3_1
- Anderson, L., Alleman, G. B., Beck, K., Blotner, J., Cunningham, W., Poppendieck, M. y Wirfs-Brock, R. (2003). Agile management-an oxymoron? Who needs managers anyway? En *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (pp. 275-277). <https://doi.org/10.1145/949344.949410>
- Aoyama, M. (1998). Agile software process and its experience. En *Proceedings of the 20th International Conference on Software Engineering* (pp. 3-12). **IEEE**. <https://doi.org/10.1109/ICSE.1998.671097>
- Baca, D. y Carlsson, B. (2011). Agile development with security engineering activities. En *Proceedings of the 2011 International Conference on Software and Systems Process* (pp. 149-158). <https://doi.org/10.1145/1987875.1987900>

- Barnett, L. y Schwaber, C. (2004). *Adopting agile development processes: Improve time-to-benefits for software projects forrester research*.
- Bartsch, S. (2011). Practitioners' perspectives on security in agile development. En *2011 Sixth International Conference on Availability, Reliability and Security* (pp. 479-484). **IEEE**. <https://doi.org/10.1109/ARES.2011.82>
- Beck, K. (2000). *Extreme programming explained: Embrace change*. Addison-Wesley.
- Beck, K. (2002). *Test driven development: By example*. Addison-Wesley.
- Beck, K. y Fowler, M. (2001). *Planning eXtreme Programming*. Addison-Wesley.
- Ben Othmane, L., Angin, P., Weffers, H. y Bhargava, B. (2014). Extending the agile development process to develop acceptably secure software. **IEEE Transactions on Dependable and Secure Computing**, 11(6), 497-509. <https://doi.org/10.1109/TDSC.2014.2298011>
- Bessam, A., Kimour, M. T. y Melit, A. (2009). Separating users' views in a development process for agile methods. En *2009 Fourth International Conference on Dependability of Computer Systems* (pp. 61-68). **IEEE**. <https://doi.org/10.1109/DepCoS-RELCOMEX.2009.16>
- Boehm, B. y Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. **IEEE Software**, 22(5), 30-39. <https://doi.org/10.1109/MS.2005.129>
- Buglione, L. y Abran, A. (2013). Improving the user story agile technique using the invest criteria. En *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement* (pp. 49-53). **IEEE**. <https://doi.org/10.1109/IWSM-Mensura.2013.18>
- Canós, J., Letelier, P. y Penadés, M. (2003). *Metodologías ágiles en el desarrollo de software*. https://www.academia.edu/download/34546906/XP_Agil.pdf
- Chowdhury, A. F. y Huda, M. N. (2011). Comparison between Adaptive Software Development and Feature-Driven Development. En *Proceedings of 2011 International Conference on Computer Science and Network Technology* (vol. 1, pp. 363-367). **IEEE**. <https://doi.org/10.1109/ICCSNT.2011.6181977>
- Coad, P., Lefebvre, E. y Luca, J. D. (1999). *Feature-driven development. En Java modeling in color with UML: Enterprise components and process*. Prentice Hall **PTR**.
- Cockburn, A. (2004). *Crystal clear: A human-powered methodology for small teams*. Addison-Wesley.
- Cohen D., Lindvall, M. y Costa P. (2004). An introduction to agile methods. *Advances in Computers*, 62(3), 1-66. [https://doi.org/10.1016/S0065-2458\(03\)62001-2](https://doi.org/10.1016/S0065-2458(03)62001-2)
- Cohn, M. y Ford, D. (2003). Introducing an agile process to an organization [software development]. *Computer*, 36(6), 74-78. <https://doi.org/10.1109/MC.2003.1204378>
- Coram, M. y Bohner, S. (2005). The impact of agile methods on software project management. En *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)* (pp. 363-370). **IEEE**. <https://doi.org/10.1109/ECBS.2005.68>
- Cugola, G. y Ghezzi, C. (1998). Software processes: A retrospective and a path to the future. *Software Process: Improvement and Practice*, 4(3), 101-123. [https://doi.org/10.1002/\(SICI\)1099-1670\(199809\)4:3%3C101::AID-SPIP103%3E3.0.CO;2-K](https://doi.org/10.1002/(SICI)1099-1670(199809)4:3%3C101::AID-SPIP103%3E3.0.CO;2-K)

- Deemer, P., Benefield, G., Larman, C. y Vodde, B. (2012). *A lightweight guide to the theory and practice of scrum. Version 2.0*. InfoQ Enterprise Software Development Series. <https://www.scruminc.com/wp-content/uploads/2014/05/scrumprimer20.pdf>
- Despa, M. L. (2014). Comparative study on software development methodologies. *Database Systems Journal*, 5(3), 37-56. <https://dbjournal.ro/archive/17/17.pdf#page=38>
- Digital.ai. (2024, 4 de marzo). *The 17th State of Agile Report*. <https://digital.ai/resource-center/analyst-reports/state-of-agile-report/>
- DSDM Consortium. (2008). *DSDM Atern Handbook V2/2*. Whitehorse Press.
- Fitzgerald, B., Hartnett, G. y Conboy, K. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems*, 15(2), 200-213. <https://doi.org/10.1057/palgrave.ejis.3000605>
- Fowler, M. (2005, 13 de diciembre). *The new methodology*. <https://www.martinfowler.com/articles/newMethodology.html>
- Fraser, S., Reinitz, R., Eckstein, J., Kerievsky, J., Mee, R. y Poppendieck, M. (2003). Xtreme programming and agile coaching. En *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (pp. 265-267). <https://doi.org/10.1145/949344.949406>
- Ghani, I. y Yasin, I. (2013). Software security engineering in eXtreme Programming methodology: A systematic literature review. *Science International*, 25(2), 215-221. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fb3ac381f336911fe-46c638abdde53376d74a5e5>
- Hewitt, B. y Walz, D. (2005). Using shared leadership to foster knowledge sharing in information systems development projects. En *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (pp. 256a-256a). **IEEE**. <https://doi.org/10.1109/HICSS.2005.666>
- Highsmith, J. A. (2002). *Agile software development ecosystems*. Addison-Wesley.
- Highsmith, J. A. (2004). *Agile project management: Creating innovative products*. Addison-Wesley.
- Highsmith, J. A. (2013). *Adaptive software development: A collaborative approach to managing complex systems*. Addison-Wesley.
- Highsmith, J. y Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*, 34(9), 120-127. <https://doi.org/10.1109/2.947100>
- Ionel, N. (2008). Critical analysis of the scrum project management methodology. *Annals of the University of Oradea, Economic Science Series*, 17(4), 435-441. <https://anale.steconomiceuoradea.ro/volume/2008/v4-management-marketing/077.pdf>
- Kanwal, F., Junaid, K. y Fahiem, M. A. (2010). A hybrid software architecture evaluation method for **FDD**: An agile process model. En *2010 International Conference on Computational Intelligence And Software Engineering* (pp. 1-5). **IEEE**. <https://doi.org/10.1109/CISE.2010.5676863>
- Khatri, S. K., Bahri, K. y Johri, P. (2014). Best practices for managing risk in adaptive agile process. En *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization* (pp. 1-5). **IEEE**. <https://doi.org/10.1109/ICRITO.2014.7014759>

- Kirkman, B. L. y Rosen, B. (1999). Beyond self-management: Antecedents and consequences of team empowerment. *Academy of Management Journal*, 42(1), 58-74. <https://doi.org/10.5465/256874>
- Larman, C. (2004). *Agile and iterative development: A manager's guide*. Addison-Wesley.
- Larman, C. y Vodde, B. (2008). *Scaling lean & agile development: Thinking and organizational tools for large-scale scrum*. Addison-Wesley.
- Larman, C. y Vodde, B. (2013). Scaling agile development. *CrossTalk*, 9, 8-12. [https://miroslawdabrowski.com/downloads/Scrum/Large%20Scale%20Scrum%20\(LESS\).pdf](https://miroslawdabrowski.com/downloads/Scrum/Large%20Scale%20Scrum%20(LESS).pdf)
- Larman, C. y Vodde, B. (2016). *Large-scale scrum: More with LeSS*. Addison-Wesley.
- Leffingwell, D. (2011). *Scaling Agile Framework (SAFe)*. <http://www.scaledagileframework.com/>
- Lindstrom, L. y Jeffries, R. (2005). Extreme Programming and agile software development methodologies. *Information Systems Management*, 21(3), 41-52.
- Lui, T. W. y Piccoli, G. (2006). Degrees of agility: Implications for information systems design and firm strategy. En K. Desouza (ed.), *Agile information systems* (pp. 122-133). Routledge. <https://doi.org/10.4324/9780080463681>
- Mahmud, D. M. y Abdullah, N. A. S. (2015). Reviews on agile methods in mobile application development process. En *2015 9th Malaysian Software Engineering Conference (MySEC)* (pp. 161-165). IEEE. <https://doi.org/10.1109/MySEC.2015.7475214>
- Marrington, A., Hogan, J. M. y Thomas, R. (2005). Quality assurance in a student-based agile software engineering process. En *2005 Australian Software Engineering Conference* (pp. 324-331). IEEE. <https://doi.org/10.1109/ASWEC.2005.38>
- Maximini, D. (2015). *The scrum culture: Introducing agile methods in organizations. management for professionals*. Springer. <https://doi.org/10.1007/978-3-319-73842-0>
- Meng, X. X., Wang, Y. S., Shi, L. y Wang, F. J. (2007). A process pattern language for agile methods. En *14th Asia-Pacific Software Engineering Conference (APSEC'07)* (pp. 374-381). IEEE. <https://doi.org/10.1109/ASPEC.2007.72>
- Meyer, B. (2014). *Agile! The good, the hype and the ugly*. Springer.
- Millett, S., Blankenship, J. y Bussa, M. (2011). *Pro agile: .NET development with scrum*. Apress.
- Misra, S. C., Kumar, V. y Kumar, U. (2009). Identifying some important success factors in adopting agile software development practices. *Journal of Systems and Software*, 82(11), 1869-1890. <https://doi.org/10.1016/j.jss.2009.05.052>
- Morgan, G. (2006). *Images of organizations*. Sage.
- Müller, M. M. y Höfer, A. (2007). The effect of experience on the test-driven development process. *Empirical Software Engineering*, 12, 593-615. <https://doi.org/10.1007/s10664-007-9048-2>
- Mundra, A., Misra, S. y Dhawale, C. A. (2013). Practical scrum-scrum team: Way to produce successful and quality software. En *2013 13th International Conference on Computational Science and Its Applications* (pp. 119-123). IEEE. <https://doi.org/10.1109/ICCSA.2013.25>
- Nerur, S., Mahapatra, R. y Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5), 72-78. <https://doi.org/10.1145/1060710.1060712>

- Newkirk, J. (2002). Introduction to agile processes and eXtreme Programming. En *Proceedings of the 24th International Conference on Software Engineering* (pp. 695-696). <https://doi.org/10.1145/581339.581450>
- Paasivaara, M., Lassenius, C. y Heikkilä, V. T. (2012). Inter-team coordination in large-scale globally distributed scrum: Do scrum-of-scrums really work? En *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 235-238). <https://doi.org/10.1145/2372251.2372294>
- Palmer, S. R. y Felsing, M. (2002). *A practical guide to Feature-Driven Development*. Prentice Hall.
- Pearce, C. L. (2004). The future of leadership: Combining vertical and shared leadership to transform knowledge work. *Academy of Management Executive*, 18(1), 47-57. <https://doi.org/10.5465/ame.2004.12690298>
- Pikkarainen, M., Salo, O. y Still, J. (2005). Deploying agile practices in organizations: A case study. En I. Richardson, P. Abrahamsson y R. Messnarz (eds.), *Software Process Improvement. EuroSPI 2005. Lecture Notes in Computer Science* (pp. 16-27). Springer. https://doi.org/10.1007/11586012_3
- Pohl, C. y Hof, H. J. (2015). Secure scrum: Development of secure software with scrum. En *Proceedings of the Ninth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2015)*. IARIA XPS Press. <https://doi.org/10.48550/arXiv.1507.02992>
- Poppendieck, M. y Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Addison-Wesley.
- Poppendieck, M. y Poppendieck, T. (2006). *Implementing Lean Software Development: From concept to cash*. Addison-Wesley.
- Qumer, A. y Henderson-Sellers, B. (2008). An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology*, 50(4), 280-295. <https://doi.org/10.1016/j.infsof.2007.02.002>
- Reifer, D. (2002). How good are agile methods? *IEEE Software*, 19(4), 16-18. <https://doi.org/10.1109/MS.2002.1020280>
- Rick, U., Vossen, R., Richert, A. y Henning, K. (2010). Designing agile processes in information management. En *2010 2nd IEEE International Conference on Information Management and Engineering* (pp. 156-160). IEEE. <https://doi.org/10.1109/ICIME.2010.5477776>
- Rieckmann, H. (1992). Dynaxibility - oder wie "systemisches". Management in der Praxis funktionieren kann. En K. Henning y B. Harendt (eds.), *Methodik und Praxis der Komplexitätsbewältigung* (pp. 17-39). Duncker & Humblot.
- Riehle, D. (2000). A comparison of the value systems of Adaptive Software Development and eXtreme Programming: How methodologies may learn from each other. En *Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP 2000)* (pp. 35-50). <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=807bd8be840ede-d41828ad9052e0d4d14b31865c>
- Rubin, K. S. (2012). *Essential scrum: A practical guide to the most popular agile process*. Addison-Wesley.
- Rossberg, J. (2014). *Beginning application lifecycle management*. Apress.

- Scale Agile Framework. (2020). *SAFe 6.0*. <https://www.scaledagileframework.com/>
- Schuh, P. (2004). *Integrating agile development in the real world*. Charles River Media.
- Schwaber, K. y Beedle, M. (2001). *Agile software development with scrum*. Prentice Hall **PTR**.
- Sidky, A. S. (2007). *A structured approach to adopting agile practices: The agile adoption framework* (tesis de doctorado, Virginia Tech). <https://vtechworks.lib.vt.edu/server/api/core/bitstreams/4ff25112-51c4-4ce7-86f3-ee3e0c84100a/content>
- Singhal, A. (2012). Integration analysis of security activities from the perspective of agility. En *2012 Agile India* (pp. 40-47). **IEEE**. <https://doi.org/10.1109/AgileIndia.2012.9>
- Sneed, H. M. (2014). Dealing with technical debt in agile development projects. En D. Winkler, S. Biffl y J. Bergsmann (eds.), *Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering*. **SWQD 2014. Lecture Notes in Business Information Processing** (pp. 48-62). Springer. https://doi.org/10.1007/978-3-319-03602-1_4
- Stapleton, J. (1997). *DSDM, dynamic systems development method: The method in practice*. Cambridge University Press.
- Sutherland, J. y Schwaber, K. (2020). *The 2020 Scrum Guide™*. <https://scrumguides.org/scrum-guide.html>
- Tahir, F. y Manarvi, I. A. (2013). Agile process model and practices in distributed environment. En J. Stjepandić, G. Rock y C. Bil (eds.), *Concurrent engineering approaches for sustainable product development in a multi-disciplinary environment* (pp. 1169-1180). Springer. https://doi.org/10.1007/978-1-4471-4426-7_98
- Tiltmann, T. (2007). *Agile Entwicklung von CSCW-Anwendungen für regionale Bildungswerke*. Mainz.
- Trist, E. (1981). The evolution of socio-technical systems: A conceptual framework and an action research program. *Occasional Paper*, 2. <https://www.lmmiller.com/blog/wp-content/uploads/2013/06/The-Evolution-of-Socio-Technical-Systems-Trist.pdf>
- Vaidya, A. (2014). Does dad know best, is it better to do less or just be safe? Adapting scaling agile practices into the Enterprise. En *32nd Annual Pacific Northwest Software Quality Conference - PNSQC 2014* (pp. 1-18). <https://pnsqc.org/archives/dad-know-best-better-less-just-enough-safe-adapting-agile-scaling-practices-enterprise/>
- Voigt, B. J., Glinz, M. y Seybold, D. I. C. (2004). *Dynamic system development method*. University of Zurich. https://files.ifi.uzh.ch/rerg/amadeus/teaching/seminars/seminar_ws0304/14_Voigt_DSMD_Ausarbeitung.pdf
- Walton, M. (1999). Strategies for lean product development: A compilation of lean aerospace initiative research. *Research Paper*, 2.
- Womack, J. P. y Jones, D. T. (2003). *Lean thinking: Banish waste and create wealth in your corporation*. Free Press.
- Womack, J. P., Jones, D. T. y Roos, D. (2007). *The machine that changed the world: The story of lean production*. Simon and Schuster.

Estándares y procesos industriales de software

► Objetivos

Los objetivos de este capítulo están enfocados en describir un conjunto de estándares internacionales del proceso de desarrollo de software y de diversos procesos de software generados, conocidos y ampliamente aplicados en la industria de software a nivel mundial. Al estudiarlo, el lector:

- Conocerá una guía sobre cómo mejorar el proceso de software a través de la utilización de estándares y procesos industriales de desarrollo de software.
- Comprenderá qué son y qué proporcionan los estándares del proceso de software.
- Examinará el proceso de estandarización y cuestiones que surgen del control y evolución de estándares.

Los estándares son reconocidos en la industria del software como una forma de transferir buenas prácticas al uso industrial. Las empresas desarrolladoras al utilizar estándares del proceso de software aseguran un nivel de calidad asociado también al producto desarrollado. Además, los estándares serán la base para que las empresas y sus productos sean certificados. Finalmente, si dos organizaciones que están certificadas colaboran en el desarrollo de un mismo producto, la cooperación entre ambas será considerablemente simplificada (Derniame *et al.*, 1999).

Las iniciativas dirigidas a la estandarización de organizaciones desarrolladoras de software apuntan en tres direcciones (Freericks, 2001):

- *Definición de estándares del proceso*: Proporcionan un manual efectivo y bien definido de calidad del proceso. Brindan definiciones y restricciones para las actividades, los roles y documentos a ejecutarse en la generación del producto de software deseado. Por ejemplo, ISO 9000:2015, ISO/IEC 12207:2017 e IEEE 1074:2006.
- *Definición de métodos de evaluación del proceso*: Proporcionan guías para evaluar la madurez del proceso que es ejecutado en una organización. Se identifican ciertos niveles de madurez y un método de evaluación localiza a la organización en un nivel determinado. Por ejemplo, Capability Maturity Model Integration (CMMI), Bootstrap e ISO/IEC 15504.
- *Definición de métodos de mejora del proceso*: Proporcionan metodologías y guías para mejorar un proceso existente. Durante su ejecución, un conjunto de métricas es recolectado, el cual es analizado y empaquetado para usos futuros. Por ejemplo, Quality Improvement Plan (QIP) o gestión de la calidad total.

Este capítulo se centrará en la primera dirección: la definición de estándares del proceso de software. Las otras dos direcciones serán tratadas en capítulos posteriores.

► 4.1. Conceptos sobre estándares

Un estándar se puede definir como un conjunto de criterios aprobados, documentados y disponibles para ser utilizados como reglas, guías o definiciones de características que permiten determinar la adecuación de una acción (estándar de proceso) o de un objeto (estándar de producto) a su propósito (David, 1995; De Vries, 2013).

Los estándares, en general, tienen siete atributos esenciales que ayudan en el proceso de ingeniería (Moore, 2006):

1. Representan la experiencia recolectada por otros, quienes han recorrido el mismo camino.
2. Describen en detalle lo que significa ejecutar una cierta actividad.
3. Pueden ser enlazados o referenciados en contratos.
4. Ayudan a asegurar que las dos partes definan una actividad de ingeniería de esta forma.

5. Incrementan la disciplina profesional.
6. Protegen el negocio y al comprador.
7. Mejoran el producto.

Los estándares proporcionan reglas, guías y heurísticas, los cuales, si se siguen, transfieren un aseguramiento de buenas prácticas. Sirven como una base para determinar niveles mínimos aceptables y consistentes de calidad, desempeño, seguridad y confiabilidad (Land *et al.*, 2012).

Los estándares son útiles por múltiples razones, entre las cuales están las enunciadas por Sommerville (1996):

- ▶ Agrupan lo mejor y más apropiado de las buenas prácticas y usos del desarrollo de software.
- ▶ Engloban los conocimientos que son patrimonio de una organización.
- ▶ Proporcionan un marco para implementar procedimientos de aseguramiento de la calidad.
- ▶ Proporcionan continuidad entre el trabajo de distintas personas.

Los estándares pueden clasificarse como *de iure* o *de facto* (Ida, 2017; West, 2003):

1. *De iure*: Se crean a través de un proceso formal de acuerdo. Son adoptados por una institución de estandarización autorizada, toman un largo tiempo su desarrollo y tienen una duración bastante larga. Generalmente, son publicados por instituciones voluntarias especializadas en escribir estándares o promulgados por agencias gubernamentales que tienen autoridad regulatoria.
2. *De facto*: Surgen sin ningún plan formal, se crean a través de un proceso implícito de acuerdo y pueden ser desarrollados rápidamente, pero a su vez tienen un tiempo de vida corto.

▶ 4.2. Estándares del proceso de software

Los estándares tienen una variedad de contenidos que describen medidas, regulaciones y políticas particulares a seguir durante el desarrollo del proceso de software. Cada uno de los contenidos de un estándar tiene un objetivo que se debe cumplir con la asistencia de diferentes tecnologías y herramientas (Freericks, 2001).

Los estándares del proceso de software son necesarios para conseguir un marco común para hablar el mismo lenguaje en el desarrollo y la gestión de proyectos de desarrollo de software. La estructura general para los estándares de proceso de software consiste en etapas de proceso, regulaciones, listas de verificación, recomendaciones, fundamentos, glosarios, herramientas y métodos (Freericks, 2001).

Las etapas del proceso responden a la pregunta ¿qué hacer?, definiendo las actividades, los roles y documentos a ser elaborados para generar el producto de software deseado. También incluye el flujo de artefactos, el cual es esencial para construir el modelo de procesos.

El nivel de obligatoriedad de un estándar de proceso puede ser clasificado como (Freericks, 2001):

1. *Obligatorio*: Debe ser seguido sin excepción.
2. *Recomendado*: Genera una advertencia sobre cuándo no debe ser seguido.
3. *Guía*: Es opcional, puede ser o no utilizado.

Los estándares de procesos de software incluyen cinco categorías de listas de verificación que evalúan su grado de cumplimiento (Freericks, 2001):

1. *Listas de verificación de documentos*: Son las más frecuentes y típicamente implican frases como “El documento describirá”.
2. *Listas de verificación del proceso*: Restrigen aspectos como el orden de las actividades, los roles que realizan las actividades, los recursos de *hardware* o software, los cronogramas o el presupuesto. Por ejemplo: “Las pruebas unitarias deben ser realizadas por una persona externa al equipo de desarrollo y finalizada antes de la prueba del sistema”.
3. *Listas de verificación de software/hardware*: Hacen referencia explícita a los atributos o cualidades que tendrá el software o *hardware*. Por ejemplo: “La interfaz de usuario debe”.
4. *Listas de verificación de roles*: Contienen las obligaciones para los participantes del proceso. Por ejemplo: “El proveedor se asegurará de que se comprenda esta política”.
5. *Listas de verificación generales*: Son reglas que se definen en un amplio espectro y no se pueden asignar a ninguna de las categorías anteriores. Por ejemplo: “El proyecto no debe exceder”.

En la figura 4.1, se muestran los estándares del proceso de software y procesos industriales que se describirán en este capítulo.

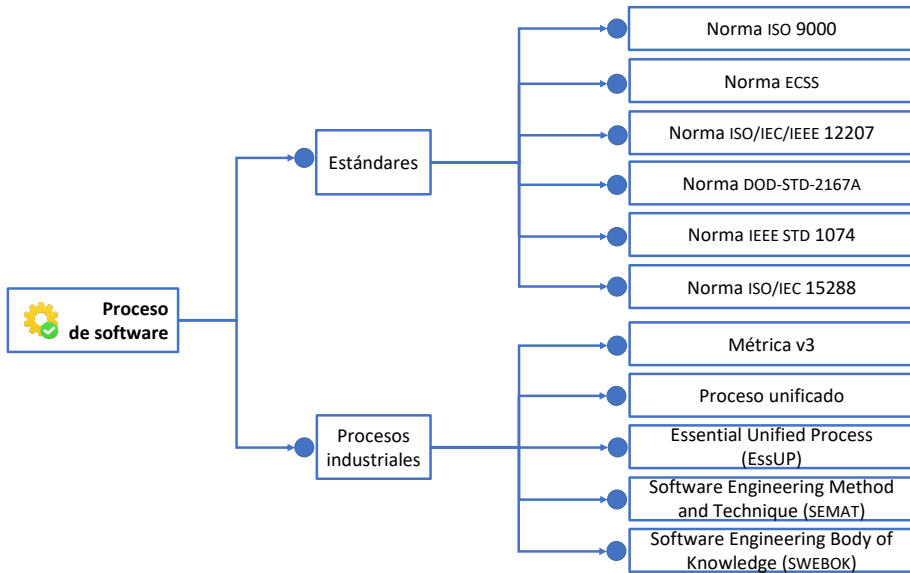


Figura 4.1. Estándares y procesos industriales de desarrollo de software

Fuente: elaboración propia.

Los estándares para productos, procesos y servicios son factores cruciales para el éxito en mercados competitivos, especialmente en los que la compatibilidad funcional es un prerrequisito, como el caso de las tecnologías informáticas (De Vries, 2013). A continuación, se presentará una descripción de los estándares más aplicados a los procesos de software: la norma **ISO 9000**, la European Cooperation for Space Standardization (**ECSS**), la norma **ISO/IEC/IEEE 12207**, la norma **DOD-STD-2167A**, la norma **IEEE STD 1074** y la norma **ISO/IEC 15288**.

4.2.1. Norma ISO 9000

La International Organization for Standardization (**ISO**), u Organización Internacional para la Estandarización, es una agencia internacional cuya finalidad es promover la estandarización internacional para facilitar el intercambio de bienes y servicios, así como el desarrollo científico y tecnológico.

La familia de estándares **ISO 9000** es una colección de normas orientadas a los sistemas de gestión de la calidad empresarial. Fue originalmente diseñada para la industria manufacturera y más tarde extendida a industrias de servicios y de software. Las organizaciones deben iniciar un sistema de calidad para supervisar todas las fases de un proceso de producción y entrega. Un sistema de calidad define los requisitos del proceso de desarrollo de la organización. Los detalles del sistema de calidad están contenidos en un manual de calidad que incluye estándares para la calidad y las actividades de desarrollo.

El estándar enfatiza tres amplios conceptos de calidad (Krishnan *et al.*, 1999):

1. Una organización debe lograr y mantener la calidad del producto o servicio producido para satisfacer continuamente las necesidades declaradas o implícitas del cliente.
2. Una organización debe hacer un seguimiento de los resultados de las prácticas de calidad y proporcionar confianza a su propia administración que se está logrando y manteniendo la calidad deseada.
3. Una organización debe otorgar confianza al comprador de que la calidad deseada se está logrando o se logrará en el producto o servicio brindado. Cuando se requiere contractualmente, esta disposición de confidencialidad puede involucrar acuerdos de demostración de requisitos.

La norma **ISO 9001:2000** aborda los requisitos para un sistema de gestión de calidad que ofrece la posibilidad a las empresas de demostrar su capacidad para entregar productos y servicios que cumplan con los requisitos de los clientes y regulatorios, que tiene como objetivo aumentar la satisfacción del cliente (Portuguese Institute of Quality, 2008). La norma es aplicable a todos los sectores y organizaciones sin importar su tipo, tamaño, producto o servicio. La norma especifica los requerimientos para establecer la confianza en la calidad de una organización (Valdés Cárdenas, 2005). Proporciona una guía descriptiva sobre los criterios que las organizaciones deben seguir para implementar un proceso de aseguramiento de la calidad que ayude a cumplir las expectativas del cliente, pero no prescribe cómo debe cumplirlos (**ISO/IEC 9000-3**, 2004). Las empresas desarrolladoras de software pueden aplicar esa norma para diseñar, desarrollar y mantener productos de software.

La norma **ISO 9001** se basa en ocho principios de gestión que pueden aplicarse para desarrollar el desempeño organizacional (Portuguese Institute of Quality, 2008):

1. *Enfoque en el cliente*: El cliente es la base fundamental de la organización. Por tanto, la organización debe entender las necesidades actuales y futuras del cliente, y realizar esfuerzos para cumplir sus expectativas.
2. *Liderazgo*: Los líderes deben definir los objetivos de la organización. A su vez, crear la ruta y las condiciones internas con las cuales los trabajadores se involucran para alcanzar los objetivos.
3. *Involucramiento del personal*: Participación efectiva del personal permitiendo que sus capacidades se utilicen para el beneficio de la organización.
4. *Enfoque en el proceso*: La forma más eficiente de obtener los resultados deseados es la gestión de las actividades y los recursos de la organización.
5. *Enfoque en el sistema en la gestión*: La visión de los procesos interrelacionados como un sistema contribuye a la efectividad y eficiencia de la organización para lograr sus objetivos.
6. *Mejora continua*: El compromiso permanente es la mejora continua en obtener un desempeño general de la organización.
7. *Enfoque factual en la toma de decisiones*: El análisis de datos es el punto de apoyo para una toma de decisiones efectiva.
8. *Relaciones con proveedores*: Las organizaciones y sus proveedores son mutuamente dependientes en que sus relaciones recíprocamente ventajosas aumentan su capacidad de crear valor.

La norma **ISO 9001** define principios de calidad total, ofrece lineamientos para definir un proceso de calidad y los procedimientos correspondientes. El manual de calidad es el documento que recoge la descripción de los procesos relevantes y los datos del proceso que deben recolectarse. Se compone de ocho secciones. Las tres primeras proporcionan información general sobre el estándar y las últimas cinco se centran en cómo implementarlo, como se observa en la figura 4.2.

1. *Sección 1: Alcance*. Especifica que la organización necesita cumplir los requisitos del cliente y requisitos regulatorios. También debe asegurarse de que sus empleados sigan las políticas y los proce-

dimientos, y a la vez optimizan la calidad a través de la mejora continua.

2. *Sección 2: Referencia normativa.* Son aquellas normativas relacionadas con la ISO 9000 para constituir los términos de la ISO 9001.
3. *Sección 3: Términos y definiciones.* Define los términos utilizados en la norma y resalta las diferencias entre versiones.
4. *Sección 4: Sistema de gestión de la calidad.* Describe los requisitos generales de la norma, abarcando actividades desde la documentación del manual de calidad y el control de documentos y registros hasta la determinación de la secuencia e interacción de los procesos para implementar acciones que logren los resultados planificados.
5. *Sección 5: Responsabilidad de la gestión.* Requiere el compromiso de la gestión con el sistema de gestión de la calidad y explica que la gestión debe dedicarse a los productos de la organización, productos y procesos de planificación y revisión.
6. *Sección 6: Gestión de recursos.* Proporciona los criterios necesarios para realizar un trabajo de manera competente y en un entorno seguro. En esta sección, también se tratan los recursos humanos, la planificación de la infraestructura y el entorno de trabajo.
7. *Sección 7: Realización del producto.* Define los pasos en el desarrollo del producto desde la fase de diseño inicial hasta la fase de entrega final.
8. *Sección 8: Medición, análisis y mejora.* Se centra en medir, analizar y mejorar el sistema de gestión de la calidad haciendo que las empresas realicen auditorías internas periódicas, monitoreen la satisfacción del cliente, controlen productos no conformes, analicen datos y tomen medidas correctivas y preventivas.

La norma ISO 9001 ha sido adaptada para la producción de software debido a que se ha reconocido como un proceso de producción con un propósito general diferente. Para adecuar la norma ISO 9001 al entorno de empresas de desarrollo de software, se definieron elementos teóricos y prácticos en una nueva norma, la ISO 9000-3, que se utiliza para interpretar la norma ISO 9001 en el ámbito de la ingeniería de software. El título de la norma ISO/IEC 9000-3 es “Guía para la aplicación de ISO 9001:2008 a software de computador” (ISO/IEC 9000-3, 2004).

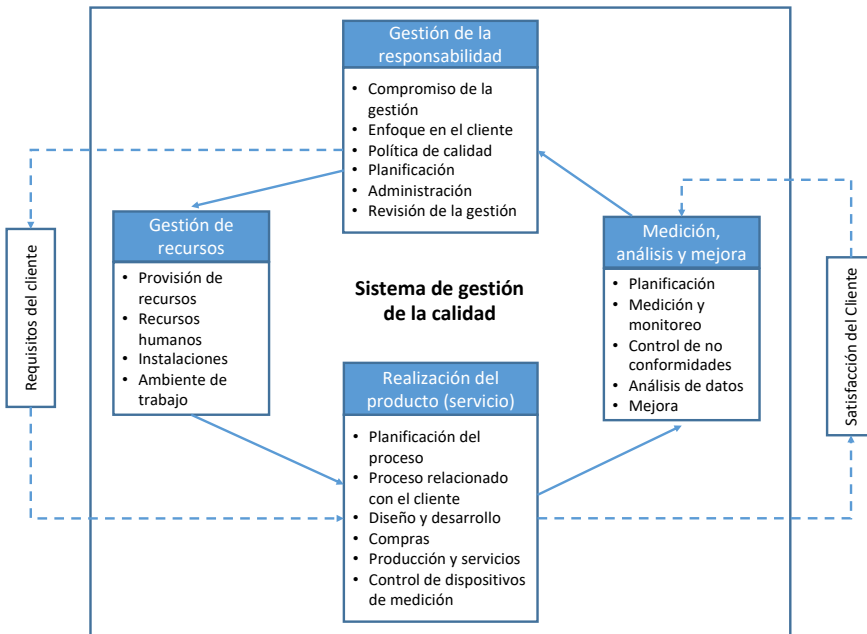


Figura 4.2. Procesos ISO 9001

Fuente: elaboración propia con base en Portuguese Institute of Quality (2008).

Las ideas fundamentales de la norma ISO 9000-3 son (Derniame *et al.*, 1999):

1. El control de calidad debe realizarse durante todas las fases de producción, adquisición y mantenimiento del software.
2. El comprador debe cooperar estrictamente con el proveedor del producto de software.
3. El proveedor debe definir su sistema de calidad y asegurarse de que su organización comprenda e implemente el sistema.

La norma ISO/IEC 9000-3 no impone un modelo de ciclo de vida específico. ISO/IEC 9000-3 se puede utilizar en situaciones contractuales cuando el comprador y el proveedor establecen que algunos elementos de calidad formarán parte del sistema de calidad del proveedor y este se compromete a seguir los principios de calidad definidos en la norma. Además, puede explotarse en situaciones no contractuales cuando el proveedor aplica voluntariamente el estándar de calidad para ser competitivo y garantizar la calidad de sus productos (Derniame *et al.*, 1999).

Las actividades específicas tratadas en **ISO/IEC 9000-3** se enumeran en la tabla 4.1. Aunque esta norma aborda principalmente el desarrollo de software personalizado en que el software específico se desarrolla según los requisitos del comprador, estos conceptos de calidad también se pueden usar en proyectos que desarrollan software empaquetado (Krishnan *et al.*, 1999).

Tabla 4.1. Actividades del proceso de software dirigidas por el sistema de calidad ISO 9000

Actividades del ciclo de vida	Actividades de soporte en desarrollo de software
• Revisión del contrato • Especificación de requisitos del comprador • Planificación del desarrollo • Planificación de la calidad • Diseño e implementación • Prueba y validación • Aceptación del producto • Replicación, entrega e instalación • Mantenimiento	• Gestión de configuración • Control de documentos • Mantenimiento de registros de calidad • Medición y métricas • Reglas, prácticas y convenciones • Herramientas y técnicas • Compras • Entrenamiento

Fuente: elaboración propia con base en Krishnan *et al.* (1999).

4.2.2. Norma ECSS

La Norma **ECSS** ha desarrollado un conjunto de estándares para los proyectos espaciales europeos (Jones *et al.*, 1997; **ECSS**, 2008). Estos estándares se derivan de **PSS-05** (**ESA** Board for Software Standardisation and Control [**BSSC**], 1995), un estándar espacial anterior que era más prescriptivo, exigía una gran documentación y modelos de cascada y desarrollo incremental.

ECSS tiene tres ramas principales: estándares de gestión, estándares de ingeniería y estándares de aseguramiento del producto. Los estándares más importantes relacionados con el proceso de software se muestran en la figura 4.3 y son:

- Estándares de ingeniería:
 - **ECSS-E-ST-10C** (2009): Especifica los requisitos de implementación de ingeniería del sistema para sistemas espaciales y desarrollo de productos espaciales.

- **ECSS-E-ST-10-06C (2009)**: Presenta los propósitos, las posiciones y los requisitos de la especificación de requisitos técnicos.
 - **ECSS-E-ST-10-02C (2009)**: Establece los requisitos para la verificación de un producto de sistema espacial, incluidos los requisitos para la implementación del programa de verificación.
 - **ECSS-E-ST-40C (2009)**: Se refiere al software del producto. Cubre todos los aspectos de la ingeniería de software espacial, incluida la definición de requisitos, diseño, producción, verificación y validación, transferencia, operaciones y mantenimiento.
- Estándares de aseguramiento del producto:
- **ECSS-Q-ST-30C (2009)**: Define el programa de garantía de confiabilidad y los requisitos de confiabilidad para los sistemas espaciales.
 - **ECSS-Q-ST-40C (2009)**: Define el programa de seguridad y los requisitos técnicos de seguridad.
 - **ECSS-Q-ST-80C (2009)**: Define los requisitos de garantía de productos de software para ser utilizados en el desarrollo y mantenimiento de software en sistemas espaciales.

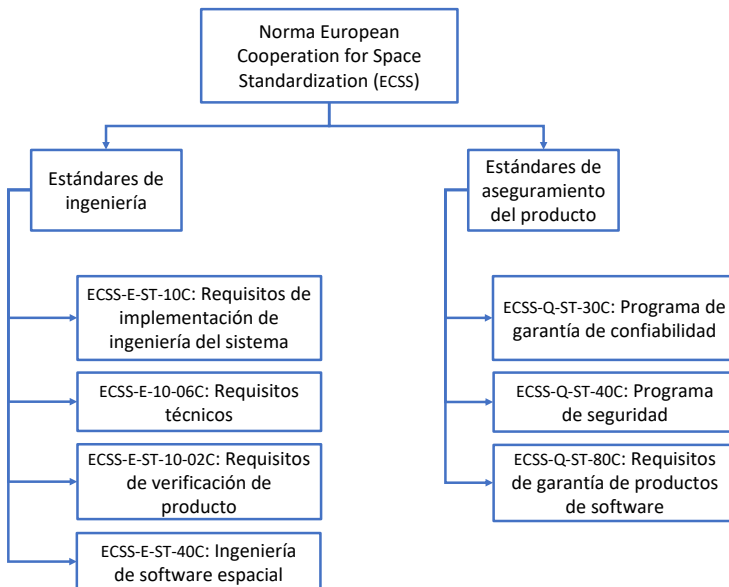


Figura 4.3. Procesos ECSS

Fuente: elaboración propia.

En la tabla 4.2, se definen los procesos de ciclo de vida y el número de artefactos o documentos requeridos por cada estándar. El término (X) indica una atención limitada dedicada al proceso.

Tabla 4.2. Procesos del ciclo de vida y número de artefactos

Procesos	ECSS-E-ST-10C	ECSS-E-10-06C	ECSS-E-ST-10-02C	ECSS-E-ST-40C	ECSS-Q-ST-30C	ECSS-Q-ST-40C	ECSS-Q-ST-80C
Diseño	X			X	X	X	X
Implementación				X	X	X	X
Gestión de configuración	X			(X)		(X)	X
Mantenimiento	X		(X)	X	X	(X)	X
Verificación	X		X	X	X	X	X
Validación				X	X	X	X
Artefactos	40	1	10	48	15	5	67

Fuente: elaboración propia con base en Ceccarelli y Silva (2013).

Se observa en la tabla 4.2 que la mayoría de los estándares describen el ciclo de vida completo. La certificación no se considera en los estándares espaciales. Los estándares **ECSS-E-ST** y **ECSS-Q** se notan como los más exigentes en términos de número de artefactos, siendo el **ECSS-E-ST-40C** el más exigente.

4.2.3. Norma ISO/IEC/IEEE 12207

El propósito del estándar **ISO/IEC 12207** (Ingeniería de sistemas y software - Procesos del ciclo de vida de software) es proporcional a un conjunto definido de procesos para facilitar la comunicación entre proveedores, clientes y otros *stakeholders* en el ciclo de vida de un producto de software (**ISO/IEC/IEEE 12207**, 2017). Los procesos están organizados en grupos y cada proceso es descrito en términos de entidades de proceso: propósito, resultados, actividades y tareas (Pino *et al.*, 2010).

El estándar **ISO/IEC 12207** agrupa las actividades que pueden ejecutarse durante el ciclo de vida en cinco procesos primarios, ocho procesos de soporte y cuatro procesos organizacionales (Emmerich, 1999; Land y Walz, 2007):

- ▶ *Procesos primarios*: Los llevan a cabo las partes principales, es decir, aquellos que inician o realizan el desarrollo, la operación o el mantenimiento de productos de software.
- ▶ *Procesos de soporte*: Respaldan los procesos primarios para contribuir al éxito y la calidad del proyecto. Un proceso de soporte es empleado y ejecutado, según sea necesario, por otro proceso.
- ▶ *Procesos organizacionales*: Utilizados por una organización para establecer e implementar la estructura subyacente del ciclo de vida, como la gestión y la mejora del proceso. Por lo general, se emplean fuera del ámbito de proyectos y contratos específicos. Sin embargo, las lecciones de estos proyectos y contratos contribuyen a la mejora de la organización.

Cada ciclo de vida está dividido en un conjunto de actividades y cada actividad está dividida en un conjunto de tareas. La figura 4.4 describe los procesos de los ciclos de vida primarios, de soporte y organizacionales.

Procesos primarios del ciclo de vida	Procesos de soporte del ciclo de vida	Procesos organizacionales del ciclo de vida
• Adquisición • Suministro • Desarrollo • Operación • Mantenimiento	• Documentación • Gestión de la Configuración • Aseguramiento de la Calidad • Verificación • Validación • Revisión • Auditoría • Resolución de Problemas	• Administración • Infraestructura • Mejora • Entrenamiento

Figura 4.4. Procesos iso/iec 12207

Fuente: elaboración propia.

Para que el estándar pueda ser aplicado a diferentes dominios, incluye un proceso de adaptación (*tailoring*) que define cómo el estándar es adaptado e indica aquellos procesos, actividades y tareas que pueden omitirse. Además, permite que se agreguen procesos, actividades y tareas, siempre que se especifiquen de una manera que cumpla con el estándar.

4.2.4. Norma DOD-STD-2167A

La norma **DOD-STD-2167A** (Defense System Software Development, o Desarrollo de software de sistemas de defensa) define un proceso de desarrollo de ocho fases requeridas para adquirir, desarrollar y mantener software por los departamentos y las agencias del Departamento de Defensa de Estados Unidos (Kempton *et al.*, 1988).

Tabla 4.3. Requisitos generales del proceso DOD-STD-2167A

Requisito general	Descripción
Gestión del desarrollo de software	• Ejecutar un proceso de desarrollo de ocho fases • Ofrecer soporte para auditoría y revisiones formales • Ejecutar la planificación del desarrollo de software produciendo un plan de desarrollo de software
Ingeniería de software	• Definir los métodos de ingeniería de software • Planificar, crear y mantener el ambiente de ingeniería de software • Planificar y gestionar: seguridad, selección de lenguaje de alto nivel, estándares de diseño y codificación, archivos de desarrollo de software, y recursos y reservas de procesamiento • Ejecutar la organización del software de computador • Documentar la trazabilidad de requisitos a diseño
Pruebas de cualificación formal	• Planificar las pruebas de cualificación formal • Planificar, crear y mantener el ambiente de pruebas de software • Proporcionar <i>testers</i> independientes • Mostrar la trazabilidad de requisitos a casos de prueba
Evaluación del producto de software	• Proporcionar evaluadores independientes • Ejecutar evaluaciones finales internas • Mantener registros de evaluación • Utilizar criterios de evaluación
Gestión de configuración de software	• Proporcionar identificación de la configuración • Proporcionar control de la configuración • Proporcionar contabilidad del estado de la configuración • Proporcionar almacenamiento, manejo y entrega de productos • Procesar propuesta de cambios de ingeniería
Transición al soporte de software	• Proporcionar código fácil de mantener • Planificar la transición • Proporcionar instalación, revisión y soporte continuo según lo contratado • Proporcionar soporte y documentación operacional

Fuente: elaboración propia con base en Kempton *et al.* (2008).

DOD-STD-2167A fue desarrollado para reducir los desfases en costos y planificación e incrementar la calidad, confiabilidad y facilidad de mantenimiento de los productos de software desarrollados por este departamento (McCord, 1990).

El estándar detalla la gestión, ingeniería, pruebas, evaluación del producto y actividades de gestión de configuración que deben realizarse en cada una de las ocho fases. La tabla 4.3 resume los requisitos generales del estándar. Los productos entregables especificados por el estándar consisten en la documentación del proyecto y el código fuente.

El ciclo del desarrollo de software consiste en las siguientes fases (Kempton *et al.*, 2008) (figura 4.5):

- ▶ *Análisis/diseño de requisitos del sistema:* Se realiza una especificación preliminar del sistema para determinar si los requisitos son consistentes y completos. El diseño y la arquitectura del sistema son documentados en una especificación de sistema. Se elabora también un plan de desarrollo de software.
- ▶ *Análisis de requisitos de software:* Se desarrolla la arquitectura total del software, se asignan requisitos a ítems de configuración de software y se planifica un conjunto de requisitos para las interfaces externas.
- ▶ *Diseño preliminar:* Establece requisitos de diseño y desarrolla un diseño preliminar de las interfaces internas. Las decisiones de diseño son documentadas en un documento de diseño de software.
- ▶ *Diseño detallado:* Se realiza asignando requisitos a las unidades de software. Fomenta el refinamiento del diseño en lugar de mantenerlo estático. Se establece un archivo de desarrollo de software utilizado para registrar información relacionada con el desarrollo o soporte.
- ▶ *Codificación y unidad de software de computador:* Se asegura que los algoritmos y la lógica utilizada son correctos y que el software satisface los requisitos especificados. Los resultados de las pruebas son registrados en archivos de desarrollo de software.
- ▶ *Pruebas e integración de componentes de software de computador:* Similar a la fase previa, pero integra múltiples unidades de software en componentes que luego son probados. Los procedimientos utilizados para iniciar, conducir y analizar las pruebas son documentados en descripciones de pruebas de software.

- *Prueba de ítems de configuración de software de computador*: Los resultados de estas pruebas son registradas en el reporte de pruebas de software.
- *Pruebas e integración del sistema*: Involucra auditorías de la configuración física y funcional.

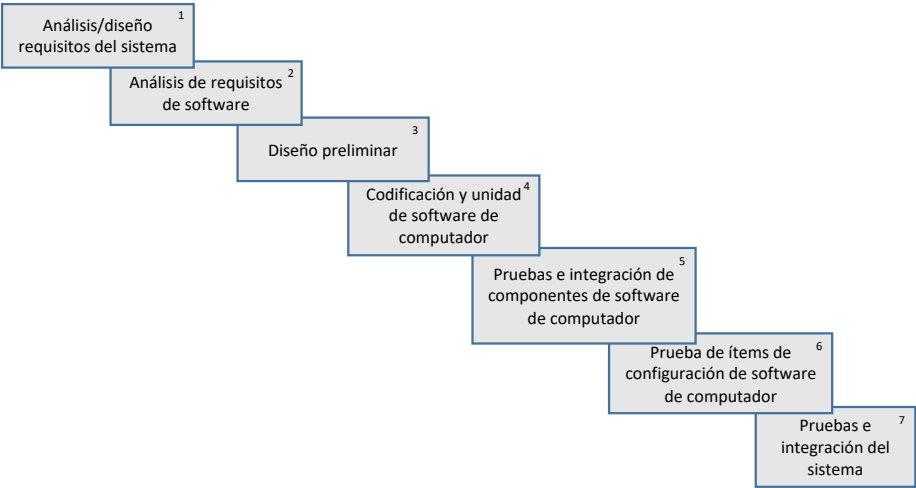


Figura 4.5. Ciclo de desarrollo de software del proceso DoD-STD-2167A

Fuente: elaboración propia con base en McCord (1990).

4.2.5. Norma IEEE STD 1074

La norma **IEEE 1074** especifica los procesos de ciclo de vida de software para desarrollar y mantener software. Identifica un conjunto no ordenado de actividades esenciales que deben incluirse para desarrollar un producto de software. Las actividades están organizadas como grupos lógicos y cohesivos llamados grupos de actividades (**IEEE-STD 1074**, 2006). Estos grupos de actividades cubren el ciclo de vida del desarrollo de software desde la identificación de la idea o necesidad de los usuarios hasta su retiro.

La norma define las actividades que constituyen los procesos necesarios para el desarrollo y el mantenimiento de software, ya sea parte de un sistema mayor o autónomo, y los procesos de gestión y soporte a lo largo de todo el ciclo de vida.

El proceso de software está compuesto de 17 grupos de actividades, como se observa en la figura 4.6. El proceso está conformado en total por 65 actividades y cubre las siguientes familias de actividades (Acuna *et al.*, 2006; Bernardos, 2004):

- ▶ *Grupo de actividades de gestión de proyectos*: Estas actividades permiten iniciar un proyecto de software, planificar su gestión y monitorear y controlar el proyecto a lo largo de su ciclo de vida. Se tiene una consideración especial a la gestión del riesgo.
- ▶ *Grupo de actividades orientadas al desarrollo de software*: Estas actividades producen, instalan, operan, mantienen y retiran software. El grupo de actividades se divide en los siguientes subgrupos.
 - *Grupo de actividades de predesarrollo*: Se realiza antes que el desarrollo inicie. Estas actividades exploran conceptos y asignan requisitos del sistema.
 - *Grupo de actividades de desarrollo*: Estas actividades son realizadas durante el desarrollo de un producto de software. Incluyen requisitos, diseño e implementación.
 - *Grupo de actividades de posdesarrollo*: Estas actividades son realizadas para instalar, mantener y retirar un producto de software.
 - *Grupo de actividades integrales*: Estas actividades son necesarias para completar exitosamente el resto de actividades del proyecto. Se realizan en forma simultánea con las actividades orientadas al desarrollo. Incluyen evaluación de procesos y productos, la gestión de configuración, el desarrollo de la documentación y el entrenamiento del personal.

El modelo está representado principalmente por actividades y artefactos. Presenta una descripción detallada de las actividades, así como de los productos realizados en los procesos de software. Sin embargo, es un modelo informal que se enfoca cualitativamente en la interacción actividad-artefacto.

La norma **IEEE** 1074 no fue proyectada para ser un modelo de ciclo de vida del software; describe las prácticas para seleccionar y adaptar un modelo de ciclo de vida. En este contexto, el modelo proporciona información sobre qué debe documentarse, pero no especifica cómo (Acuna *et al.*, 2006). Por tanto, el modelo debe ser adaptado a cada proyecto particular de software

asociando las actividades estipuladas en la norma a su propio ciclo de vida de software.

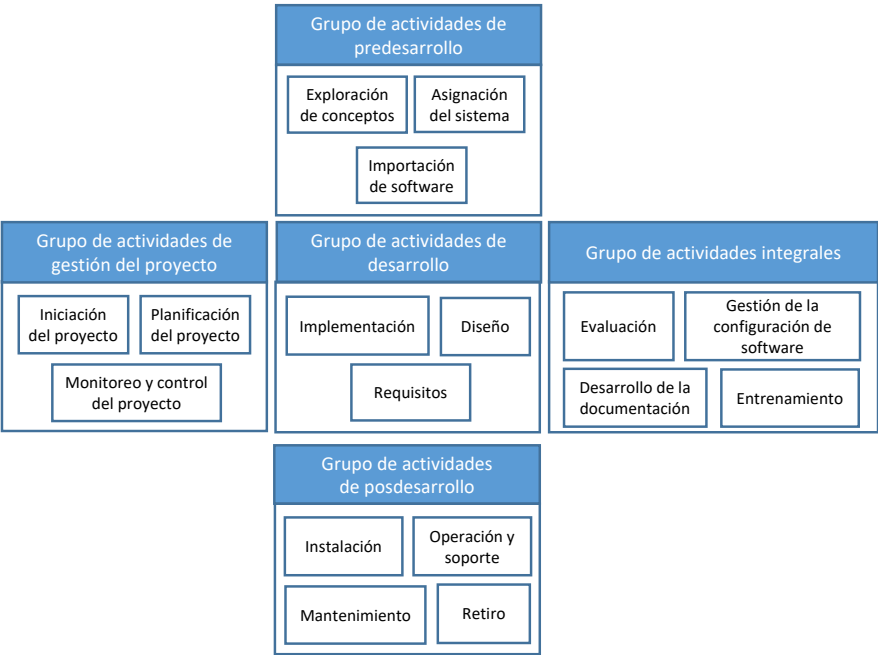


Figura 4.6. Procesos de la norma IEEE 1074

Fuente: elaboración propia con base en IEEE-STD 1074 (2006).

4.2.6. Norma ISO/IEC 15288

La norma ISO/IEC 15288:2015 (Ingeniería de sistemas y software: Procesos del ciclo de vida del sistema) (ISO/IEC 15288, 2015) establece un marco común para los procesos del ciclo de vida del sistema. Se aplica a la adquisición de sistemas, que pueden estar compuestos por productos, servicios o ambos, así como a la oferta, el desarrollo, la operación, el mantenimiento y la eliminación de sistemas, ya sea que se realicen interna o externamente a una organización.

El objetivo de ISO/IEC 15288:2015 es proporcionar un conjunto de finido de procesos para facilitar la comunicación entre los adquirentes, proveedores y otras partes interesadas en el ciclo de vida de un sistema.

Los cuatro grupos de procesos de **ISO/IEC 15288**, así como las relaciones primarias entre los grupos, se representan en la figura 4.7. La empresa establece acuerdos en su función de cadena de suministro como proveedor de productos o servicios de valor agregado, así como un adquirente de actividades, productos y servicios de organizaciones proveedoras. La empresa autoriza, respalda y supervisa proyectos para realizar trabajos relacionados con el sistema. Las organizaciones generalmente tienen la responsabilidad de las actividades de operación continua, mientras la mayoría de los proyectos están dirigidos a trabajos relacionados con el sistema utilizando procesos técnicos y sus actividades dentro de periodos discretos (**ISO/IEC/IEEE 15288**, 2015).

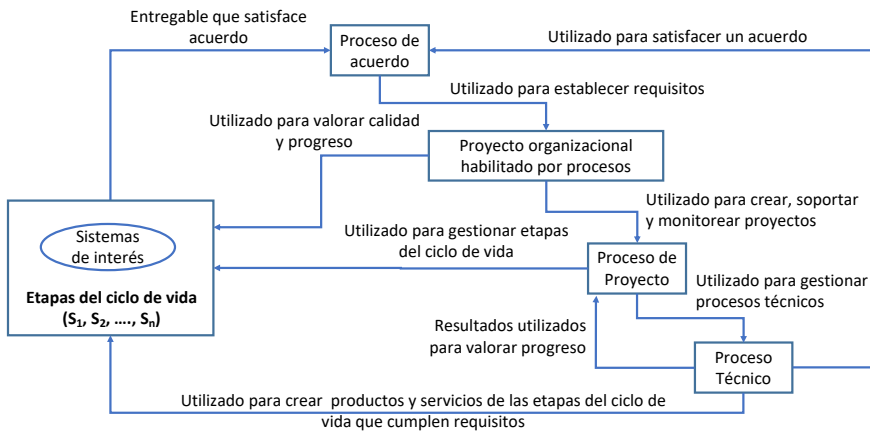


Figura 4.7. Rol de los procesos **ISO/IEC 15288**

Fuente: elaboración propia con base en **ISO/IEC/IEEE 15288** (2015).

La norma **ISO/IEC 15288** se diseñó y desarrolló para satisfacer necesidades organizacionales fundamentales al proporcionar procesos adecuados en cada categoría, como se muestra en la figura 4.8. A continuación, se describen en detalle los cuatro procesos (**ISO/IEC 15288**, 2008):

- *Procesos de acuerdo:* Establecen acuerdos con entidades organizacionales internas y externas al proyecto para adquirir o proveer un producto o servicio. El proceso de adquisición proporciona los medios para realizar negocios con un proveedor de productos del sistema. El proceso de suministro proporciona los medios para llevar a cabo un proyecto en el que el resultado es un producto o servicio que se entrega al adquirente.

- *Procesos empresariales*: Para asegurar la capacidad de la organización en adquirir y suministrar productos o servicios a través de la iniciación, el soporte y control de proyectos.
- *Procesos del proyecto*: Para establecer y evolucionar los planes del proyecto, ejecutar los planes del proyecto, evaluar el logro real y el progreso en relación con los planes, y controlar la ejecución del proyecto hasta su cumplimiento.
- *Procesos técnicos*: Para contribuir a la satisfacción de los objetivos técnicos para una o más etapas del ciclo de vida. Estos procesos se utilizan a fin de definir los requisitos de un sistema, transformar los requisitos en un producto efectivo para permitir una reproducción consistente del producto donde sea necesario, usar el producto para proporcionar los servicios requeridos y mantener la prestación de esos servicios y deshacerse del producto cuando se retira del servicio.

Cada proceso se define con un propósito, resultados y actividades. La norma ISO/IEC 15288 comprende 25 procesos que tienen 123 resultados derivados de 403 actividades.

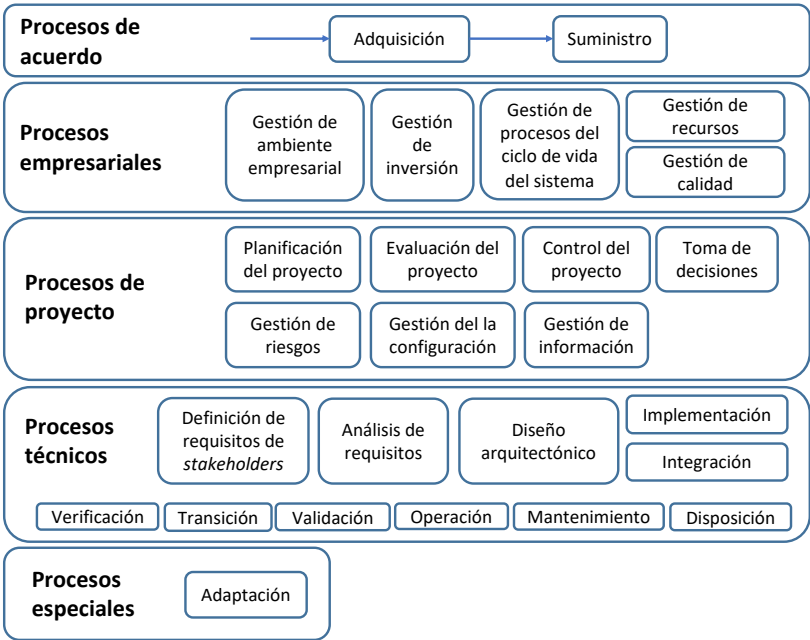


Figura 4.8. Categorías de procesos de ISO/IEC 15288

Fuente: elaboración propia con base en ISO/IEC/IEEE 15288 (2015).

Además de los procesos, el estándar **ISO/IEC 15288** define un proceso de adaptación que se debe utilizar para guiar a las organizaciones, las empresas y los proyectos en su adaptación del estándar a su entorno, así como para cumplir con sus requisitos y necesidades específicas. La adaptación puede ser necesaria para cumplir un acuerdo, influir en un proyecto en el que se requiere cumplir un acuerdo basado en el uso del estándar y reflejar las necesidades de una organización en el suministro de productos o servicios del sistema (Strandberg, 2016).

El resultado de la adaptación pueden ser procesos del ciclo de vida del sistema nuevos o modificados, la definición de etapas individuales que influyen en un acuerdo o un modelo de ciclo de vida definido en términos de etapas que definen las contribuciones que hacen al sistema.

► 4.3. Procesos industriales

Hay otros tipos de procesos que no son estándares internacionales, sino que son propuestas con origen en la academia y la industria, y que llegado el caso se convierten en estándares nacionales o recomendaciones implementadas por diversos sectores de la industria de software. A continuación, se describirán los procesos Métrica v.3, el proceso unificado, Essential Unified Process (EssUP), Software Engineering Methods and Theory (**SEMAT**) y Software Engineering Body of Knowledge (**SWEBOK**).

4.3.1. Métrica v.3

La metodología Métrica v.3 perteneciente al Ministerio de Hacienda y Función Pública de España tiene un enfoque orientado al proceso. Esta descompone cada uno de los procesos en actividades y estas a su vez en tareas. Para cada tarea, se describe su contenido haciendo referencia a sus principales acciones, productos, técnicas, prácticas y participantes (Métrica v.3, 2020).

Los procesos de la estructura principal de Métrica v.3 son los siguientes y se observan en la figura 4.9 (Métrica v.3, 2020):

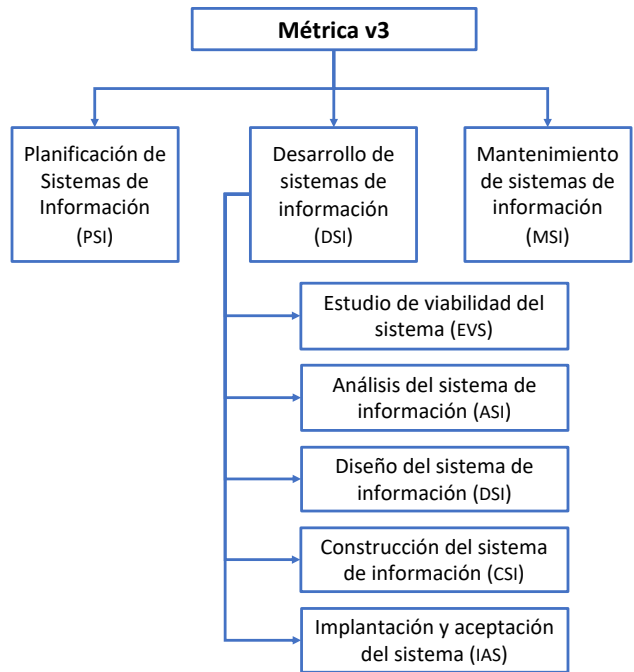


Figura 4.9. Categorías de procesos de Métrica v.3

Fuente: elaboración propia.

- *Planificación de sistemas de información (PSI)*: Tiene como objetivo la obtención de un marco de referencia para el desarrollo de sistemas de información que responda a los objetivos estratégicos de la organización. El marco de referencia consta de una descripción de la situación actual, un conjunto de modelos que constituye la arquitectura de información, una propuesta de proyectos a desarrollar en los próximos años, una propuesta de calendario para la ejecución de esos proyectos, la evaluación de los recursos necesarios para los proyectos a desarrollar en el próximo año y un plan de seguimiento y cumplimiento de lo propuesto mediante unos mecanismos de evaluación adecuados.
- *Desarrollo de sistemas de información*: Está conformado por los siguientes subprocesos.
 - *Estudio de viabilidad del sistema (EVS)*: Tiene como objetivo el análisis de un conjunto concreto de necesidades para propo-

ner una solución a corto plazo, que considere restricciones económicas, técnicas, legales y operativas.

- *Análisis del sistema de información (ASI)*: Su objetivo es la obtención de una especificación detallada del sistema de información que satisfaga las necesidades de información de los usuarios y sirva de base para el posterior diseño del sistema.
- *Diseño del sistema de información (DSI)*: Su objetivo es la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información.
- *Construcción del sistema de información (CSI)*: Tiene como objetivo generar el código de los componentes del sistema de información, se desarrollan los procedimientos de operación y seguridad, y se elaboran los manuales de usuario final y de explotación, a fin de asegurar el correcto funcionamiento del sistema para su posterior implantación.
- *Implantación y aceptación del sistema (IAS)*: Su objetivo es la entrega y aceptación del sistema en su totalidad y la realización de todas las actividades necesarias para el paso a su producción.
- *Mantenimiento de sistemas de información (MSI)*: Su objetivo es la obtención de una nueva versión de un sistema de información a partir de las peticiones de mantenimiento que los usuarios realizan con motivo de un problema detectado en el sistema o por la necesidad de una mejora de este.

Para cada fase, Métrica v.3 define un conjunto de tareas y objetivos que deben ser realizados. Incluye también un conjunto de interfaces que definen una serie de actividades de tipo organizacional o de soporte al proceso. Las interfaces descritas en la metodología son (Métrica v.3, 2020):

- *Gestión de proyectos (GP)*: Se encarga de la planificación, el seguimiento y el control de las actividades y de los recursos humanos y materiales que intervienen en el desarrollo de un sistema de información.
- *Seguridad (SEG)*: Incorpora mecanismos de seguridad adicionales a los que se proponen en la metodología, asegurando el desarrollo de cualquier tipo de sistema a lo largo de los procesos que se realicen para su obtención.

- *Aseguramiento de la calidad (CAL)*: Proporciona un marco común de referencia para la definición y puesta en marcha de planes específicos de aseguramiento de calidad aplicables a proyectos concretos. Aplica técnicas de calidad para controlar la calidad de los resultados.
- *Gestión de configuración (GC)*: Mantiene la integridad de los productos que se obtienen a lo largo del desarrollo, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo del sistema disponen de la versión adecuada de los productos que manejan.

Métrica v.3 proporciona un conjunto de métodos y técnicas que guía a los distintos profesionales en la obtención de los diversos productos de los procesos del ciclo de vida de un proyecto informático. Las técnicas introducidas se categorizan en desarrollo, gestión de proyectos, prácticas y soporte por herramientas (Métrica v.3, 2020).

La estructura y los perfiles de los participantes que intervienen en los diversos procesos deberán adaptarse y dimensionarse en cada momento de acuerdo con las características particulares de cada proyecto. Los perfiles establecidos son directivo, jefe de proyecto, consultor, analista y programador. Para cada uno de estos perfiles, se analiza una serie de características importantes a la hora de delimitar su participación en el proyecto: correspondencia con participantes de Métrica v.3, responsabilidades o funciones a desempeñar en cada uno de los procesos y perfil o características propias de cada uno de los participantes (Métrica v.3, 2020).

4.3.2. El proceso unificado

El proceso unificado es un conjunto de mejores prácticas recomendadas para el desarrollo de software (Priestley y Utt, 2000). Este tiene sus orígenes en el proceso *Objectory* y recibió influencia de muchas fuentes que incluye el proceso desarrollado de *Rational*. El trabajo conjunto de Ivar Jacobson, Grady Booch y James Rumbaugh logró la unificación en 1999 de los principales métodos para el análisis y diseño orientado a objetos. El proceso unificado es un *framework* genérico de procesos basado en componentes y utiliza Unified Modeling Language (**UML**) como mecanismo para modelar sus artefactos (Halling *et al.*, 2002). El proceso unificado posee las siguientes características (Booch *et al.*, 2017):

- *Guiado por casos de uso*: Como un modelo de proceso de software típico, su característica dominante es conducir procesos por medio de casos, en particular, se utilizan los casos de uso que provienen de **UML** (figura 4.10). Los casos de uso no solo definen los requisitos del sistema, sino que también impulsan el análisis, la implementación y las pruebas del sistema. De esta forma, los usuarios se involucran tempranamente en el desarrollo y se facilita la trazabilidad de los requisitos.

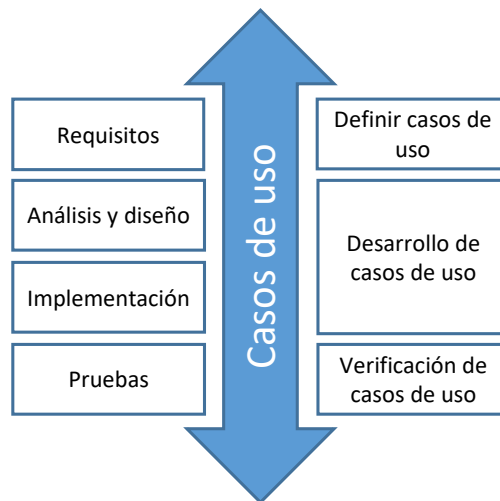


Figura 4.10. El proceso unificado es guiado por casos de uso

Fuente: elaboración propia.

- *Centrado en la arquitectura*: La arquitectura incluye cinco vistas: modelo de casos de uso, modelo de análisis, modelo de diseño, modelo operativo y modelo de implementación. Cada vista describe un modelo del sistema y sirve de marco guía para el trabajo del proceso iterativo. Las vistas están basadas en modelos visuales **UML**. Los modelos no son independientes, ya que los elementos de un modelo trazan dependencias con otros modelos. La figura 4.11 ilustra las relaciones entre los modelos.

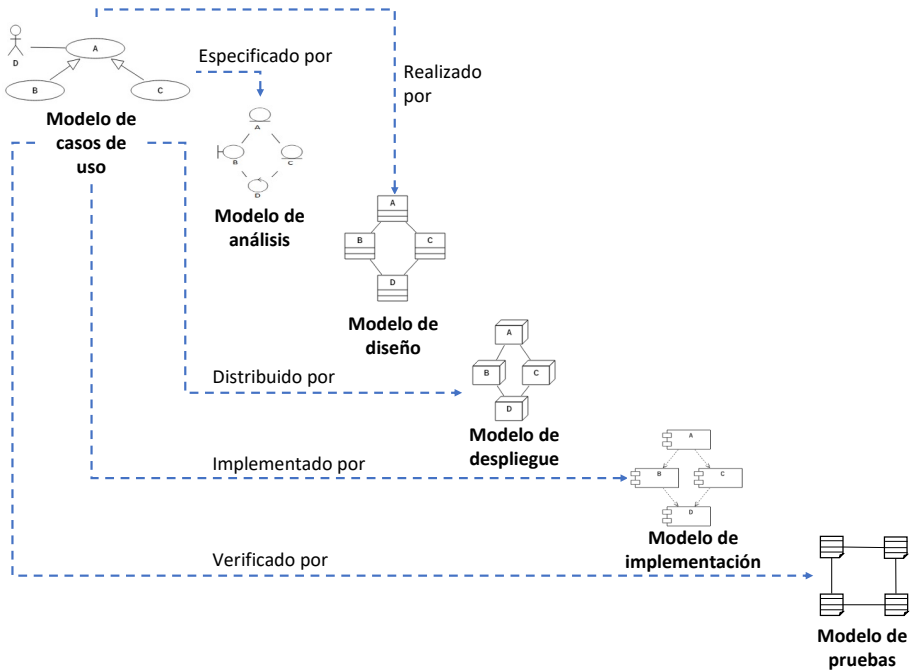


Figura 4.11. El proceso unificado es centrado en una arquitectura basada en modelos

Fuente: elaboración propia con base en Jacobson *et al.* (2012).

- **Ciclo de vida iterativo e incremental:** El proceso unificado repite una serie de ciclos que constituyen el proceso de desarrollo del sistema. Al final, una versión del producto se entrega a los usuarios. El ciclo de vida pasa por cuatro fases: inicio, elaboración, construcción y transición. Cada fase contiene una o más iteraciones. Una iteración típica cubrirá los siguientes flujos de trabajo centrales: modelado de negocios, requisitos, análisis, diseño, implementación, pruebas y despliegue. Estos elementos se presentan en la figura 4.12. Cada fase tiene sus objetivos claros y define hitos para esos objetivos. Los hitos determinan la obtención de un conjunto de artefactos.

La figura 4.12 muestra los flujos de trabajo a la izquierda. Las curvas se aproximan a la medida en que los flujos de trabajo se llevan a cabo en cada iteración a través del proceso de desarrollo. La figura muestra dos tipos de relaciones (Pons *et al.*, 2000):

- *Relaciones horizontales*: Entre modelos que pertenecen al mismo flujo de trabajo en diferentes iteraciones.
- *Relaciones verticales*: Entre modelos que pertenecen a la misma iteración en diferentes flujos de trabajo.

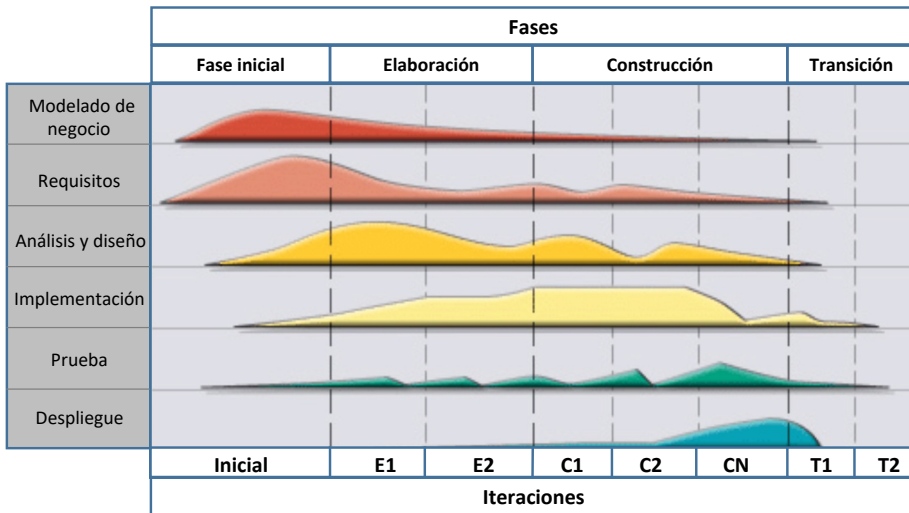


Figura 4.12. El proceso unificado tiene un ciclo de vida iterativo e incremental

Fuente: elaboración propia con base en Jacobson *et al.* (2012).

El proceso unificado tiene cuatro conceptos básicos: trabajador, actividad, artefacto y flujo de trabajo (Priestley y Utt, 2000):

- *Trabajador*: Básicamente un rol es un conjunto de comportamientos y responsabilidades que un individuo puede llevar a cabo. De hecho, un trabajador puede representar a más de un individuo y un individuo puede representar a más de un trabajador. Ejemplos: arquitecto, diseñador de pruebas, implementador, etc.
- *Actividad*: Es algo que hace un trabajador y que proporciona un resultado significativo. Ejemplos: desarrollo de guías de diseño, prueba de un componente, etc.
- *Artefacto*: Es un producto de una actividad. Ejemplos: modelo de diseño, componentes, procedimiento de pruebas, etc.
- *Flujo de trabajo*: Es un grupo lógico de actividades. Ejemplos: diseño, implementación, pruebas, etc.

En la tabla 4.4, se presentan las cuatro fases del proceso unificado junto con su descripción e hitos.

Tabla 4.4. Procesos del ciclo de vida y número de artefactos

Fase	Descripción	Hitos
Inicio	Se realiza una descripción de objetivos del producto, un modelo de casos de uso, un análisis de negocio del producto y se identifican riesgos.	Objetivos del ciclo de vida.
Elaboración	Se realiza una especificación detallada de los casos de uso, se diseña la arquitectura del sistema y un plan de proyecto detallado.	Línea base de la arquitectura con modelos constituyentes del sistema.
Construcción	Creación del producto de software.	Funcionalidad operativa inicial.
Transición	Se realizan pruebas al producto en las que se detectan y corrigen defectos. Incluye actividades de formación e instalación.	Versión del producto.

Fuente: elaboración propia con base en Jacobson *et al.* (2012).

La descripción de los flujos de trabajo del proceso unificado se presenta en la tabla 4.5.

Tabla 4.5. Flujos de trabajo del proceso unificado

Flujo de trabajo	Objetivo	Roles	Actividades	Modelos
Requisitos	Descripción de los requisitos del sistema	- Analista de sistemas - Arquitecto	- Encontrar actores y casos de uso - Priorizar casos de uso	Modelo de casos de uso
		- Especificador de casos de uso - Diseñador de interfaces de usuario	- Detallar casos de uso - Realizar prototipo de interfaz de usuario - Estructurar el modelo de casos de uso	

Flujo de trabajo	Objetivo	Roles	Actividades	Modelos
Análisis	Los requisitos del cliente se refinan y estructuran para una mejor comprensión del sistema	• Arquitecto • Ingeniero de casos de uso • Ingeniero de componentes	• Análisis de la arquitectura • Analizar casos de uso • Analizar clases • Analizar paquetes	Modelo de análisis
Diseño	Búsqueda y definición del sistema de software (solución)	• Arquitecto • Ingeniero de casos de uso • Ingeniero de componentes	• Diseño de la arquitectura • Diseñar casos de uso • Diseñar clases • Diseñar subsistemas	Modelo de diseño y modelo de despliegue
Implementación	Codificación del diseño (solución) en términos de componentes y subsistemas de implementación	• Arquitecto • Integrador de sistema • Ingeniero de componentes	• Implementación de la arquitectura • Integrar sistemas • Implementar subsistemas • Implementar clases • Realizar pruebas de unidad	Modelo de implementación
Pruebas	Verificación de la implementación probando cada construcción	• Ingeniero de pruebas • Ingeniero de pruebas de integración • Ingeniero de pruebas de sistema • Ingeniero de componentes	• Planificar pruebas • Diseñar pruebas • Implementar pruebas • Realizar pruebas de integración • Realizar pruebas de sistema • Evaluar pruebas	Modelo de pruebas

Fuente: elaboración propia con base en Jacobson *et al.* (2012).

4.3.3. EssUP

El proceso EssUP fue desarrollado por Ivar Jacobson en 2006, está basado en el Rational Unified Process (**RUP**) y tiene en cuenta la filosofía del desarrollo ágil. Propone un enfoque novedoso para proporcionar al usuario la “esencia” de la información mediante presentaciones más agradables, como tarjetas coloridas y brindando al usuario la base para aprender más, según sea necesario (Hui *et al.*, 2015).

Los aspectos “esenciales” del proceso se refieren a codificar los siguientes elementos (Ivar Jacobson International, 2015):

- Factores críticos de éxito: Elementos que, si los equipos no pueden hacer, o no lo hacen bien, entonces lucharán constantemente para tener éxito.
- Prácticas comunes, útiles, modulares y ágiles: Que ofrecen un valor real si se implementan apropiadamente.
- Elementos que no se deberían tener que cambiar, sino que se espera construir y reutilizar como una base segura para escalar la aplicación de las prácticas ágiles.

EssUP codifica un conjunto de prácticas ágiles populares estándares en la forma de “herramientas” granulares, a diferencia de las prácticas genéricas más grandes en las que suelen agruparse. Las prácticas EssUP proporcionan una caja de herramientas básica que cubre los aspectos comunes y críticos del desarrollo basado en equipos.

Las prácticas definidas en EssUP son (Ivar Jacobson International, 2015):

- *Aspectos esenciales de la propiedad del producto*: Poseer, evolucionar y comunicar la visión del producto y guiar la evolución del producto para lograr la visión.
- *Aspectos esenciales del backlog del producto*: Capturar las necesidades de los usuarios de un sistema de software como una lista clasificada prioritariamente de elementos que pueden ser compilados independientemente.
- *Aspectos esenciales de los equipos ágiles*: Un equipo autoorganizado maximiza su rendimiento mediante el uso de un enfoque de colaboración altamente colaborativo.
- *Aspectos esenciales de la reunión diaria*: Realizar reuniones cortas diarias, de todo el equipo, para reafirmar el enfoque de la entrega, evaluar el progreso, acordar planes de trabajo y tomar medidas para eliminar los obstáculos al progreso.

- ▶ *Aspectos esenciales del desarrollo ágil*: Agregar valor a un producto extendiéndolo de forma incremental y asegurándose de que siga siendo utilizable, liberable y fácil de mantener.
- ▶ *Aspectos esenciales de la retrospectiva ágil*: Realizar mejoras graduales en la forma de trabajar a través de retrospectivas repetidas regularmente.
- ▶ *Aspectos esenciales del timebox*: Realizar avances del trabajo como una serie de marcos de tiempo enfocados, y evaluar y replanificar el trabajo al final de cada *timebox*.

El proceso proporciona mapas relacionados con qué se trabaja y qué es lo que se hace de la siguiente manera (Ivar Jacobson International, 2015):

- ▶ *Tres áreas de preocupación*: Cliente (color verde), solución (color amarillo) y esfuerzo (color azul).
- ▶ *Siete asuntos clave que se deben gestionar (conocidos como ALPHA)*: Stakeholders, oportunidades, requisitos, sistema de software, equipo, forma de trabajo y trabajo, como se ilustra en la figura 4.13.
- ▶ *Quince espacios de actividades*: Describen los tipos generales de elementos que se deben hacer bien si se quiere progresar y completar un esfuerzo de desarrollo de software.

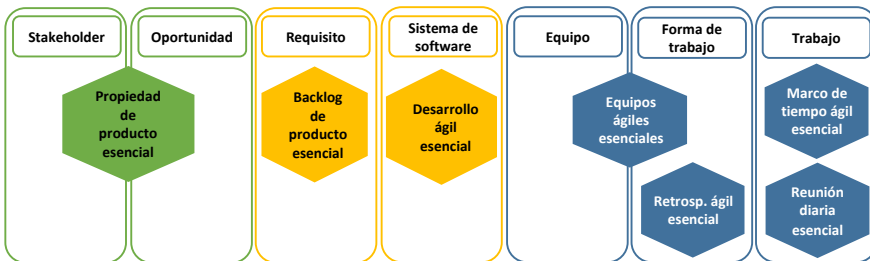


Figura 4.13. Prácticas esenciales mostradas a través de Abstract-Level Progress Health Attribute (ALPHA)

Fuente: elaboración propia con base en Ivar Jacobson International (2015).

4.3.4. SEMAT

El **SEMAT** es un grupo abierto que consta de voluntarios que comparten una motivación y dedicación comunes hacia la refundación de la ingeniería de software. Fue creado en septiembre de 2009 por Bertrand Meyer, Richard

Soley e Ivar Jacobson, quienes afirmaron que había llegado el momento de cambiar fundamentalmente la forma en que las personas trabajan con los métodos de desarrollo de software (Kajko-Mattsson *et al.*, 2012; Jacobson *et al.*, 2012; Elvesæter *et al.*, 2012).

La iniciativa **SEMAT** se enfoca en dos objetivos principales: a) encontrar un núcleo de elementos ampliamente acordados y b) definir una base teórica sólida (Dahhane *et al.*, 2016).

El enfoque del núcleo del **SEMAT** permite definir una base común para proyectos de desarrollo de software y construir métodos a partir de prácticas que se puedan definir y aplicar de manera independiente (Object Management Group [OMG], 2015). Las prácticas se agrupan para formar métodos adaptados a necesidades específicas de la organización, proyecto o equipo de trabajo.

El núcleo se puede describir usando un pequeño conjunto de elementos, los cuales se agrupan en tres áreas de conocimiento: cliente, solución y esfuerzo (Dahhane *et al.*, 2016):

- *Cliente*: Contiene todo lo relacionado con el uso real y la explotación del sistema de software que se producirá.
- *Solución*: Contiene todo lo que se necesita para hacer las especificaciones y el desarrollo del sistema de software.
- *Esfuerzo*: Contiene todo lo relacionado con el equipo y la forma en que abordan su trabajo.

El núcleo se describe utilizando un lenguaje que define la sintaxis abstracta, la semántica dinámica, la sintaxis gráfica y la sintaxis textual. Las relaciones entre estos conceptos se muestran en la figura 4.14.

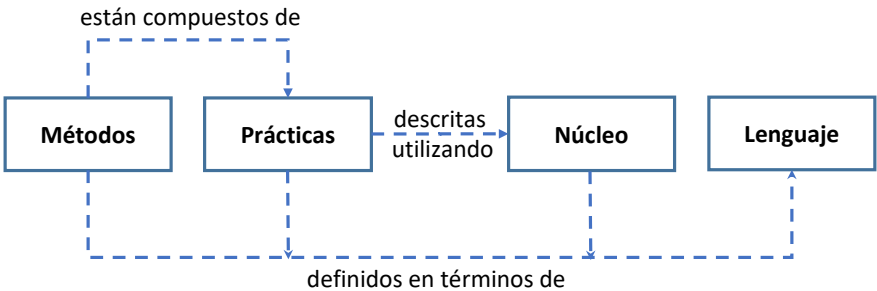


Figura 4.14. Elementos del SEMAT

Fuente: elaboración propia con base en OMG (2015).

Los elementos que componen cada una de las áreas de conocimiento y que a su vez definen el núcleo son **ALPHA**, espacios de actividad y competencias (OMG, 2015).

Cada área contiene una o más **ALPHA**. Estas son representaciones de aspectos esenciales para trabajar proporcionando descripciones del tipo de cosas que un equipo administrará, producirá y utilizará en el proceso de desarrollo, mantenimiento y soporte. **ALPHA** es un acrónimo de un atributo de salud del progreso de nivel abstracto (Abstract-Level Progress Health Attribute). Un **ALPHA** se caracteriza por diferentes estados, cada estado representa un indicador de salud sobre la progresión del proyecto (figura 4.15). También se pueden verificar cuáles son las dificultades que impiden avanzar. Esto se garantiza por un conjunto de listas de verificación que describen los elementos que deben abordarse para llegar a un estado específico.

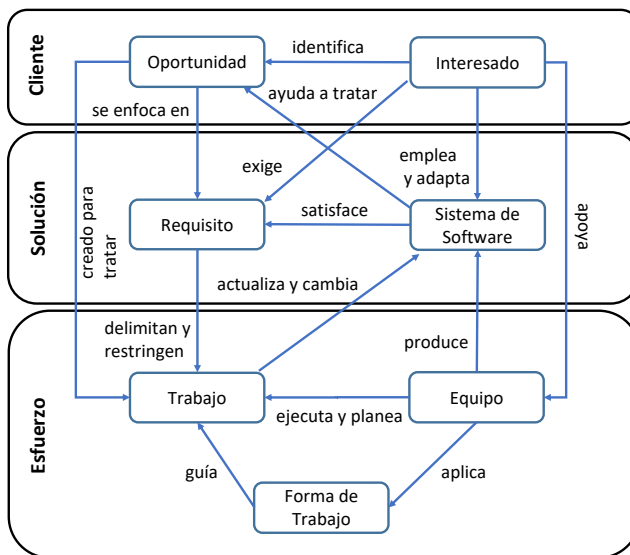


Figura 4.15. ALPHA del SEMAT

Fuente: elaboración propia con base en OMG (2015).

El núcleo ha definido elementos esenciales como trabajo, forma de trabajo, equipo, interesados, requisitos, sistema de software y oportunidad. El lenguaje ha definido los constructos principales como **ALPHA** que representan los elementos esenciales; actividades, espacios de actividades

que deben rellenarse con un conjunto de actividades que se llevarán a cabo en un esfuerzo de ingeniería de software específico; productos de trabajo y prácticas. Finalmente, la práctica ha aplicado el núcleo y el lenguaje al describir prácticas para evaluar su aplicabilidad, precisión, idoneidad, cobertura y facilidad de uso (Kajko-Mattsson *et al.*, 2012).

La figura 4.16 muestra las cosas que se deben hacer para los proyectos de desarrollo de software, es decir, los espacios de actividad. Los cuadros de diferentes colores representan las tres áreas de conocimiento: cliente, solución y esfuerzo. Las competencias representan las habilidades clave que se requieren para el desarrollo de software (figura 4.17).

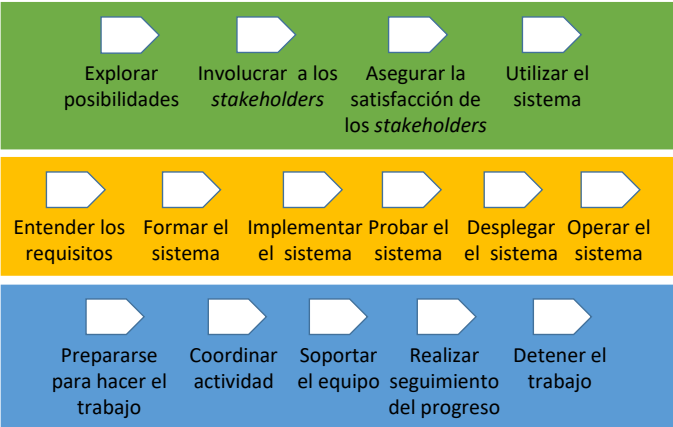


Figura 4.16. Espacios de actividad

Fuente: elaboración propia con base en oMG (2015).

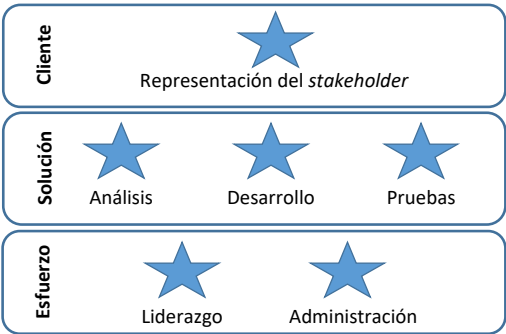


Figura 4.17. Competencias del núcleo

Fuente: elaboración propia con base en oMG (2015).

4.3.5. SWEBOK

El **SWEBOK** es una guía desarrollada y aprobada por la **IEEE** Computer Society y publicada originalmente en 2004. Esta guía tiene como propósito recolectar criterios a fin de establecer prácticas adecuadas para ejecutar un proceso de desarrollo de productos y soluciones software (Alarcón *et al.*, 2013).

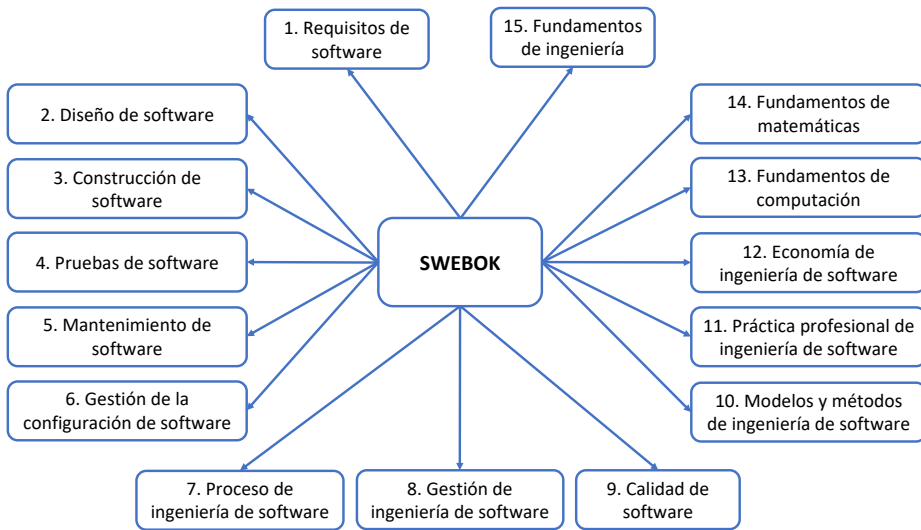


Figura 4.18. Áreas de conocimiento del SWEBOK

Fuente: elaboración propia con base en Bourque y Fairley (2014).

Muchos planes de estudios académicos y programas de capacitación industrial se basan en la guía **SWEBOK**, al igual que los programas de certificación, criterios de acreditación y criterios de licencias (Bourque y Fairley, 2014). La versión 3 es la última publicada.

Los objetivos fundamentales de la guía **SWEBOK** son los siguientes (Bourque y Fairley, 2014):

- ▶ Promover una visión coherente de la ingeniería de software en todo el mundo.
- ▶ Especificar el alcance de la ingeniería de software con respecto a otras disciplinas relacionadas.
- ▶ Caracterizar los contenidos de la disciplina de ingeniería de software.

- Proporcionar acceso al cuerpo de conocimiento de la ingeniería de software.

El **SWEBOK** versión 3 ha sido expandido para incluir quince áreas de conocimiento, que se presentan en la figura 4.18 y sus tópicos en la tabla 4.6.

Tabla 4.6. Áreas de conocimiento del **SWEBOK** junto con sus tópicos

Área de conocimiento	Tópicos
Requisitos de software	Fundamentos de requisitos, proceso de requisitos, obtención de requisitos, análisis de requisitos, especificación de requisitos, validación de requisitos, consideraciones prácticas y herramientas de requisitos de software.
Diseño de software	Fundamentos de diseño, cuestiones clave en diseño de software, estructura y arquitectura de software, diseño de interfaz de usuario, análisis y evaluación de la calidad de diseño de software, notaciones de diseño de software, estrategias y métodos de diseño de software y herramientas de diseño de software.
Construcción de software	Fundamentos de construcción de software, gestión de la construcción, consideraciones prácticas, tecnologías de construcción y herramientas de construcción de software.
Pruebas de software	Fundamentos de prueba de software, niveles de prueba, técnicas de prueba, medidas relacionadas con las pruebas, proceso de prueba y herramientas de prueba de software.
Mantenimiento de software	Fundamentos de mantenimiento de software, cuestiones clave en mantenimiento de software, proceso de mantenimiento, técnicas de mantenimiento y herramientas de mantenimiento de software.
Gestión de configuración de software	Gestión del proceso de gestión de configuración de software, identificación de la configuración del software, control de la configuración de software, contabilidad de estado de configuración de software, auditoría de configuración de software, administración y entrega de software y herramientas de gestión de configuración de software.
Proceso de ingeniería de software	Definición del proceso de software, ciclos de vida del software, evaluación y mejora de procesos de software, medición de software y herramientas de proceso de ingeniería de software.
Gestión de ingeniería de software	Iniciación y definición del alcance, planificación de proyectos de software, ejecución de proyectos de software, revisión y evaluación, cierre, medición de ingeniería de software y herramientas de gestión de ingeniería de software.
Calidad de software	Fundamentos de calidad de software, procesos de gestión de calidad de software, consideraciones prácticas y herramientas de calidad de software.

Área de conocimiento	Tópicos
Modelos y métodos de ingeniería de software	Modelado, tipos de modelos, análisis de modelos y métodos de ingeniería de software.
Práctica profesional de la ingeniería de software	Profesionalismo, dinámica de grupo y psicología y habilidades de comunicación.
Economía de la ingeniería de software	Fundamentos de la economía de ingeniería de software, economía del ciclo de vida, riesgo e incertidumbre, métodos de análisis económico y consideraciones prácticas.
Fundamentos de computación	Técnicas de resolución de problemas, abstracción, fundamentos de programación, conceptos básicos de lenguajes de programación, herramientas y técnicas de depuración, estructura y representación de datos, algoritmos y complejidad, concepto básico de sistemas, organización del computador, conceptos básicos de sistemas operativos, conceptos básicos del compilador, fundamentos de bases de datos.
Fundamentos de ingeniería	Conceptos básicos de comunicación en red, computación paralela y distribuida, factores humanos básicos del usuario, factores humanos básicos y desarrollo y mantenimiento de software seguro.
Fundamentos de matemáticas	Conjuntos, relaciones y funciones, lógica básica, técnicas de prueba, conceptos básicos de conteo, grafos y árboles, probabilidad discreta, máquinas de estado finito, gramáticas, precisión numérica, exactitud y errores, teoría de números y estructuras algebraicas.

Fuente: elaboración propia con base en Bourque y Fairley (2014).

Además de las quince áreas de conocimiento, el **SWEBOK** versión 3 enumera siete disciplinas relacionadas con la ingeniería de software: ingeniería de computación, ciencias de la computación, gestión de proyectos, gestión de la calidad, administración general, ingeniería de sistemas y matemáticas. Estas siete disciplinas relacionadas indican el alcance y los límites de la práctica de la ingeniería de software. El **SWEBOK** no describe el contenido de conocimiento de estas disciplinas relacionadas.

La guía **SWEBOK** utiliza una organización jerárquica para descomponer cada área de conocimiento en un conjunto de temas con etiquetas reconocibles. Una jerarquía de dos o tres niveles proporciona una forma razonable de encontrar temas de interés (Dupuis *et al.*, 2003).

► Desafíos

Con la revisión realizada en los apartados anteriores, se puede observar que los procesos de software actualmente han sido desarrollados y estructurados tanto por organizaciones orientadas a la estandarización como por empresas e industrias relevantes en el área del desarrollo de software. Sin embargo, la adopción y adaptación de estándares por empresas del sector de tecnología de la información (Π) aún posee numerosos desafíos para su implementación exitosa.

Generalmente, los usuarios tienen tareas no triviales de análisis del texto original del estándar y transcribirlo en herramientas y metodologías de soporte (Freericks, 2001). Los estándares son difíciles de asimilar debido a la dificultad de entender mejor el texto en lugar de enfocarse en aspectos de implementación de los aspectos tecnológicos (Mahonen, 2000).

Existen evidencias de que la mayoría de las pequeñas organizaciones de software no están adoptando estándares existentes, ya que los perciben como orientados hacia grandes organizaciones. Los estudios han demostrado que las percepciones negativas de las pequeñas empresas sobre los estándares de los modelos de procesos se deben principalmente a las opiniones negativas sobre el costo, la documentación y la burocracia. Se ha informado que las pymes tienen dificultades para relacionar los estándares con sus necesidades comerciales y para justificar la aplicación de los estándares internacionales en sus operaciones (Kuhrmann *et al.*, 2016).

► Ejemplos

Una versión más liviana y de libre acceso del proceso unificado es OpenUP. Es un proceso ágil que incorpora un contenido mínimo suficiente y fundamental. Sin embargo, es completo en el sentido de que puede manifestarse como un proceso para construir un sistema de software. OpenUP tiene las características esenciales de un proceso que aplica enfoques iterativos e incrementales en un ciclo de vida estructurado comprobado.

El proceso OpenUP se puede descargar desde su sitio web (OpenUP, 2022), el cual incluye un conjunto de páginas web para consultar sus fases de ciclo de vida, prácticas, roles, productos de trabajo, tareas, guías, herramientas y muchos elementos más.

El contenido de OpenUP aborda la organización del trabajo a nivel personal, de equipo y *stakeholders*, como se ve en la figura 4.19.

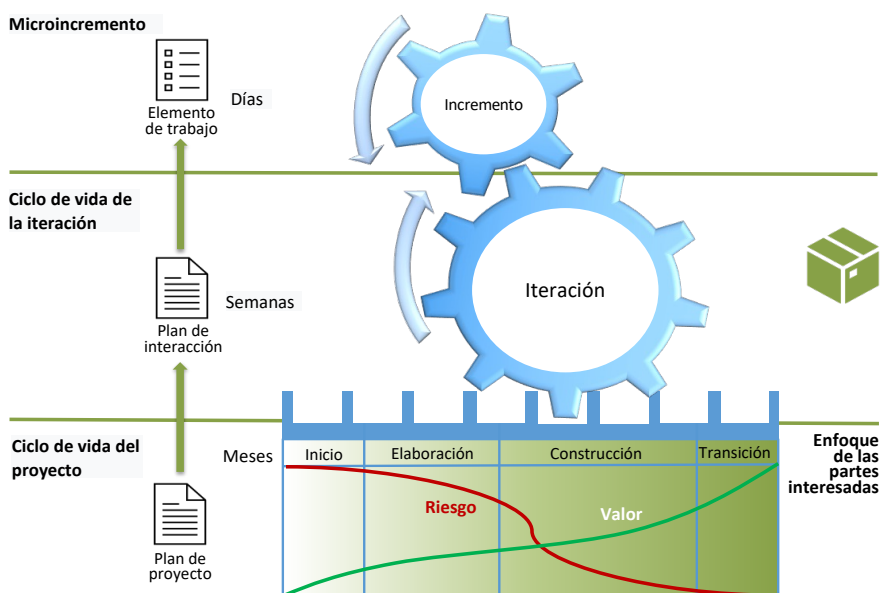


Figura 4.19. OpenUP: microincrementos, ciclo de vida de iteración y ciclo de vida del proyecto

Fuente: OpenUP (2022).

Los miembros del equipo en un proyecto OpenUP contribuyen con su trabajo en microincrementos, que representan el resultado de unas pocas horas a unos pocos días de trabajo. La aplicación evoluciona un microincremento a la vez y el progreso se ve efectivamente todos los días. El proyecto se divide en iteraciones: intervalos planificados en un marco de tiempo que normalmente se miden en semanas.

OpenUP estructura el ciclo de vida del proyecto en las cuatro fases del proceso unificado: inicio, elaboración, construcción y transición. Cada fase cuenta con un flujo de trabajo, como se observa en la figura 4.20.

A su vez, cada actividad del flujo de trabajo se descompone en tareas (figura 4.21).

Cuando se selecciona una tarea, se despliega su descripción correspondiente (figura 4.22).

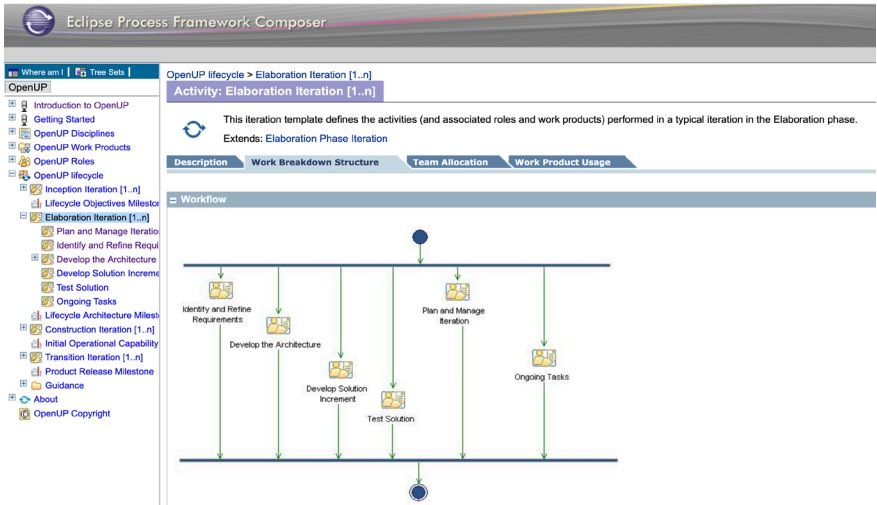


Figura 4.20. Flujo de trabajo de la fase de elaboración de OpenUP
Fuente: OpenUP (2022).

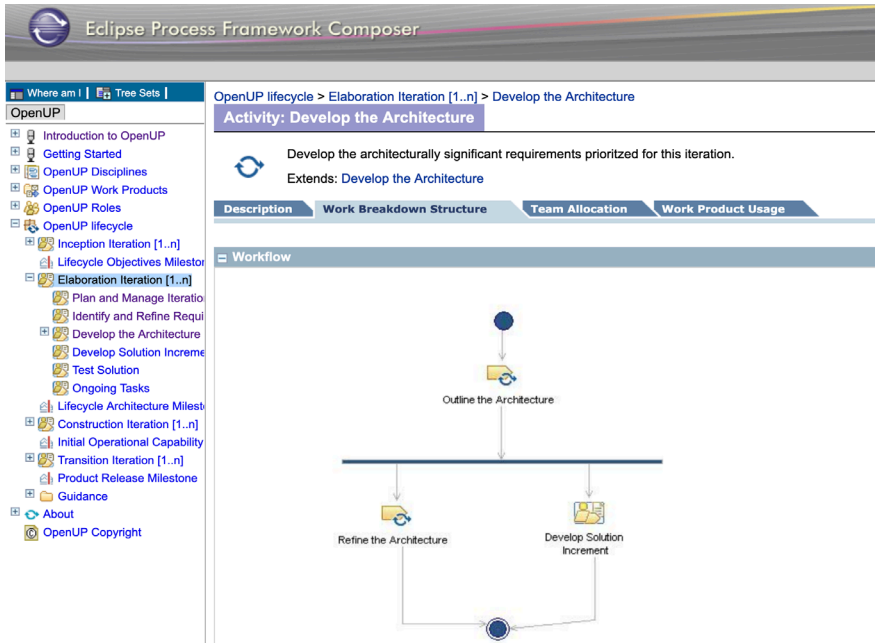


Figura 4.21. Flujo de tareas de la fase de elaboración de OpenUP
Fuente: OpenUP (2022).

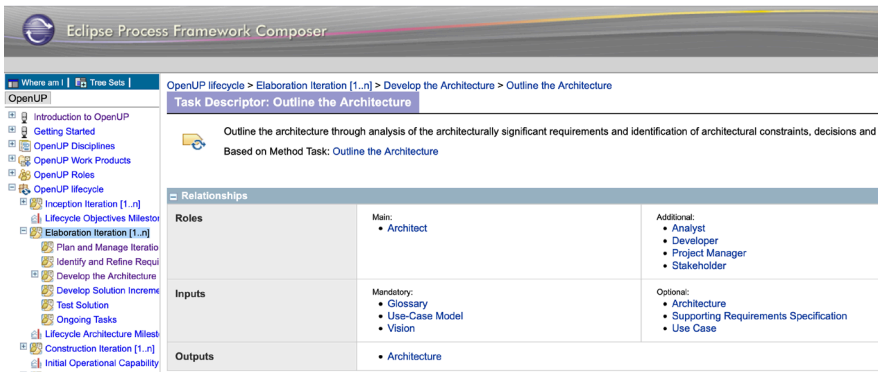


Figura 4.22. OpenUP: microincrementos, ciclo de vida de iteración y ciclo de vida del proyecto

Fuente: OpenUP (2022).

De esta manera, el equipo de desarrollo cuenta con una herramienta web donde puede consultar los elementos del proceso que necesita para desarrollar un proyecto de software con información bastante completa y elaborada.

Se recomienda descargar el proceso OpenUP desde la web y consultar los siguientes elementos del proceso:

- ▶ **Prácticas:** Se dividen en prácticas de gestión, técnicas y de despliegue. Cada práctica tiene asociada conceptos clave, productos de trabajo, tareas y guías.
- ▶ **Roles:** Se descomponen en roles básicos, de despliegue y de ambiente. Cada rol tiene asociado un flujo de trabajo con actividades y artefactos.
- ▶ **Productos de trabajo:** Se dividen en arquitectura, despliegue, desarrollo, ambiente, gestión del proyecto, requisitos y pruebas.
- ▶ **Guías:** Se descomponen en listas de verificación, conceptos, ejemplos, guías, prácticas, referencias, reportes, hojas de ruta y plantillas.

▶ Resumen

Las iniciativas de estándares están dirigidas a la definición de estándares, métodos de evaluación y métodos de mejora del proceso. Un estándar se define como un conjunto de criterios aprobados, documentados y disponi-

bles para ser utilizados como reglas, guías o definiciones de características que permiten determinar la adecuación de una acción o de un objeto a su propósito. Los estándares se pueden clasificar como *de iure* o *de facto*.

La norma **ISO 9000** está orientada al desarrollo de sistemas de gestión de la calidad. La norma **ISO 9001** adapta los principios de la norma general a empresas que desarrollan productos y servicios que cumplan con los requisitos de los clientes. Esta última norma se puede aplicar a empresas desarrolladoras de software; sin embargo, ha sido especializada para la producción de software en una norma aparte, la **ISO 9000-3**, ya que el desarrollo de software se considera un proceso de producción con un propósito general diferente. Cubre tanto actividades del ciclo de vida como de soporte.

La norma europea **ECSS** ofrece directrices para el desarrollo de software espacial y tres ramas principales: estándares de gestión, estándares de ingeniería y estándares de aseguramiento del producto.

La norma **ISO/IEC 12207** agrupa las actividades a ejecutarse durante el ciclo de vida en cinco procesos primarios, ocho procesos de soporte y cuatro procesos organizacionales.

La norma **DOD-STD-2167A** del Departamento de Defensa de Estados Unidos define las siguientes fases en el ciclo de desarrollo de software: análisis/diseño de requisitos del sistema, análisis de requisitos de software, diseño preliminar, diseño detallado, codificación y unidad de software de computador, pruebas e integración de componentes de software de computador, prueba de ítems de configuración de software de computador y pruebas e integración del sistema.

La norma **IEEE 1074** está organizada en grupos de actividades: gestión de proyectos, actividades de predesarrollo, desarrollo, posdesarrollo e integrales. La norma **ISO/IEC 15288** establece cuatro procesos: procesos de acuerdo, procesos empresariales, procesos del proyecto y procesos técnicos. Incluye también un proceso de adaptación.

La guía Métrica v.3 es una metodología española que descompone sus procesos en actividades y estas a su vez en tareas. Para cada tarea se describe su contenido haciendo referencia a sus principales acciones, productos, técnicas, prácticas y participantes. Los procesos principales son planificación de sistemas de información, estudio de viabilidad del sistema, análisis del sistema de información, diseño del sistema de información, construcción del sistema de información, implantación y aceptación del sistema, y mantenimiento de sistemas de información.

El proceso unificado es un conjunto de mejores prácticas y se caracteriza por ser guiado por casos de uso, centrado en la arquitectura y tener un ciclo de vida iterativo e incremental. Posee cuatro conceptos básicos: trabajador, actividad, artefacto y flujo de trabajo. Las actividades se realizan en un marco bidimensional basado en cuatro fases: inicio, elaboración, construcción y transición; y seis flujos de trabajo: modelado de negocios, requisitos, análisis y diseño, implementación, pruebas y despliegue.

El proceso EssUP es una variante del proceso unificado que captura prácticas “esenciales” basadas en el enfoque ágil. Contiene tres áreas de interés: cliente, solución y esfuerzo; junto con siete **ALPHA** que se deben gestionar y progresar: *stakeholders*, oportunidades, requisitos, sistema software, equipo, forma de trabajo y trabajo, así como 15 espacios de actividades que describen las acciones que se deben realizar.

El **SEMAT** se enfoca en encontrar un núcleo de elementos ampliamente acordados y definir una base teórica sólida. Sus elementos se basan en un conjunto de métodos compuestos de prácticas, los cuales se describen en un núcleo. Todos estos elementos están definidos en términos de un lenguaje.

El **SWEBOK** es una guía desarrollada por la **IEEE** Computer Society que proporciona acceso al cuerpo de conocimiento de la ingeniería de software, incluye 15 áreas de conocimiento y siete disciplinas relacionadas que se cruzan con la ingeniería de software.

► Ejercicios

1. ¿Para qué tipo de proyectos de software aplicaría el proceso unificado?
2. Explicar tres características del proceso unificado de desarrollo de software.
3. Analizar las cuatro fases del proceso unificado y su relación con las actividades de desarrollo, como el análisis de requisitos, el diseño y las pruebas.
4. Realizar un cuadro comparativo entre las normas **ISO 9000-1** e **ISO 9000-3**.
5. Consultar la normativa Métrica v.3 y responder a las siguientes preguntas:

- ¿Cuáles son las actividades del proceso de implantación y aceptación del sistema?
 - ¿Qué roles participan en todas las actividades del proceso de estudio de viabilidad del sistema?
 - ¿Qué modelo de proceso de software recomienda utilizar Métrica v.3?
 - ¿Cuáles son las actividades del proceso de construcción del sistema de información?
6. ¿Qué es y cuál es la utilidad de las **ALPHA** en el proceso del **SEMAT**?
 7. Si los riesgos son elementos esenciales en los proyectos de software, ¿por qué cree que no son un **ALPHA** en el proceso del **SEMAT**?
 8. Realizar un cuadro comparativo entre los estándares **ISO/IEC 12207** y **IEEE 1074**.
 9. Leer el área de conocimiento de “Proceso de ingeniería de software” del **SWEBOK** y realizar un breve resumen.
 10. ¿En qué se diferencia el proceso EssUP del proceso unificado en que se basa?

► Referencias

- Acuna, S. T., Juristo, N., Moreno, A. M. y Mon, A. (2006). *A software process model handbook for incorporating people's capabilities*. Springer.
- Alarcón, A., Martínez, N. y Sandoval, J. (2013). Use of learning strategies of **SWEBOK**® guide proposed knowledge areas. En L. Uden, F. Herrera, J. Bajo Pérez y J. Corchado Rodríguez (eds.), *7th International Conference on Knowledge Management in Organizations: Service and Cloud Computing. Advances in Intelligent Systems and Computing* (pp. 243-254). Springer. https://doi.org/10.1007/978-3-642-30867-3_22
- Bernardos, M.^a del S. (2004). Guideline for developing a software life cycle process in natural language generation projects. En A. Gelbukh (eds.), *Computational Linguistics and Intelligent Text Processing. CICLing 2004. Lecture Notes in Computer Science* (pp. 355-359). Springer. https://doi.org/10.1007/978-3-540-24630-5_43
- Booch, G., Rumbaugh, J. y Jacobson, I. (2017). *The unified modeling language user guide*. Addison-Wesley.
- Bourque, P. y Fairley, R. E. (eds.) (2014). *Guide to the software engineering body of knowledge, version 3.0*. **IEEE** Computer Society. http://artemisa.unicauca.edu.co/~cardila/IS_SWEBOKv3.pdf
- Ceccarelli, A. y Silva, N. (2013). Qualitative comparison of aerospace standards: An objective approach. En 2013 **IEEE** International Symposium on Software Reliability

- Engineering Workshops (ISSREW)* (pp. 331-336). **IEEE**. <https://doi.org/10.1109/ISSREW.2013.6688916>
- Dahhane, W., Berrich, J., Bouchentouf, T. y Rahmoun, M. (2016). **SEMAT** Essence's Kernel applied to O-MaSE. En *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)* (pp. 799-804). **IEEE**. <https://doi.org/10.1109/ICMCS.2016.7905565>
- David, P. A. (1995). Standardization policies for network technologies: The flux between freedom and order revisited. En R. Hawkins, R. Mansell y J. Skea (eds.), *Standards, innovation and competitiveness: The politics and economics of standards in natural and technical environments* (pp. 15-35). Edward Elgar.
- De Vries, H. J. (2013). *Standardization: A business approach to the role of national standardization organizations*. Springer.
- Derniame, J. C., Kaba, B. A. y Wastell, D. (eds.) (1999). *Software process: Principles, methodology, and technology*. Springer.
- Dupuis, R., Bourque, P. y Abran, A. (2003). **SWEBOK** guide an overview of trial usages in the field of education. En *Proceedings of the 33rd Annual Frontiers in Education (FIE 2003)*. **IEEE**. <https://doi.org/10.1109/FIE.2003.1265987>
- ECSS-E-ST-10-02c** - Verification (2009, 6 de marzo). <https://ecss.nl/standard/ecss-e-st-10-02c-verification/>
- ECSS-E-ST-10-06c** - Technical requirements specification. (2009, 6 de marzo). <https://ecss.nl/standard/ecss-e-st-10-06c-technical-requirements-specification/>
- ECSS-E-ST-10c** Rev.1 - System engineering general requirements. (2017, 15 de febrero). <https://ecss.nl/standard/ecss-e-st-10c-rev-1-system-engineering-general-requirements-15-february-2017/>
- ECSS-E-ST-40c** - Software (2009, 6 de marzo). <https://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/>
- ECSS-Q-ST-30c** Rev.1 - Dependability (2017, 15 de febrero). <https://ecss.nl/standard/ecss-q-st-30c-rev-1-space-product-assurance-dependability-15-february-2017/>
- ECSS-Q-ST-40c** - Safety. (2009, 6 de marzo). <https://ecss.nl/standard/ecss-q-st-40c-safety/>
- ECSS-Q-ST-80c** Rev.1 - Software product assurance. (2017, 15 de febrero). <https://ecss.nl/standard/ecss-q-st-80c-rev-1-software-product-assurance-15-february-2017/>
- Elvesæter, B., Striewe, M., McNeile, A. y Berre, A. J. (2012). Towards an agile foundation for the creation and enactment of software engineering methods: The **SEMAT** approach. En *Proceedings of the Co-located Events at the 8th European Conference on Modelling Foundations and Applications (ECMFA 2012)* (pp. 279-290). Technical University of Denmark. https://www.dcs.bbk.ac.uk/~amcnei01/docs/be_pmde_2012_paper.pdf
- Emmerich, W. (1999). Software process: Standards, assessments and improvement. En J. C. Derniame, B. A. Kaba y D. Wastell (eds.), *Software Process: Principles, Methodology, and Technology. Lecture Notes in Computer Science* (pp. 15-25). Springer. https://doi.org/10.1007/3-540-49205-4_2
- ESA Board for Software Standardisation and Control. (1995). *Guide to the Software Engineering Standards*. http://everyspec.com/ESA/ESA_PSS-05-06_ISSUE-1_REVISION-1_10567/
- Freericks, C. (2001). Open source standards on software process: A practical application. **IEEE Communications Magazine**, 39(4), 116-123. <https://doi.org/10.1109/35.917513>

- Halling, M., Zuser, W., Kohle, M. y Biffl, S. (2002). Teaching the unified process to undergraduate students. En *Proceedings 15th Conference on Software Engineering Education and Training (CSEET 2002)* (pp. 148-159). **IEEE**. <https://doi.org/10.1109/CSEE.2002.995207>
- Hui, Y., Yan, Y., Quanyu, W. y Zhiwen, C. (2015). Compare essential unified process (Es-sUP) with rational unified process (RUP). En *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)* (pp. 472-476). **IEEE**. <https://doi.org/10.1109/IC-IEA.2015.7334159>
- Ida, T. (2017). Evolutionary stability of de jure and de facto standards. *Working Paper*. **IEEE-STD** 1074. (2006). **IEEE Standard for Developing a Software Project Life Cycle Process, IEEE Std 1074-2006 (Revision of IEEE Std 1074-1997)**. <https://doi.org/10.1109/IEEESTD.2006.219190>
- ISO/IEC** 15288:2015. (2015). *Systems engineering - System life cycle processes*. International Standardization Organization.
- ISO/IEC** 9000-3:2004. (2004). *Software engineering - Guidelines for the application of ISO 9001:2000 to computer software*. International Organization for Standardization.
- ISO/IEC/IEEE** 12207:2017. (2017). *Systems and software engineering - Software life cycle processes*. International Organization for Standardization.
- Ivar Jacobson International. (2015). *How to use the Agile Essentials Practice Pack*. <https://www.ivarjacobson.com/services/agile-essentials-starter-pack-agile-practices>
- Jacobson, I., Ng, P. W., McMahon, P. E., Spence, I. y Lidman, S. (2012). The essence of software engineering: The **SEMAT** kernel. *Communications of the ACM*, 55(12), 42-49. <http://doi.acm.org/10.1145/2380656.2380670>
- Jones, M., Mortensen, U. K. y Fairclough, J. (1997). The **ESA** software engineering standards: Past, present and future. En *Proceedings of IEEE International Symposium on Software Engineering Standards* (pp. 119-126). **IEEE**. <https://doi.org/10.1109/SESS.1997.595952>
- Kajko-Mattsson, M., Striwe, M., Goedicke, M., Jacobson, I., Spence, I., Huang, S. ... y Seymour, E. (2012). Refounding software engineering: The SEMAT initiative (Invited presentation). En *2012 34th International Conference on Software Engineering (ICSE)* (pp. 1649-1650). **IEEE**. <https://doi.org/10.1109/ICSE.2012.6227214>
- Kempton, S., Sobell, C. y Withrow, C. (1988). **DOD-STD-2167A** applied to software maintenance. En *1988 Conference on Software Maintenance* (pp. 159-164). **IEEE Computer Society**. <https://doi.ieeecomputersociety.org/10.1109/ICSM.1988.10156>
- Krishnan, M. S., Mukhopadhyay, T. y Zubrow, D. (1999). Software process models and project performance. *Information Systems Frontiers*, 1, 267-277. <https://doi.org/10.1023/A:1010054412650>
- Kuhrmann, M., Münch, J., Richardson, I., Rausch, A. y Zhang, H. (eds.) (2016). *Managing software process evolution: Traditional, agile and beyond—how to handle process change*. Springer. <https://doi.org/10.1007/978-3-319-31545-4>
- Land, S. K. y Walz, J. W. (2007). *Practical support for ISO 9001 Software Project Documentation using IEEE Software Engineering Standards*. Wiley-**IEEE** Press.
- Land, S. K., Smith, D. B. y Walz, J. W. (2012). *Practical support for lean six sigma software process definition: Using IEEE software engineering standards*. John Wiley & Sons.

- Mahonen, P. (2000). The standardization process in it-too slow or too fast? En *Information technology standards and standardization: A global perspective* (pp. 35-47). IGI Global. <http://dx.doi.org/10.4018/978-1-878289-70-4.ch003>
- McCord, J. W. (1990). Software development-process and implementation: **DOD-STD-2167A** vs. traditional methodologies. En *IEEE Conference on Aerospace and Electronics* (pp. 681-687). IEEE. <https://doi.org/10.1109/NAECON.1990.112848>
- Métrica v.3. (2020). *Metodología de planificación, desarrollo y mantenimiento de sistemas de información*. https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html
- Moore, J. W. (2006). *The road map to software engineering: A standards-based guide*. Wiley-IEEE Computer Society Press.
- Object Management Group. (2015). *Essence - Kernel and Language for Software Engineering Methods. Version 1.1*. <https://www.omg.org/spec/Essence/1.0/PDF>
- OpenUP. (2022). *Proceso unificado abierto*. <https://www.utm.mx/~caff/doc/OpenUPWeb/>
- Pino, F. J., Baldassarre, M. T., Piattini, M., Visaggio, G. y Caivano, D. (2010). Mapping software acquisition practices from **ISO 12207** and **CMMI**. En L. A. Maciaszek, C. González-Pérez y S. Jablonski (eds.), *Evaluation of Novel Approaches to Software Engineering*. **ENASE ENASE 2009 2008**. Communications in Computer and Information Science (pp. 234-247). Springer. https://doi.org/10.1007/978-3-642-14819-4_17
- Pons, C., Giandini, R. y Baum, G. (2000). Dependency relations between models in the Unified Process. En *Tenth International Workshop on Software Specification and Design*. **IWSSD-10 2000** (pp. 149-157). IEEE. <https://doi.org/10.1109/IWSSD.2000.891136>
- Portuguese Institute of Quality. (2008). **NP EN ISO 9001:2008 - Quality Management Systems - Requirements**.
- Priestley, M. y Utt, M. H. (2000). A unified process for software and documentation development. En *18th Annual Conference on Computer Documentation*. **IPCC SIGDOC 2000**. Technology and Teamwork. Proceedings. IEEE Professional Communication Society International Professional Communication Conference (pp. 221-238). IEEE. <https://doi.org/10.1109/IPCC.2000.887279>
- Sommerville, I. (1996). Software process models. **ACM Computing Surveys (CSUR)**, 28(1), 269-271. <https://dl.acm.org/doi/pdf/10.1145/234313.234420>
- Strandberg, T. (2016). What is **ISO/IEC 15288**? (A concise introduction). *White Paper*.
- Valdés Cárdenas, L. E. (2005). *Guía para la implementación de la Norma ISO 9001:2000 en las empresas de software*. Colciencias.
- West, J. (2003). The role of standards in the creation and use of information systems. En *Proceedings of the Workshop on Standard Making: A Critical Research Frontier for Information Systems* (pp. 314-326). MIS Quarterly.



► Parte II. Ciclo de vida del proceso de software

Para una implementación efectiva de un enfoque orientado al proceso en las empresas de desarrollo de software, el proceso de software debe pasar por una serie de fases que conforman un ciclo de vida. El objetivo de esta segunda parte es comprender las diferentes fases que constituyen el ciclo de vida de un proceso de software. El proceso se debe definir utilizando un lenguaje de modelado de procesos. Luego, a medida que este se ejecute siguiendo su definición, se recolectarán métricas que midan su uso real y efectivo. Las métricas permitirán, por un lado, controlar el proceso de tal manera que, si se obtienen variaciones significativas durante su ejecución, se tomen las acciones correctivas adecuadas, y por otro, las métricas permitirán evaluar el proceso y proponer estrategias de mejora que conllevarán una redefinición de este reiniciando el ciclo de vida. Este ciclo de vida brinda un análisis estratégico muy similar al de mejoramiento de procesos industriales en control. Esta segunda parte está compuesta por los siguientes capítulos.

► Capítulo 5. Lenguajes de modelado de procesos de software

El proceso de software se debe modelar utilizando algún tipo de notación gráfica estandarizada que permita entender sus diferentes elementos incorporando diversos niveles de abstracción. Este capítulo presenta los principales conceptos sobre modelado para luego profundizar en modelado de procesos de software. Luego, se describirán los principales lenguajes de modelado de procesos de software con énfasis en sus elementos notacionales y presentando ejemplos de fases específicas de procesos de software modeladas con estos lenguajes. El capítulo finaliza con una comparación

entre los lenguajes de modelado de procesos de software presentados de acuerdo con varios criterios y aspectos prácticos de implementación.

► Capítulo 6. Métricas de proceso de software

Para entender cómo realmente un proceso de software se está ejecutando (conocer su estado actual) y para evaluar si se están cumpliendo sus objetivos, se deben recolectar métricas, las cuales son medidas cuantitativas que deben ser interpretadas para evaluar si los procesos se están realizando correctamente. Para ello, el capítulo describe conceptos generales sobre medición; luego, su aplicación al software y, en concreto, al proceso de software (utilizando una ontología). También se describirá un estándar de medición del proceso de software, en específico, la norma **ISO/IEC/IEEE 15939**. Además, presenta un conjunto específico de métricas del proceso de software basado en entidades y atributos medibles.

► Capítulo 7. Control del proceso de software

Este capítulo aborda el control del proceso utilizando técnicas estadísticas. El proceso de software definido, a medida que se ejecute, puede tener desviaciones significativas que hacen que no se cumplan los objetivos previamente definidos. Por ello, estas deben ser detectadas y responder prontamente para que el proceso se considere nuevamente bajo control, permaneciendo dentro de rangos de límites aceptables. El capítulo describe una colección de gráficos de control tomados del dominio de la gestión de la calidad de procesos, pero adaptados al dominio del proceso de software. Se presentarán varios gráficos de control para variables y atributos. Además, la técnica de seguimiento binario y la de valor ganado para realizar el seguimiento de proyectos.

► Capítulo 8. Mejora del proceso de software

Este capítulo se ocupa de la mejora del proceso de software. Explica en qué consiste y cómo se debe implementar una estrategia de mejora del proceso. Para ello, se describen los principales conceptos sobre la mejora del proceso de software y se mostrará un ciclo genérico de mejora continua.

Luego, se explica una colección de diferentes modelos de mejora de procesos, cada uno con sus características, elementos y recomendaciones sobre su uso. También se presentan metodologías de evaluación del proceso que permiten comprender la madurez actual del proceso en las organizaciones. Por último, se describen diversas herramientas que se pueden aplicar con éxito para implementar las estrategias de mejora del proceso.

Lenguajes de modelado de procesos de software

► Objetivos

Los objetivos de este capítulo están enfocados en describir diversos lenguajes de modelado de procesos de software. Al estudiar este capítulo, el lector:

- Conocerá los principales conceptos sobre el modelado.
- Aprenderá qué es un modelo de proceso de software.
- Entenderá qué es un lenguaje de modelado de procesos y conocerá la notación gráfica de los principales lenguajes de modelado de proceso utilizados en la industria.

► 5.1. Conceptos sobre modelos

Los modelos son un componente fundamental de la ciencia y la ingeniería. Se utilizan en diversas disciplinas para probar evidencias universales y encontrar información más detallada sobre sus especulaciones. Frecuentemente, los modelos marcan un paso intermedio en el desarrollo de artefactos, como puentes, automóviles y teléfonos móviles. En ingeniería de software, la importancia de los modelos radica en que abarcan diversos productos durante todo el proceso de desarrollo desde la representación de pasos intermedios hasta los puntos finales del trabajo. Los mo-

delos son una especificación, pero también un programa se considera un modelo (Ludewig y Lichter, 2023).

Un modelo es un conjunto de declaraciones sobre un sistema en estudio (Seidewitz, 2003). Las declaraciones se refieren a expresiones del sistema que pueden ser verdaderas o falsas. El modelo se utiliza para describir este sistema y será correcto si todas las declaraciones son verdaderas. El objetivo principal de un modelo es la comunicación de ideas y conceptos (Moro, 2004). Los modelos ayudan a gestionar la complejidad de procesos y productos.

Hay muchas razones para modelar un proceso. El primero es simplemente comprender el proceso y compartir su comprensión con las personas involucradas diariamente en el proceso (Génova, 2012).

Los modelos pueden tener las siguientes propiedades generales que comparten entre sí (Broy y Rumpe, 2007):

- Un propósito (o varios propósitos).
- Una referencia al original.
- Abstracción de ciertas cualidades del original, de tal manera que se logra una simplificación de la realidad suprimiendo detalles irrelevantes del sistema en estudio.
- Una característica pragmática en la que en ciertas condiciones o problemas los modelos pueden complementar el original.

En conjunto, se pueden distinguir tres tipos de propiedades de un modelo (Ludewig y Lichter, 2023):

- *Propiedades esenciales*: Son las propiedades del original consideradas en el modelo. Por tanto, los modelos son simplificaciones del sistema en estudio.
- *Propiedades predeterminadas*: Son propiedades del original que no se consideran en el modelo.
- *Propiedades abundantes*: Son propiedades en el modelo no presentes en el original, lo cual conlleva el riesgo de llegar a conclusiones erróneas con respecto al original.

La figura 5.1 muestra las conexiones entre el original y su modelo a partir de sus propiedades esenciales, predeterminadas y abundantes.

Generalmente, los modelos se clasifican en dos tipos (Ludewig y Lichter, 2023; Moro, 2004):

- *Modelos prescriptivos*: Describen conexiones o sistemas existentes.

- *Modelos descriptivos*: Se pueden considerar manuales para la construcción de sistemas.

En cuanto al desarrollo de software, los modelos tienen algunas características especiales. En procesos de desarrollo más complejos, se deben considerar al menos dos tipos de modelos (Brcina, 2007; DeMarco, 2004):

- Un modelo para el sistema objetivo.
- Un modelo para los procesos del proyecto de desarrollo.

Este capítulo se centrará principalmente en el segundo tipo de modelos con énfasis en modelos de procesos de software.

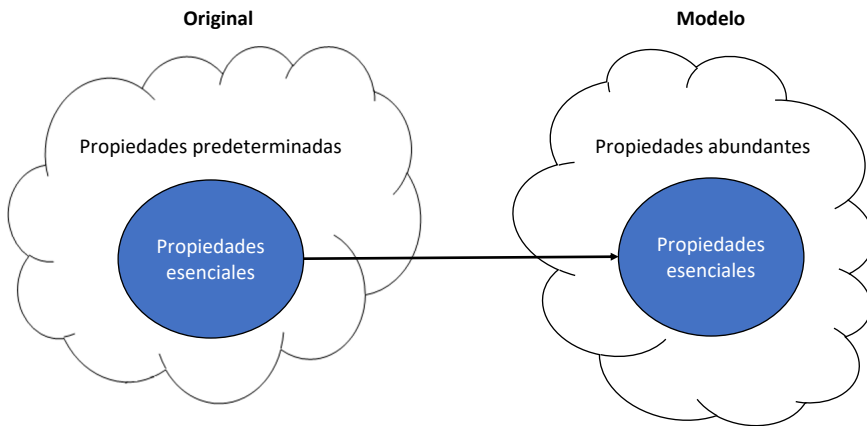


Figura 5.1. Propiedades del original y del modelo

Fuente: elaboración propia con base en Ludewig y Lichter (2023).

► 5.2. Modelos de procesos de software

Las empresas desarrolladoras de software fabrican productos de software a través de diferentes proyectos que tienden a seguir patrones comunes. Estos deben ser capturados en una representación del proceso. Un modelo de proceso es un tipo especial de representación de este, ya que se expresa en un formalismo adecuado.

Por tanto, el modelado de procesos ayuda a comprender mejor el proceso y a identificar y prevenir problemas. Este paso hacia una comprensión

profunda es el requisito previo para llevar a cabo el análisis, el rediseño o la automatización del proceso.

Lo que constituye un buen modelo de proceso depende del propósito para el cual se va a utilizar. Es importante que el modelo capture la mayoría de los aspectos del proceso que son relevantes para el propósito para el cual se va a utilizar el modelo. Los modelos del proceso de software se han utilizado para una amplia variedad de propósitos, como (Hunter y Thayer, 2001):

- Diseñar entornos de desarrollo de software.
- Estudiar la evolución y el mantenimiento del software.
- Simplificar y mejorar el proceso de software.
- Identificar partes críticas del proceso de software.
- Diseñar herramientas para apoyar el proceso de software.

Los elementos básicos de un modelo de proceso de software se presentan en la figura 5.2.

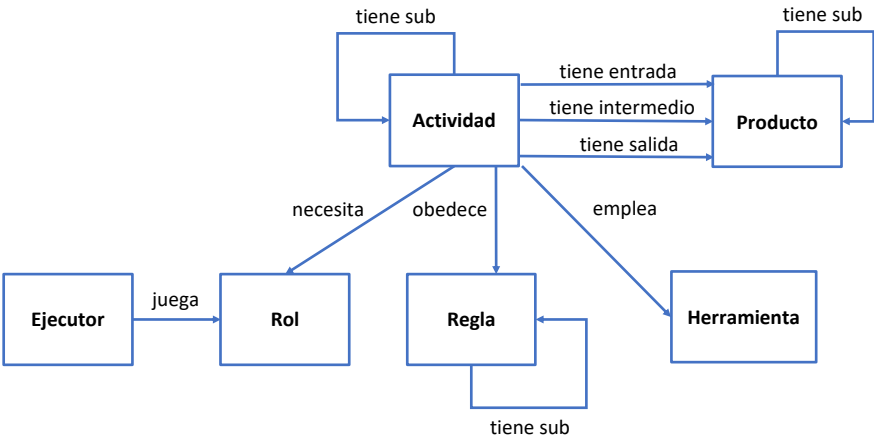


Figura 5.2. Conceptos básicos del modelado de procesos de software

Fuente: elaboración propia con base en Ludewig y Lichter (2023).

- *Actividades*: Operaciones atómicas o compuestas, o pasos de un proceso. Se enfocan en generar o modificar un conjunto de artefactos.
- *Productos*: Son los artefactos que se desarrollarán, entregarán y mantendrán en un proyecto. Los productos se descomponen en subproductos. Las actividades requieren productos de entrada para su ejecución, pueden tener productos intermedios y tener como

resultado productos de salida. Generalmente, hay cierta correspondencia entre la descomposición de las actividades en subactividades y la del producto en subproductos, pero no es obligatorio.

- ▶ *Ejecutores y herramientas*: Son recursos que se consideran activos que necesitan una actividad para que se lleve a cabo. Los ejecutores son agentes humanos del proceso. Las herramientas son agentes computarizados que se utilizan en el desarrollo de software.
- ▶ *Roles*: Son conjuntos de responsabilidades, obligaciones y habilidades. Los ejecutores y las herramientas están relacionados con una actividad a través de sus roles.
- ▶ *Reglas*: El carácter de la organización impacta el proceso indirectamente a través de reglas (políticas, direccionamientos y procedimientos) que rigen las actividades. Las reglas generalmente se incorporan en manuales y, por tanto, están estructuradas.

Los modelos de procesos pueden tener diferentes niveles de detalle capturando familias más pequeñas de subprocesos que corresponden a diversos niveles de interés (Derniame *et al.*, 1999):

- ▶ *Ciclos de vida*: Son representaciones informales de un proceso de software. Su propósito es abordar problemas metodológicos globales. Estas representaciones se aplican en muchas empresas y dominios de aplicaciones. El modelo en cascada o el modelo espiral se pueden considerar ciclos de vida.
- ▶ *Modelos de procesos genéricos*: Son representaciones abstractas que se pueden usar en muchos proyectos y organizaciones similares que comparten propiedades y características comunes. La figura 5.2 se puede considerar un modelo de proceso genérico.
- ▶ *Modelos de procesos personalizados*: Son más detallados y generalmente se derivan de un modelo genérico en atención a características locales específicas, por ejemplo, el dominio de la aplicación. Ejemplos de este tipo de modelos son los procesos estándar, la personalización de un estándar para adaptarse a las necesidades de una organización específica o a un proyecto específico de la organización.
- ▶ *Modelos de procesos establecidos*: Son el nivel de personalización más detallado. Definen el proceso a seguir en un proyecto específico y están listos para ser cargados en una máquina con el fin de proporcionar soporte automatizado a las personas involucradas en el proceso.

Según Dowson y Fernström (1994), se debe distinguir el dominio de ejecución e implementación del proceso. El primero se refiere al acto de participación de personas y herramientas en un proceso. El segundo está relacionado con el acto de conducir automáticamente el proceso, es decir, interpretar el modelo de proceso más detallado.

Los modelos de proceso frecuentemente no pretenden transmitir una comprensión de todo el proceso, sino solo una visión particular de interés, como (Derniame *et al.*, 1999):

- *Modelos de actividad*: Se centran en los tipos, la estructura y las propiedades de las actividades en el proceso y sus relaciones.
- *Modelos de producto*: Describen los tipos, la estructura y las propiedades de los elementos de software de un proceso (artefactos).
- *Modelos de recursos*: Describen los recursos necesarios o suministrados al proceso, relevantes desde un punto de vista gerencial.
- *Modelos de roles*: Describen un conjunto particular de recursos con las habilidades que tienen y las responsabilidades que aceptan.

► 5.3. Lenguajes de modelado de procesos de software

Un modelo de proceso de software se expresa en un determinado lenguaje de modelado. Si el lenguaje es gráfico, expresará el modelo mediante símbolos gráficos en forma de diagramas (Génova, 2012).

La definición de un lenguaje de modelado requiere una sintaxis y una semántica. Un lenguaje consiste en una notación sintáctica (sintaxis), que es un conjunto posiblemente infinito de elementos legales, junto con el significado de esos elementos que se expresa al relacionar la sintaxis con un dominio semántico. Por tanto, cualquier definición de lenguaje debe consistir en el dominio semántico de la sintaxis y el mapeo semántico desde los elementos sintácticos hasta el dominio semántico (Harel y Rumpe, 2004).

La semántica de un modelo se refiere a su significado. Debe proporcionar el significado de cada expresión, y este debe ser un elemento en un dominio bien definido y bien entendido. La semántica tiene dos aspectos (Génova, 2012):

- *Correspondencia semántica, interpretación*: Correspondencia entre cada símbolo del modelo y la entidad que representa del sistema en estudio.
- *Teoría deductiva, derivación*: Relación del modelo con otros modelos derivables a partir de él utilizando determinadas reglas deductivas.

La sintaxis se refiere a la estructura del modelo. En lenguajes gráficos, se pueden tener conjuntos distintos de símbolos que representan el mismo modelo. La sintaxis concreta o notación se refiere al conjunto de símbolos gráficos utilizados en el lenguaje de modelado. La sintaxis tiene dos aspectos (Génova, 2012):

- ▶ *Sintaxis abstracta, estructura lógica del lenguaje*: Reglas que especifican las expresiones bien formadas de símbolos.
- ▶ *Sintaxis concreta, aspecto visual*: Conjunto de símbolos gráficos utilizados para dibujar los diagramas.

Una definición sólida de un lenguaje de modelado debe relacionar las expresiones sintácticas con los elementos del dominio semántico, para que cada parte sintáctica se corresponda con su significado. En particular, al definir la semántica de un lenguaje, es esencial asociar explícita y claramente cada operador sintáctico con su significado como operador sobre el dominio semántico (Harel y Rumpe, 2004).

En la literatura, se pueden encontrar por lo menos 70 lenguajes de modelado de procesos (Pawel, 2010). Algunos de ellos están basados en reglas, por ejemplo, **MARVEL** (Kaiser *et al.*, 1990); otros en redes de Petri, por ejemplo, Software and Systems Process Engineering Metamodel (**SPEM**) (Bandinelli *et al.*, 1993), o en lenguajes de programación, por ejemplo, Satellite Procedure Execution Language and Library (**SPELL**) (Conradi *et al.*, 1992) y Ada Process Programming Language with Aspen (**APPL/A**) (Sutton *et al.*, 1995). Aunque estos lenguajes de primera generación eran ejecutables y ponían un fuerte énfasis en la formalidad, no obtuvieron mucha atención de la industria (Nitto *et al.*, 2002).

La continua proliferación de estos lenguajes de modelado de procesos ha planteado la necesidad de estandarizar las descripciones de los procesos de software. Este capítulo se centrará básicamente en cuatro de ellos, ampliamente utilizados en el dominio de procesos de software: **SPEM**, Business Process Model and Notation (**BPMN**), diagramas de actividad Unified Modeling Language (**UML**), Integration Definition for Function Modeling (**IDF0**), Electronic Product Code (**EPC**) y redes de Petri.

5.3.1. Software and Systems Process Engineering Metamodel

SPEM 2.0 es el estándar del Object Management Group (**OMG**) para modelado de procesos de sistemas y software (**OMG**, 2008a). La versión 2.0 fue publicada en 2008. **SPEM** se define como un metamodelo basado en

Meta-Object Facility (**MOF**) (OMG, 2008b). **SPEM** proporciona los conceptos para modelar, documentar, presentar, publicar, gestionar, intercambiar y realizar métodos y procesos de software. Este metamodelo funciona como una plantilla para la creación de modelos de procesos concretos. Por tanto, **SPEM** es un metamodelo de nivel M2 por **MOF**.

El metamodelo **SPEM** se basa en la idea de que un proceso de desarrollo de software es una colaboración entre roles que realizan actividades para desarrollar productos de trabajo. Su modelo conceptual está dado por la figura 5.3. Estos conceptos se pueden usar para construir una estructura de proceso personalizada. Se pueden usar conjuntos seleccionados de estos conceptos para crear instancias de un modelo de proceso, y así administrar y realizar las fases de producción de un sistema de software.

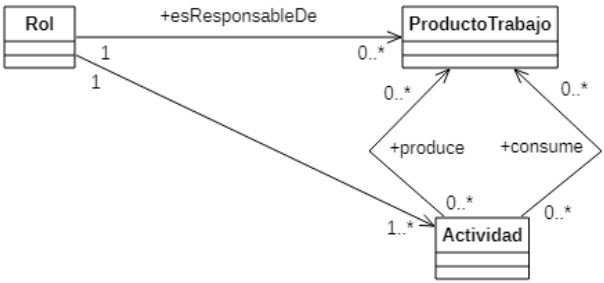


Figura 5.3. Modelo conceptual de SPEM

Fuente: elaboración propia con base en Hurtado Alegría *et al.* (2011) desarrollado con la herramienta StarUML.

El metamodelo **SPEM** 2.0 se compone de siete paquetes principales (Gallina *et al.*, 2014; Pereira *et al.*, 2012) que se ilustran en la figura 5.4:

- *Paquete core*: Define conceptos que son la base de los otros paquetes.
- *Paquete contenido de método (method content)*: Define conceptos que permiten especificar una base de conocimiento de elementos de proceso reutilizables e independientes de cualquier proceso específico.
- *Paquete de estructura de proceso (process structure)*: Define conceptos que permiten la representación de modelos de proceso compuestos por actividades interrelacionadas, roles y productos de trabajo.
- *Paquete de contenido gestionado (managed content)*: Define conceptos que permiten agregar descripciones en lenguaje natural de un proceso de software.

- *Paquete proceso con método (process with method)*: Define conceptos que integran los elementos del paquete estructura de proceso con contenido de método.
- *Paquete método plugin (plugin method)*: Define mecanismos que permiten la reutilización y la administración del contenido de método y procesos.
- *Paquete de comportamiento del proceso (process behavior)*: Define mecanismos y conceptos que permiten extender la estructura del proceso con especificación del comportamiento.

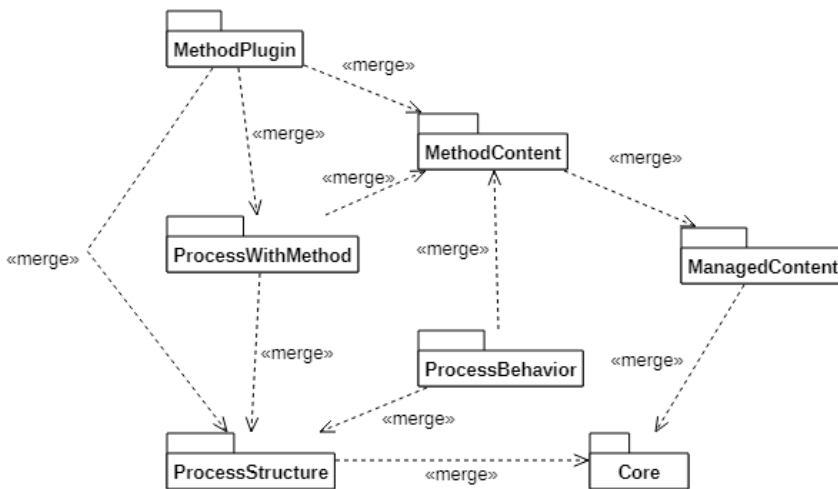


Figura 5.4. Estructura de SPEM 2.0

Fuente: elaboración propia con base en OMG (2008a) desarrollado con la herramienta StarUML.

La figura 5.5 presenta un marco conceptual para el uso de una implementación específica de **SPEM 2.0** (OMG, 2008a):

- *Representación estandarizada y bibliotecas de “contenidos de método” reutilizables*: **SPEM** tiene como objetivo ayudar a los profesionales del desarrollo a establecer una base de conocimiento de capital intelectual para el desarrollo de software que les permitirá administrar e implementar su contenido utilizando un formato estandarizado.
- *Soporte al desarrollo, la gestión y el crecimiento de los procesos de desarrollo*: Los equipos de desarrollo deben definir cómo aplicar sus métodos de desarrollo y mejores prácticas durante el ciclo de vida

del proyecto. **SPEM** proporciona la base conceptual para que ingenieros de procesos y gerentes de proyectos ensamblen procesos para sus proyectos de desarrollo concretos.

- *Establecimiento de un marco de trabajo personalizado a las necesidades del proyecto:* Ningún proyecto de desarrollo es exactamente igual a otro y el mismo proceso de desarrollo nunca se ejecuta dos veces. **SPEM** ofrece conceptos que proporcionan capacidades para la reutilización de procesos o patrones de procesos, la variabilidad de modelado y la adaptación que permiten a los usuarios definir sus propias extensiones, omisiones y puntos de variabilidad en procesos estándares reutilizados.
- *Creación de plantillas de plan de proyecto para la ejecución de procesos en el contexto de proyectos de desarrollo:* Una definición de proceso solo proporciona valor si impacta y orienta el comportamiento de los equipos de desarrollo. **SPEM** proporciona estructuras de definición de procesos que permiten a los ingenieros de proceso expresar cómo se debe implementar un proceso dentro de estos sistemas.



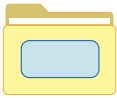
Figura 5.5. Framework de uso de SPEM 2.0

Fuente: elaboración propia con base en OMG (2008a).

En la tabla 5.1, se presenta un subconjunto de elementos gráficos de un modelo **SPEM** definidos en su metamodelo. Las tareas, los roles y productos de trabajo se consideran comúnmente elementos básicos del proceso (Bendraou *et al.*, 2010).

Tabla 5.1. Elementos gráficos de SPEM 2.0

Ícono	Estereotipo	Descripción
	Actividad	Representa unidades generales de trabajo asignables dentro de un proceso que son asignables a ejecutores específicos representados por un rol.
	Tarea	Representa una definición concreta sobre cómo el trabajo es ejecutado por los roles. Las tareas tienen asociados productos de trabajo de entrada y salida.
	Paso	Representa los pasos necesarios para realizar una tarea.
	Producto de trabajo	Representa un artefacto de entrada o salida (usado, modificado o producido) relacionado con una actividad o tarea.
	Rol	Conjunto de perfiles, competencias y responsabilidades de un individuo o conjunto de ellos que lleva a cabo una tarea o actividad dentro del proceso.
	Conjunto de roles	Representa una jerarquía anidada de equipos o miembros de equipos. Los equipos tienen diferentes habilidades y recursos para realizar diferentes tipos de trabajo.
	Métrica	Representa una o más restricciones que proporcionan mediciones estándar de cualquier elemento del proceso.
	Hito	Representa un evento significativo para un proyecto de desarrollo.

Ícono	Estereotipo	Descripción
	Herramienta	Describe las capacidades de una herramienta Computer-Aided Software Engineering (CASE) de propósito general o cualquier otra unidad de automatización que admita instancias asociadas a los roles al realizar el trabajo definido por una tarea.
	Categoría	Utilizada para categorizar el contenido según los criterios del usuario, así como para definir estructuras de árbol completas de categorías anidadas, y así navegar de manera sistemática por el contenido de los procesos basados en estas categorías.

Fuente: elaboración propia con base en omg (2008a).

En las figuras 5.6-5.9, se presenta un ejemplo de modelo de proceso de software elaborado con la notación de **SPEM**. El modelo representa las fases del proceso unificado abierto (Open Unified Process [OpenUP]) (2012).

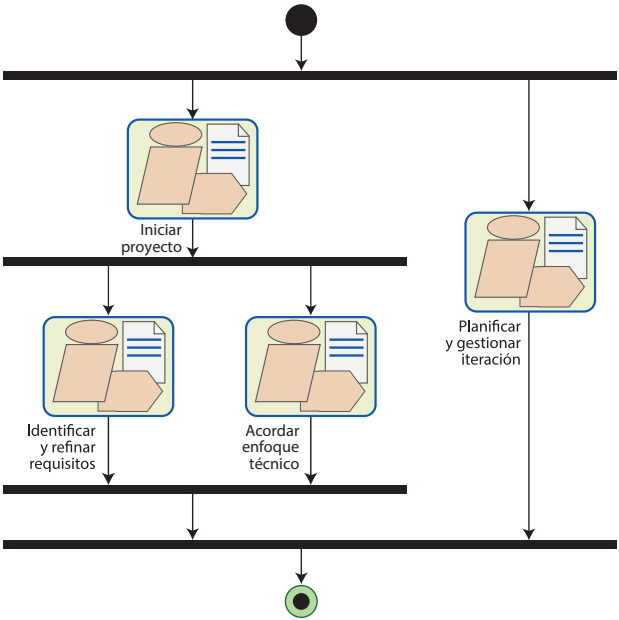


Figura 5.6. Fase de inicio de OpenUP modelado con la notación SPEM

Fuente: elaboración propia con base en OpenUP (2012) desarrollado con la herramienta Enterprise Architect.

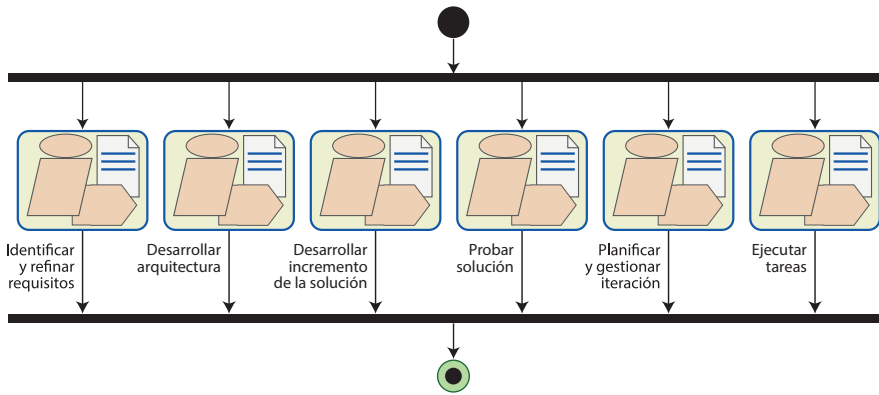


Figura 5.7. Fase de elaboración de OpenUP modelado con la notación SPEM

Fuente: elaboración propia con base en OpenUP (2012) desarrollado con la herramienta Enterprise Architect.

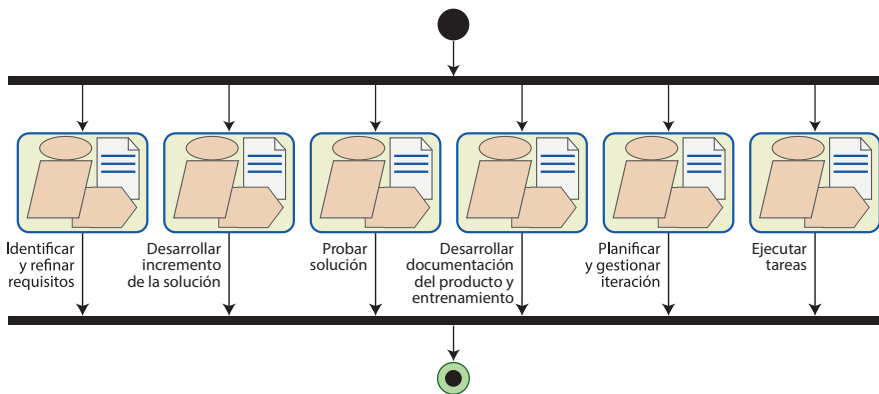


Figura 5.8. Fase de construcción de OpenUP modelado con la notación SPEM

Fuente: elaboración propia con base en OpenUP (2012) desarrollado con la herramienta Enterprise Architect.

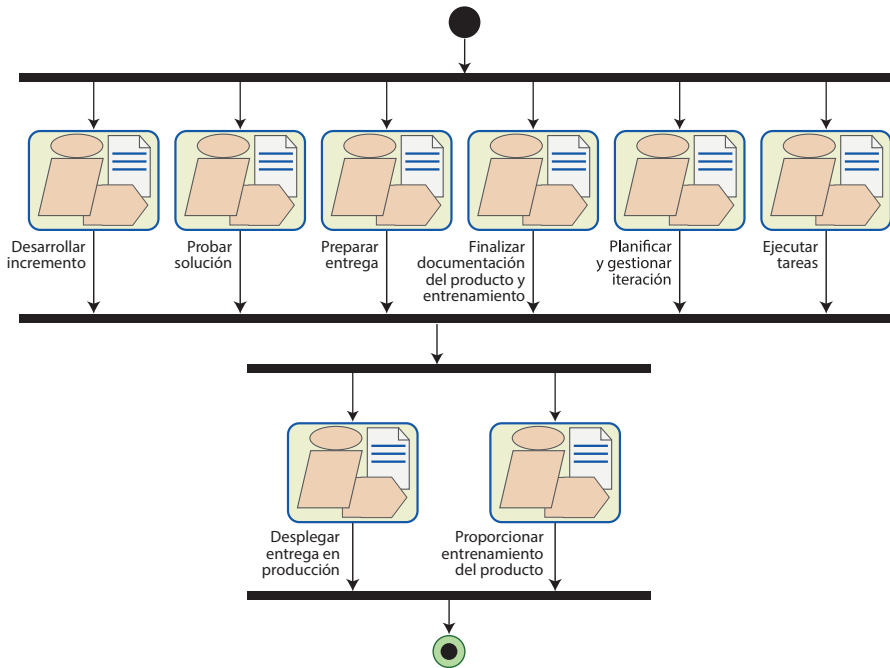


Figura 5.9. Fase de transición de OpenUP modelado con la notación SPEM

Fuente: elaboración propia con base en OpenUP (2012) desarrollado con la herramienta Enterprise Architect.

5.3.2. Business Process Model and Notation (BPMN)

La **BPMN** está enfocada en representar procesos de negocios de manera gráfica para que los usuarios finales puedan comprender los conceptos de los procesos, como el flujo de actividades, la información utilizada y la asignación de roles (Campos y Oliveira, 2013). La **BPMN** es abierta, estandarizada y mantenida por la **OMG**, la misma organización que mantiene **SPEM**.

El concepto de proceso es un concepto mucho más general, se puede definir como “colección de eventos, actividades y decisiones interrelacionados que involucran varios actores y recursos, y que en forma conjunta permiten un resultado que proporciona valor para una organización o sus clientes” (Dumas *et al.*, 2013). De tal manera que los procesos de software son un tipo específico de proceso de negocio.

El objetivo principal de **BPMN** es proporcionar una notación fácil de comprender para usuarios del negocio, desde los analistas de negocio hasta desarrolladores técnicos responsables de implementar la tecnología que ejecutarán los procesos. Por tanto, la **BPMN** crea un puente estandarizado para la brecha entre el diseño del proceso de negocio y la implementación del proceso.

Los elementos gráficos de la **BPMN** se dividen en cinco categorías básicas: objetos de flujo, datos, objetos conectores, carriles y artefactos, como se observa en la tabla 5.2. Hay tres tipos básicos de submodelos en la **BPMN**:

► Orquestación:

- *Procesos de negocio privados*: Tanto ejecutables (modelados con el propósito de documentar el comportamiento del proceso en un nivel definido de detalle) como no ejecutables (modelados para ser ejecutados de acuerdo con la semántica). Ambos son procesos internos de la organización. En la figura 5.10, se presenta un ejemplo de proceso privado no ejecutable.

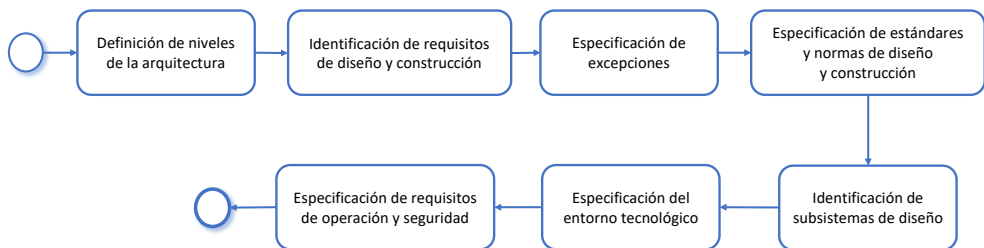


Figura 5.10. Ejemplo de un proceso de negocio privado

Fuente: elaboración propia con base en omg (2008a) desarrollado con la herramienta Bizagi Modeler.

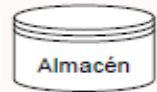
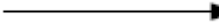
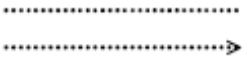
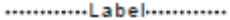
- *Procesos públicos*: Representan la interacción entre un proceso de negocio privado y otro proceso o participante. En la figura 5.11, se presenta un ejemplo de proceso público de la actividad de establecimiento de requisitos del proceso de análisis del sistema de información de métrica V3.

Tabla 5.2. Elementos gráficos de la BPMN

Categoría	Elemento	Descripción	Notación
Objetos de flujo	Eventos	Eventos simples: Eventos sin especificar. Indican puntos de inicio, de fin y situaciones intermedias.	 Inicio  Intermedio  Fin  Fin del proceso
		Mensajes: Recepción y envío de mensajes.	       
		Temporales: Puntos en el tiempo, lapsos, límites (<i>time-outs</i>). Pueden ser eventos únicos o cíclicos.	     
		Escalables: Cambio a un nivel más alto de responsabilidad.	     
		Condicionales: Reaccionan a cambios en las condiciones de negocios o integración de reglas de negocio.	     
		Enlaces: Conectores fuera de página. Dos conectores de enlace equivalen a un flujo de secuencia.	 
		Errores: Capturan y lanzan errores conocidos con un nombre.	  

Categoría	Elemento	Descripción	Notación
Objetos de flujo		Cancelaciones: Reaccionan a la cancelación de una transacción/ solicitud de cancelación.	
		Compensaciones: Manejo/solicitud de compensación.	
	Eventos	Señales: Intercambio de señales entre procesos. Una señal puede ser capturada varias veces.	
		Múltiples: Capturan un evento de un conjunto de eventos. Lanzas todos los eventos definidos.	
		Paralelos múltiples: Capturan todos los eventos de un conjunto de eventos en paralelo.	
	Actividades	Son términos genéricos para el trabajo que realiza la empresa en un proceso. Una actividad puede ser atómica o no atómica (compuesta).	Tarea Subproceso

Categoría	Elemento	Descripción	Notación
Objetos de flujo	Compuertas	Se utilizan para controlar la divergencia y la convergencia de los flujos de secuencia en un proceso y en una coreografía. Por tanto, determinan la ramificación, bifurcación, fusión y unión de rutas.	   Exclusiva Compleja
			  Basada en eventos Exclusiva basada en eventos
			  Paralela Paralela basada en eventos
			 Inclusiva
Datos	Objetos de datos	Proporcionan información sobre las actividades que deben realizarse o lo que producen. Pueden representar un objeto singular o una colección de objetos.	
	Entradas y salidas de datos	Las actividades y los procesos necesitan datos para ejecutarse. Además, pueden producir datos durante o como resultado de la ejecución.	

Categoría	Elemento	Descripción	Notación
Datos	Almacenes de datos	Proporcionan un mecanismo para que las actividades recuperen o actualicen información almacenada que persistirá más allá del alcance del proceso.	
	Flujos de secuencia	Se utilizan para mostrar el orden en el que las actividades se ejecutarán.	 Condicional
			 Por defecto
			 Secuencia
Objetos conectores	Flujos de mensaje	Muestran el flujo de comunicación entre dos participantes.	
	Asociaciones	Se utilizan para vincular información y artefactos con elementos gráficos de la BPMN. Pueden tener una punta de flecha que indica la dirección de flujo.	
	Asociaciones de datos	Es un elemento base contenido en una actividad o evento que se usa para modelar cómo se introducen o extraen los datos de elementos.	
Carriles	Participantes, <i>pools</i>	Es la representación gráfica de un participante en una colaboración. Actúa como un “carril de navegación” y un contenedor gráfico para particionar actividades. Puede presentar detalles internos o externos.	

Categoría	Elemento	Descripción	Notación
Carriles	Carriles, <i>lanes</i>	Es una subpartición dentro de un proceso, a veces dentro de un <i>pool</i> , y extienden la longitud del proceso vertical u horizontalmente. Los carriles se usan para organizar y categorizar actividades.	
Artefactos	Grupos	Son agrupaciones de elementos gráficos que están dentro de la misma categoría. Pueden tener detalles internos o no.	
	Anotaciones de texto	Son un mecanismo para proporcionar información de texto adicional.	

Fuente: elaboración propia con base en omg (2008a).

- *Coreografías*: Son definiciones del comportamiento esperado, básicamente un contrato procedimental entre los participantes que están interactuando. En una coreografía, las actividades son interacciones que representan un conjunto de intercambios de mensajes, que involucra a dos o más participantes. A diferencia de un proceso normal, no existe un controlador central, entidad responsable u observador del proceso. En la figura 5.12, se presenta un ejemplo de una coreografía.

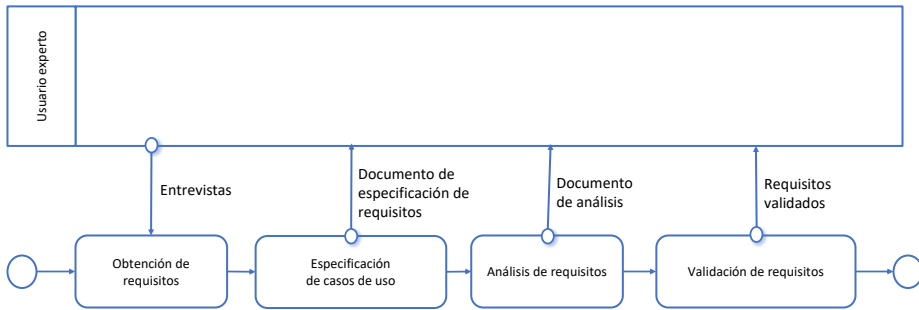


Figura 5.11. Ejemplo de un proceso de negocio público

Fuente: elaboración propia desarrollada con la herramienta Bizagi Modeler.

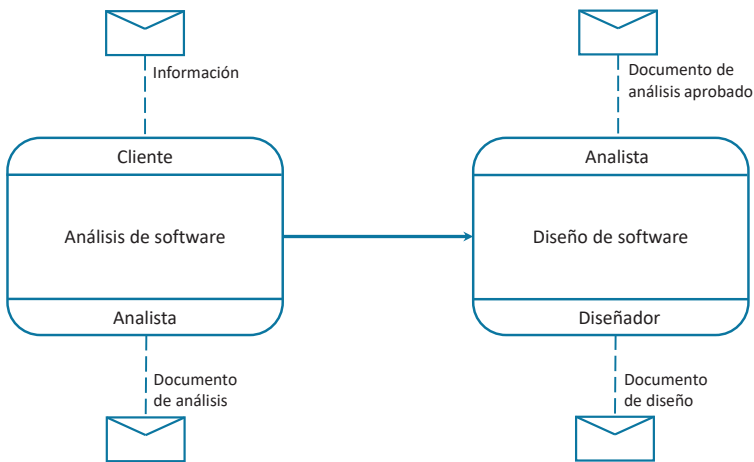


Figura 5.12. Ejemplo de una coreografía

Fuente: elaboración propia desarrollada con la herramienta Enterprise Architect.

- **Colaboraciones:** Representan las interacciones entre dos o más entidades de negocio. Generalmente, contienen dos o más grupos, que representan a los participantes en la colaboración. El intercambio de mensajes entre los participantes se muestra mediante un flujo de mensajes que conecta dos grupos (o los objetos dentro de los grupos). Los mensajes asociados con los flujos de mensajes también se pueden mostrar. Una colaboración muestra cómo dos o más procesos públicos se comunican entre sí (figura 5.13).

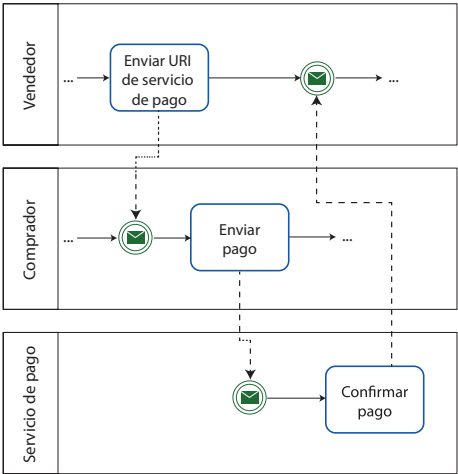


Figura 5.13. Ejemplo de una colaboración

Fuente: elaboración propia con base en Decker (2009) desarrollado con la herramienta Bizagi Modeler.

- **Conversaciones:** Es un uso particular y una descripción informal de un diagrama de colaboración. Una conversación es la relación lógica de los intercambios de mensajes entre objetos de negocio. Un ejemplo de diagrama de conversación se presenta en la figura 5.14 con conversaciones (representadas como hexágonos) entre participantes.

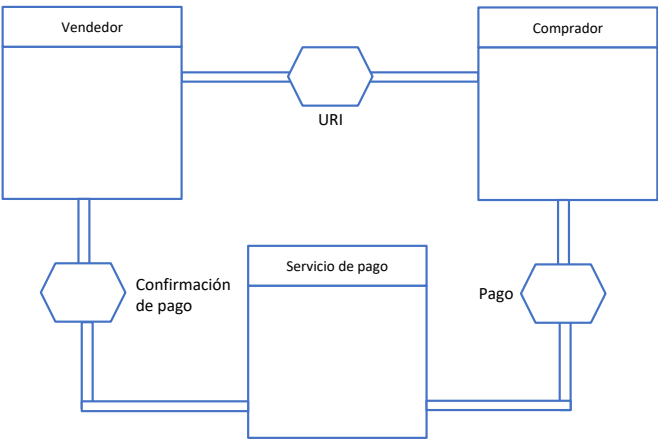


Figura 5.14. Ejemplo de un diagrama de conversación

Fuente: elaboración propia con base en Decker (2009) desarrollado con la herramienta Enterprise Architect.

En la figura 5.15, se presenta un ejemplo de modelo de proceso de software elaborado con la notación de BPMN 2.0. El modelo de proceso de software representa el proceso *scrum* con sus roles, eventos y artefactos principales.

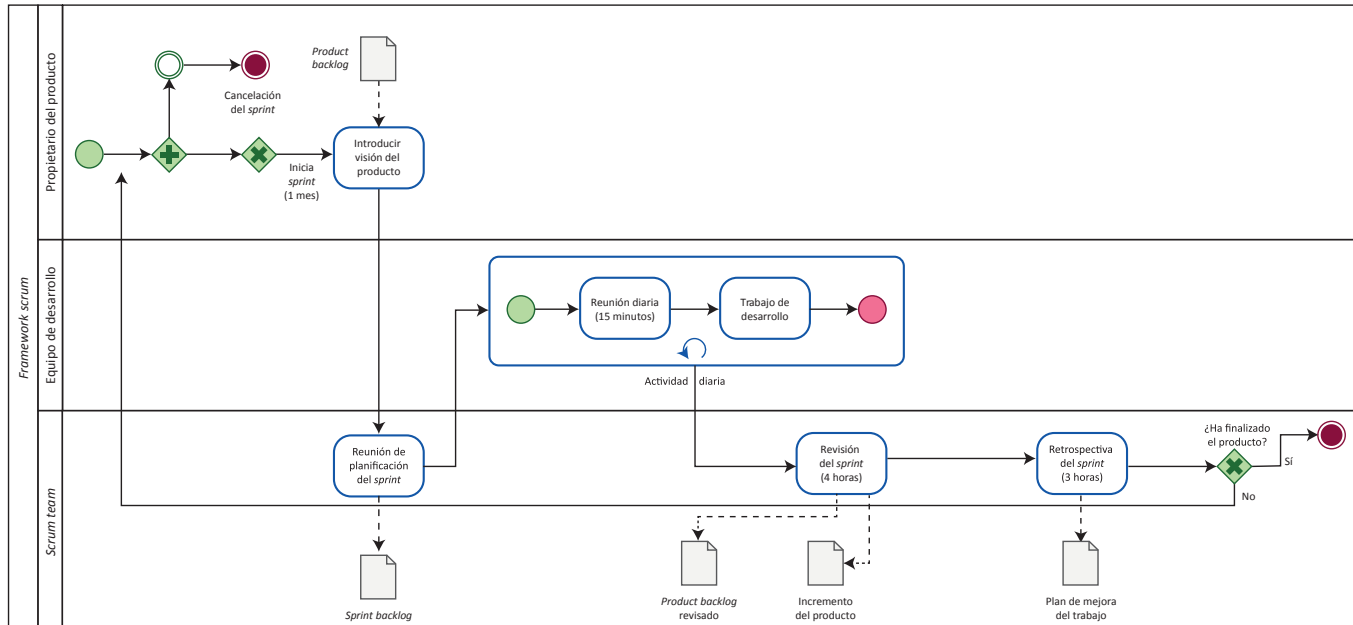


Figura 5.15. Proceso *scrum* modelado con BPMN 2.0

Fuente: elaboración propia con base en Ris-Ala (2016) desarrollado con la herramienta Bizagi Modeler.

5.3.3. Diagramas de actividad UML

Dentro del lenguaje de modelado visual **UML** (OMG, 2017), se encuentran los diagramas de actividad, un tipo de diagrama que permite modelar procesos y flujos de trabajo. Los diagramas de actividad son modelos de comportamiento de un sistema de software.

Los diagramas de actividad modelan una secuencia de acciones y condiciones dentro de un proceso mostrando el flujo de las acciones dentro de una actividad utilizando nodos y transiciones para modelar tanto el flujo de control como el de datos (OMG, 2017). La figura 5.16 muestra el metamodelo parcial de los diagramas de actividad **UML** correspondiente a un subconjunto compatible del estándar **UML**.

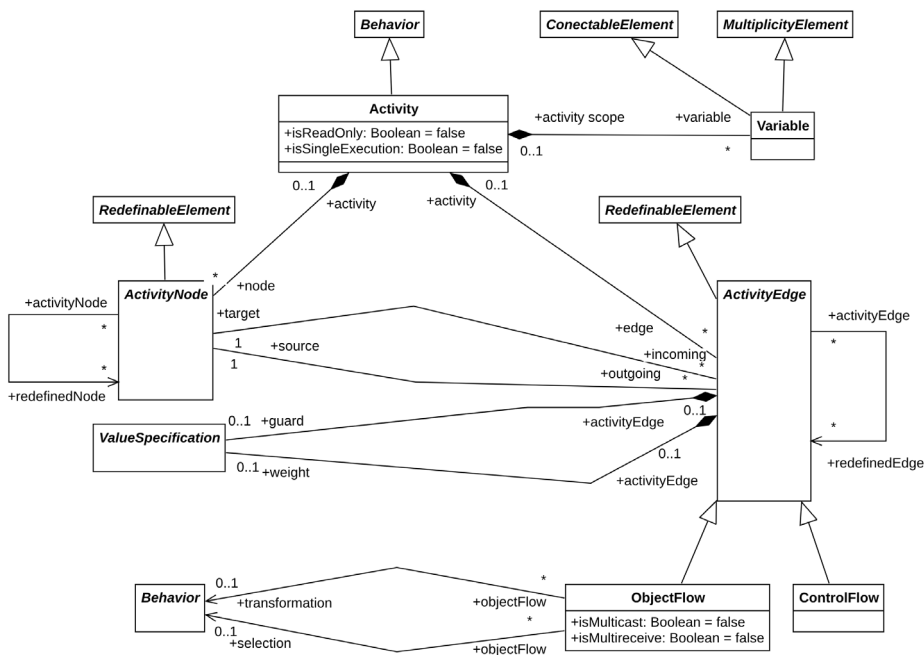
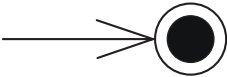
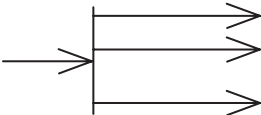
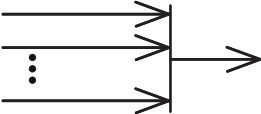
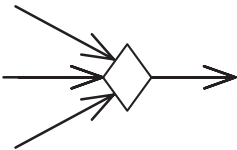
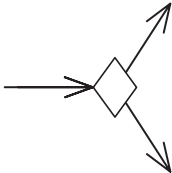


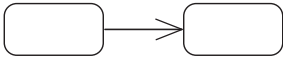
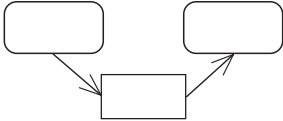
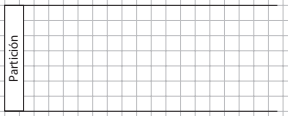
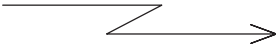
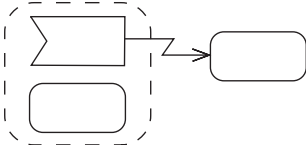
Figura 5.16. Sintaxis abstracta de diagramas de actividad

Fuente: elaboración propia con base en OMG (2017).

En la tabla 5.3, se presenta un listado de elementos gráficos de un diagrama de actividad **UML**.

Tabla 5.3. Elementos gráficos de los diagramas de actividad UML

Elemento	Ícono	Descripción
Nodo acción		Realizan operaciones con los datos que reciben y pasan control y datos a otras acciones.
Nodo de control inicial		Inicia el flujo cuando se invoca la actividad.
Nodo de control final de flujo		Termina un flujo. Destruye todos los <i>tokens</i> que llegan a él, pero no tiene ningún efecto sobre otros flujos en la actividad.
Nodo de control final de actividad		Detiene todos los flujos de una actividad.
Nodo de control de bifurcación (<i>fork</i>)		Tiene un flujo entrante y múltiples flujos salientes. Se usa para dividir el flujo entrante en múltiples flujos concurrentes. Sirve para apoyar el paralelismo en las actividades.
Nodos de control de unión (<i>join</i>)		Tiene múltiples flujos entrantes y un flujo saliente, y se usa para sincronizar flujos concurrentes entrantes. Sirven para apoyar el paralelismo en las actividades.
Nodo de control de decisión de fusión (<i>merge</i>)		Reúne múltiples flujos alternativos entrantes para aceptar un solo flujo saliente.
Nodo de control de decisión (<i>decision</i>)		Acepta <i>tokens</i> en uno o dos flujos entrantes y selecciona un flujo saliente de uno o más flujos salientes. Permiten incorporar condicionales en las actividades.
Nodos objeto		Se utilizan para definir el flujo de objetos en una actividad. Indica que una instancia de un clasificador particular, en un estado particular, puede estar disponible en un punto particular de la actividad.

Elemento	Ícono	Descripción
Flujo de control		Comienza un nodo de actividad después que el nodo previo haya terminado. Sirve para modelar el paso de una acción a otra.
Flujo de datos		Contiene objetos o datos que se envían pasando por él. Sirve para modelar el paso de información de una acción a otra.
Partición		Es un grupo de actividad para acciones que tienen alguna característica común. Corresponden a unidades organizacionales o actores de negocio en un modelo de negocio.
Actividad estructurada		Actividad que puede ampliarse a nodos de actividades subordinadas. Los nodos subordinados pertenecen a una sola actividad estructurada, pero pueden anidarse.
Excepción		Sirven para modelar acciones que hay que llevar a cabo en caso que una excepción ocurra durante la ejecución de un proceso, que se interrumpe al llegar la excepción.
Regiones de actividad interrumpibles		Es una región dentro de la actividad que interrumpe todos sus flujos cuando un <i>token</i> atraviesa sus límites a través de uno de sus flujos de salida.

Fuente: elaboración propia con base en OMG (2017).

En la figura 5.17, se presenta un ejemplo de modelo de proceso de software elaborado con la notación de los diagramas de actividad UML. El modelo representa una práctica de trabajo de la International Organization for Standardization (ISO), u Organización Internacional de Normalización.

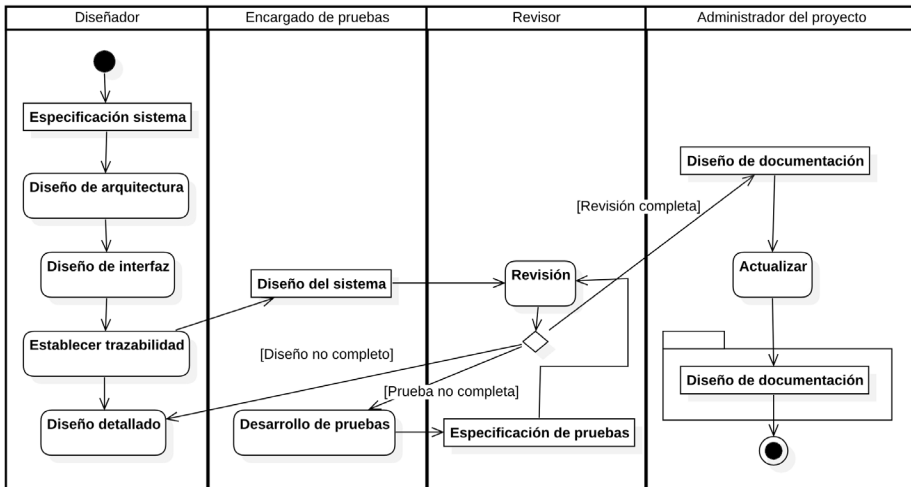


Figura 5.17. Práctica de trabajo de la ISO modelada con un diagrama de actividad UML.

Fuente: elaboración propia con base en Holt (2017) desarrollado con la herramienta StarUML.

5.3.4. IDEF0

IDEF0 es una metodología de modelado funcional para describir funciones de manufactura (Cempel y Dąbal, 2014). Se basa en Structured Analysis and Design Technique (**SADT**), técnica de análisis y diseño estructurado, y fue desarrollado por Douglas T. Ross y SofTech, Inc.

IDEF0 incluye una definición de lenguaje de modelado gráfico (con sintaxis y semántica) y una descripción de una metodología integral para desarrollar modelos. Su método es *top-down* y utiliza gráficos simples para describir las actividades y la interrelación entre estas actividades estructuralmente.

IDEF0 está basado en el enfoque Input, Control, Output, Mechanism (**ICOM**), o entrada, control, salida y mecanismo. Los dos componentes principales de modelado en **IDEF0** son funciones (representadas por cuadros en un diagrama) y los datos y objetos que interrelacionan tales funciones (representadas por flechas). Los modelos **IDEF0** se componen de tres tipos de información: diagramas gráficos, texto y glosario. Estos tipos de diagramas tienen referencias cruzadas entre sí. El diagrama gráfico es el componente principal de un modelo **IDEF0**, que contiene recuadros, flechas, interconexiones de caja/flecha y relaciones asociadas (Cempel y Dąbal, 2014).

Los elementos fundamentales del modelado de procesos con **IDEF0** se presentan en la figura 5.18. Las funciones se representan con cajas y sus elementos constituyentes: entradas, salidas, controles, mecanismos y llamadas, se representan con flechas. En la tabla 5.4, se presenta una descripción de los elementos gráficos de **IDEF0**.

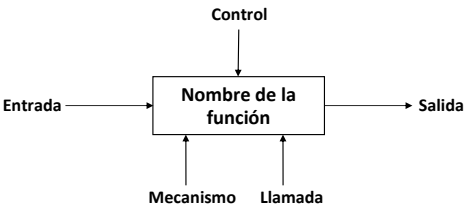


Figura 5.18. Sintaxis abstracta de diagramas de actividad
Fuente: elaboración propia.

Tabla 5.4. Elementos gráficos de los modelos IDEF0

Elemento	Ícono	Descripción
Cajas		Proporcionan una descripción en una función designada.
Entradas		Datos u objetos que la función transforma en resultados.
Salidas		Datos u objetos producidos por una función.
Control		Condiciones requeridas para producir una salida correcta. Los datos u objetos modelados como controles pueden ser transformados por la función creando salidas.
Mecanismos		Medios utilizados para realizar una función.
Llamada		Tipo de mecanismo que permite compartir detalles entre modelos (uniéndolos) o dentro de un modelo.

Fuente: elaboración propia.

Las funciones se descomponen en diagramas más detallados, hasta que el proceso se describe a un nivel necesario para comprender los objetivos de un proyecto en particular (figura 5.19). El diagrama de nivel superior en el modelo proporciona la descripción más general o abstracta del tema representado por el modelo. Este diagrama es seguido de una serie de diagramas secundarios que proporcionan más detalles.

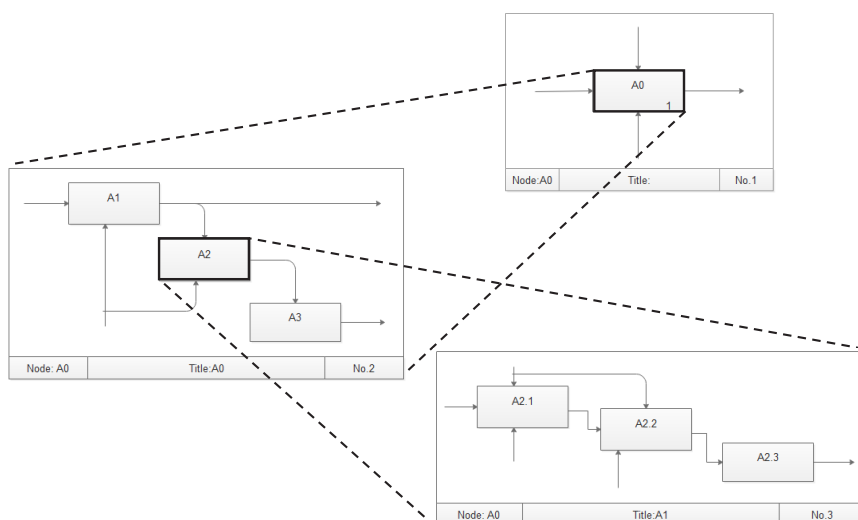


Figura 5.19. Estructura de descomposición en IDEF0

Fuente: elaboración propia.

En las figuras 5.20 y 5.21, se presenta un ejemplo de modelo de proceso de software elaborado en la notación de IDEF0. El modelo representa la actividad de “Diseño de casos de uso reales” perteneciente al proceso de “Diseño del sistema de información” de la metodología métrica V3.

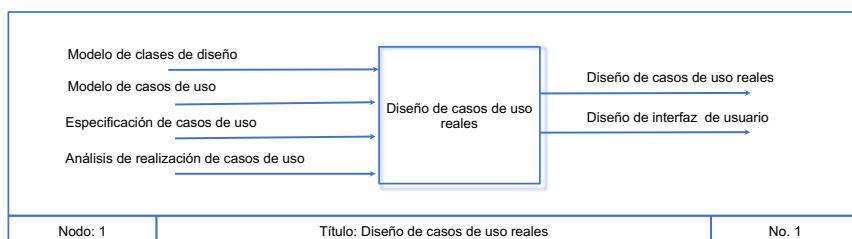


Figura 5.20. Estructura de descomposición de nivel superior en IDEF0

Fuente: elaboración propia desarrollada con la herramienta Edraw Max.

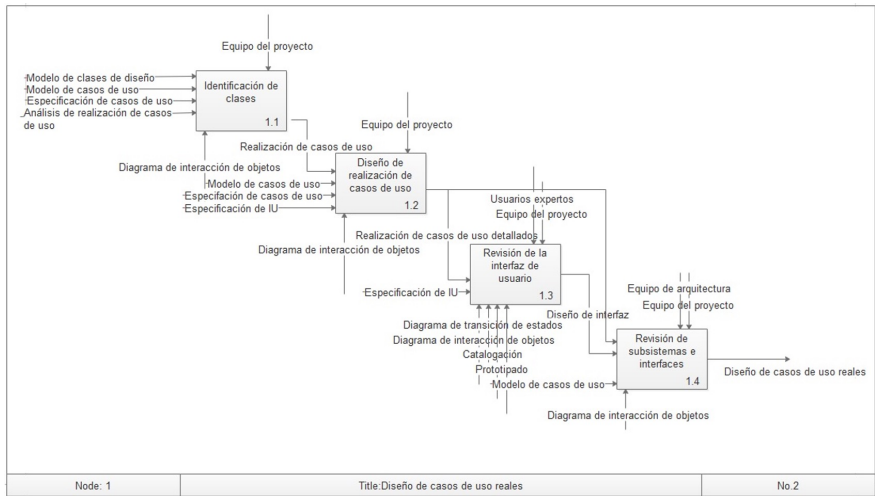


Figura 5.21. Estructura de descomposición desglosada en IDEF0

Fuente: elaboración propia desarrollada con la herramienta Edraw Max.

5.3.5. Event-Driven Process Chain (EPC)

Las cadenas de proceso basadas en eventos (Event-Driven Process Chain [EPC]) son otra técnica popular que permite modelar procesos de negocio. Su enfoque está basado en una documentación semiformal del proceso en lugar de una especificación formal (Mendling *et al.*, 2005). EPC fue desarrollado en el marco del proyecto ARIS por Wilhelm-August Scheer en la Universidad del Sarre (Alemania) a principios de la década de 1990.

EPC es un gráfico dirigido de eventos y funciones. Implica diversos elementos de conexión, con los que se pueden describir las rutas de ejecución alternativas y paralelas. Utiliza operadores lógicos como **OR**, **AND** y **XOR**. La inteligibilidad y la sencillez de EPC son uno de sus mayores puntos fuertes. Sin embargo, su sintaxis y semántica no están bien definidas (Kelemen *et al.*, 2013).

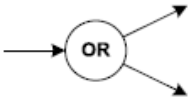
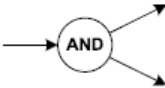
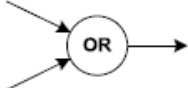
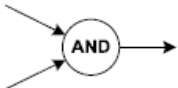
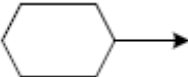
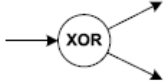
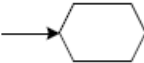
EPC siempre comienza con un evento inicial y termina con un evento final. Los eventos se utilizan para definir la condición previa y posterior de una función. Los eventos y las funciones tienen un arco de flujo de control entrante y uno saliente. Se utiliza un operador para conectar varios eventos y funciones. Los operadores lógicos se utilizan para la toma de decisiones.

Los objetos de proceso adicionales son opcionales en **EPC** y se utilizan para mostrar los objetos entregables o de información (Amjad *et al.*, 2018). Las pocas reglas semánticas importantes de **EPC** son (Mili *et al.*, 2010):

1. Los eventos no pueden tomar decisiones **OR/XOR**. Los eventos solo se pueden vincular con el operador **AND**.
2. Las funciones se pueden asociar con los tres operadores lógicos (**AND**, **OR**, **XOR**) para la toma de decisiones.
3. Los objetos de proceso adicionales solo se pueden conectar con las funciones de **EPC**.

La tabla 5.5 proporciona una visión general de los diferentes elementos notacionales de **EPC**.

Tabla 5.5. Elementos notacionales gráficos de EPC

Elemento	Notación gráfica	Elemento	Notación gráfica
Función		Conector OR-división	
Conector AND-división		Conector OR-unión	
Conector AND-unión		Evento inicial	
Conector XOR-división		Evento final	
Conector XOR-unión		Evento intermedio	
Participante (unidad organizacional)		Aplicación	
Datos			

Fuente: elaboración propia.

Los elementos de los **EPC** pueden ser de tipo de función (elementos activos), de evento (elementos pasivos) o tres tipos de conectores: **AND**, **OR** o **XOR**. Estos objetos están vinculados mediante arcos de flujo de control.

Las funciones corresponden a actividades. Una función tiene precisamente un arco de entrada y un arco de salida, por tanto, la división y la unión solo se pueden modelar utilizando conectores. Hay tres tipos de eventos (inicio, intermedio y final). Los eventos y las funciones deben alternarse a lo largo de cualquier ruta, es decir, no se permite conectar eventos a eventos o funciones a funciones. Además, existe el estándar **EPC** Markup Language (**EPML**) que permite almacenar los modelos desarrollados en formato Extensible Markup Language (**XML**).

En la figura 5.22, se presenta un ejemplo de modelo de proceso de software elaborado utilizando **EPC**.

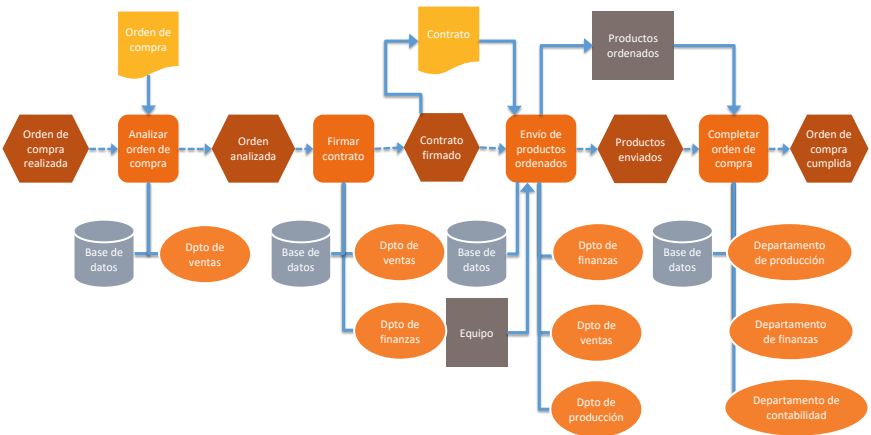


Figura 5.22. Ejemplo de modelo de proceso de software utilizando EPC

Fuente: elaboración propia con base en plantillas de la herramienta ConceptDraw.

5.3.6. Redes de Petri

Las redes de Petri son uno de los lenguajes de modelado de procesos más antiguos y mejor investigados (Van der Aalst, 2016). Su amplio uso se debe a que estas son formales (con un sólido fundamento matemático), tienen técnicas de análisis asociadas y se basan en el estado más que en los eventos (Li y Mao, 2010).

Una red de Petri se compone de lugares, transiciones y arcos. Un lugar está representado por un círculo y puede contener un número discreto de *tokens* (instancia particular de un proceso). Una transición está representada por un rectángulo y se utiliza para generar *tokens* cuando está habilitada. La generación de una transición consume un *token*. Las redes de Petri tienen una teoría matemática bien definida para verificar los procesos de negocio.

Una red de Petri es una tripleta (P, T, F) donde (Amjad *et al.*, 2018):

- P es un conjunto finito de lugares.
- T es un conjunto finito de transiciones ($P \cap T = \emptyset$).
- $F \subseteq (P \times T) \cup (T \times P)$ es un conjunto de arcos.

Una red de Petri que modela el flujo de control de un proceso de negocio es denominada red de flujo de trabajo, si y solo si (Amjad *et al.*, 2018):

- Hay un único lugar inicial $i \in P$, tal que $i = \emptyset$.
- Hay un único lugar final $o \in P$, tal que $o = \emptyset$.
- Cada nodo $x \in P \cup T$ es una ruta de i a o .

En la figura 5.23, se muestra una red de Petri para modelar un proceso de envío de documentos.

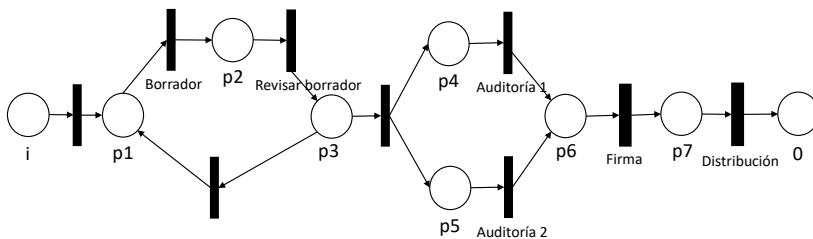


Figura 5.23. Modelo de red de Petri para el proceso de envío de documentos

Fuente: elaboración propia con base en Li y Mao (2010) desarrollado con ConceptDraw.

► 5.4. Comparación de lenguajes de modelado de procesos de software

Los seis lenguajes de modelado de procesos de software presentados en este capítulo permiten construir buenos modelos de procesos de software.

A continuación, en la tabla 5.6 se realiza una comparación de modelos basada en los siguientes criterios según Bendraou *et al.* (2010):

- *Basado en MOF*: El lenguaje de modelado debe estar basado en el estándar **MOF (OMG, 2008b)** para facilitar su exportación, almacenamiento y presentación en diferentes formatos.
- *Modelado de comportamiento*: El lenguaje ofrece facilidades para modelar el comportamiento o aspectos dinámicos del proceso.
- *Automatización del proceso*: El lenguaje tiene soporte para ejecutar modelos de procesos que ayuda a coordinar a los participantes del proceso, hacer cumplir el enrutamiento de artefactos y garantizar las integridades de las restricciones y los plazos del proceso. También ayuda a la simulación y prueba del proceso.
- *Complejidad de la notación gráfica*: El lenguaje incorpora elementos clave del proceso, como procesos, actividades, artefactos, roles y muchos más.
- *Control proactivo*: Es una especificación imperativa del orden en el que se deben ejecutar las actividades. Los principales medios para formalizar el control proactivo son relaciones de precedencia y acciones de llamada.
- *Control reactivo*: Es la especificación de las condiciones o eventos en respuesta a las actividades que se deben ejecutar. Ejemplos de medios utilizados para expresar el control reactivo son las excepciones y la gestión de eventos.
- *Modularidad*: Se trata de la capacidad de combinar diferentes fragmentos de procesos para construir uno nuevo.
- *Formalidad*: Soporte de diferentes enfoques para verificar, validar o analizar modelos de procesos.
- *Herramientas de soporte*: Se encuentran diferentes herramientas comerciales y libres que permiten construir los modelos de procesos.

Como se observa en la tabla 5.6, el lenguaje que cumple más criterios es **BPMN**, el cual tiene la notación gráfica más completa, ofrece capacidades de automatización del proceso y control reactivo. Entre sus desventajas están que tiene una menor formalidad y los procesos que modela son procesos de negocio, los cuales pueden adaptarse a los de software. Las fortalezas de **SPEM** están enfocadas en un lenguaje orientado directamente a procesos de software, mayor grado de formalidad e incorporar diagramas **UML** para complementar los procesos definidos. Entre sus desventajas está no incluir aspectos de modelado de comportamiento y automatización del proceso.

Los diagramas de actividad **UML** tienen las mismas características de **SPEM**, solo que incluye modelado de comportamiento, pero su notación gráfica es más básica. **IDEF0** y **EPC** son los lenguajes que menos criterios cumplen, pero permiten realizar un modelado bastante elemental y modular sin muchos adornos gráficos y sin elementos propios de los procesos de software.

Tabla 5.6. Comparación de lenguajes de modelado de procesos

Criterios	SPEM	BPMN	DA UML	IDEF0	EPC	Red de Petri
Basado en MOF	Sí	Sí	Sí	No	No	Sí
Modelado de comportamiento	No	Sí	Sí	No	No	Sí
Automatización del proceso	No	Sí	No	No	No	No
Complejidad de notación gráfica	Básica pero orientada a procesos de software	Más completa, orientada a procesos de negocios	Muy básica para modelado de <i>workflow</i>	Elemental, orientada a modelado funcional	Orientada a procesos de negocio	Básica pero formal
Control proactivo	Sí	Sí	Sí	Sí	Sí	Sí
Control reactivo	No	Sí (mensajes y <i>time-outs</i>)	No	No	No	No
Modularidad	Sí	Sí	Sí	Sí	Sí	Sí
Formalidad	Media, con OCL	No	Media, con OCL	No	No	Sí
Herramientas de soporte	Sí	Sí	Sí	Sí	Sí	Sí

Fuente: elaboración propia con base en Bendraou *et al.* (2010).

► Desafíos

En las últimas décadas, los lenguajes de modelado de procesos de software han sido un tema ampliamente investigado en la ingeniería de software. De hecho, a lo largo de los años han aparecido diferentes propuestas, ya sea basadas en **UML**, metamodelos o en gramáticas, aunque la tendencia actual trata de utilizar metamodelos como tecnología base al generar nuevos lenguajes de modelado de procesos (Henderson-Sellers y González-Pérez, 2005).

Los lenguajes de modelado presentados en este capítulo se caracterizan por su facilidad de entendimiento, todos incluyen elementos gráficos que representan un aspecto particular del proceso de software. Llegado el caso, algunos ofrecen metáforas para ayudar al entendimiento y facilitar el aprendizaje a usuarios no técnicos (García-García *et al.*, 2019).

La mayoría de los lenguajes de modelado que se han analizado definen su propio conjunto de conceptos para modelar procesos de software. La mayoría de las propuestas hacen su definición basada en metamodelos y algunas definen perfiles **UML** en lugar de usar diagramas **UML**. Todos los modelos presentados incorporan mecanismos de granularidad que permiten la reutilización de bloques lógicos, lo cual resulta muy atractivo en tareas de diseño y gestión debido a los potenciales beneficios que se traducen en ahorros de tiempo y mejoras cualitativas (Hauser, 2010).

Algunos desafíos que presentan los modelos de procesos de software reseñados en este capítulo están relacionados con aspectos, como ejecutabilidad, medición y reglas de negocio (García-García *et al.*, 2019).

Con respecto a las capacidades de ejecución, los lenguajes de modelado de procesos de software presentados tienen como debilidad la falta de ejecutabilidad, aunque algunos diagramas **UML** pueden ejecutarse (simularse) si se les asigna una semántica de ejecución. Los lenguajes de modelado deberían incluir reglas de transformación que permitan generar algún tipo de código ejecutable que facilite la implementación del proceso de software. Es perentorio que los lenguajes de modelado contemplen mecanismos para la ejecución de procesos. A su vez, la ejecución del modelo de proceso permitirá su correcta validación y verificación (García-Borgoñón *et al.*, 2014).

En cuanto a aspectos de medición, solo **SPEM** admite esta característica, aunque su soporte es parcial, ya que admite la definición de medidas, pero no el cálculo automático en tiempo de ejecución de esas métricas/indicadores recolectadas.

Por último, un desafío bastante interesante es la incorporación de reglas de negocio que permitan definir directrices y restricciones que ayuden a regular los procesos de software definidos en las organizaciones. Las reglas de negocio deben expresarse en una notación precisa, aunque presentan problemas como la dificultad de utilización por las personas y el orden de las tareas no puede ser controlado (Atkinson *et al.*, 2004).

Otras cuestiones pendientes están relacionadas con el desarrollo de nuevas propuestas orientadas a integrar modelos de procesos de software

con herramientas de apoyo a la decisión, mejorar la evolución de los procesos y cómo las tecnologías podrían integrarse en el modelado de procesos de software (García-Borgoñón *et al.*, 2014).

► Ejemplos

El Sistema Integrado de Simulación Astros (**sis**-Astros) tenía como objetivo principal el desarrollo de un sistema de simulación integrado para apoyar la enseñanza del uso de una batería de cohetes de artillería. El equipo de desarrollo estaba conformado por siete PhD, tres investigadores, cuatro desarrolladores, siete estudiantes de maestría y trece estudiantes de pregrado. El proyecto requería algunas soluciones innovadoras relacionadas con la integridad del simulador, generación de escenarios 3D y navegación autónoma.

Para incluir entregas formales, el ciclo de vida se organizó en fases e iteraciones. Se planearon dos fases: inicio y construcción, finalizando con un hito importante.

La fase de iniciación solo ocurrió una vez y era responsable de definir una descripción general del sistema en desarrollo y brindar una comprensión clara del dominio de negocios del sistema.

La fase de construcción era responsable de la ejecución de las actividades técnicas que generarían una nueva versión del software. El proyecto podía tener tantas fases de construcción como fuera necesario, y cada una, múltiples iteraciones. Ambas fases debían respetar las obligaciones contractuales, por lo que se culminaban con un hito que indicaba una entrega formal.

Dado que las fases se referían a lapsos grandes (semestres, años), se optó por dividir las muchas iteraciones para acelerar el proceso. Cada iteración tenía su ciclo de desarrollo completo, desde la definición de requisitos hasta la evaluación de la versión (figura 5.24). Las versiones de software desarrolladas en la iteración siempre se entregaban cuando finalizaba la fase. El número de fases y la cantidad de iteraciones en cada una de estas fases dependía del proyecto y se podían ajustar.

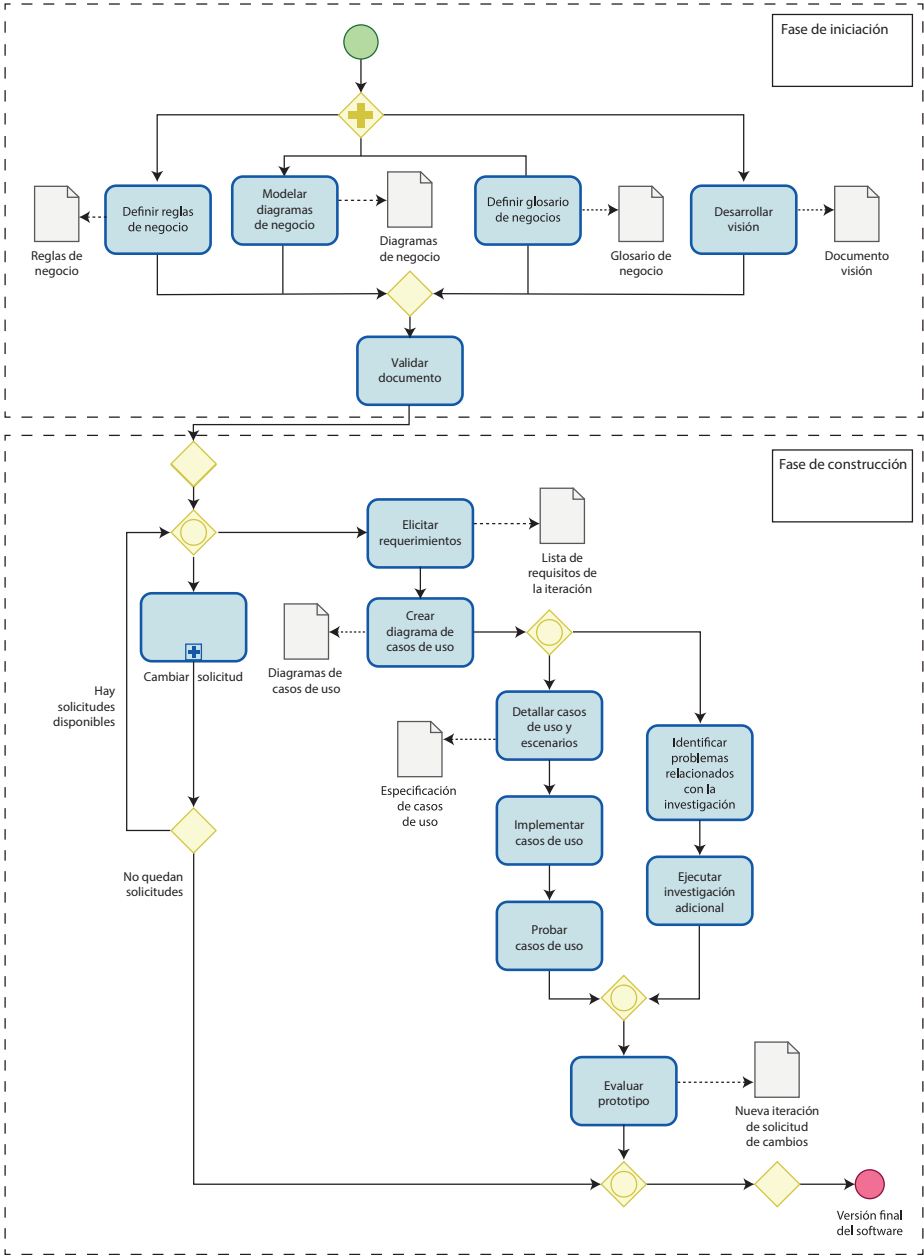


Figura 5.24. Modelo SPeM del sis-Astros

Fuente: elaboración propia con base en Brondani *et al.* (2019).

Al final de la iteración, se realizaban reuniones con el cliente para presentar la versión intermedia del software en las que se discutían posibles mejoras, cambios y evoluciones.

► Resumen

Un modelo se considera un conjunto de declaraciones sobre un sistema en estudio. Los modelos tienen un propósito, una referencia al original, abstracción de cualidades del original y una característica pragmática.

Los modelos de procesos de software poseen elementos básicos, como actividades, productos, herramientas, reglas, roles y ejecutores.

Los modelos de procesos se expresan por medio de lenguajes de modelado de procesos. Tales lenguajes poseen una sintaxis (conjunto de elementos legales) y una semántica (su significado).

SPEM es un lenguaje de modelado de procesos de software desarrollado por la **OMG**. Su metamodelo está conformado por siete paquetes principales. Está orientado específicamente a procesos de software, tiene un mayor grado de formalidad y puede incorporar diagramas **UML**.

BPMN es un lenguaje de modelado de procesos desarrollado también por la **OMG** que permite modelar procesos de negocio y, por tanto, también procesos de software. Su notación gráfica es la más completa y extensa, ofrece capacidades de automatización del proceso y control reactivo.

Los diagramas de actividad son un tipo de diagramas **UML** que permiten modelar el comportamiento de un sistema de software y flujos de trabajo en un modelo de proceso de software.

IDEFO es una metodología de modelado funcional que permite desarrollar modelados muy básicos de procesos de software por medio de notaciones gráficas para representar entradas, controles, salidas y mecanismos.

EPC también permite modelar procesos de negocio a través de una documentación semiformal. Se caracteriza por estar compuesto de un gráfico de eventos y funciones relacionados por medio de operadores lógicos.

Las redes de Petri son los elementos de modelado más antiguos con una amplia investigación y un sólido formalismo matemático. Una red de Petri se compone de lugares, transiciones y arcos.

Cualesquiera de estos lenguajes de modelado de procesos tienen sus ventajas y desventajas para modelar procesos de software. **SPEM** es el único

definido para diseñar modelos concretos orientados al proceso de software. Sin embargo, **BPMN** y los diagramas **UML** tienen también una amplia acogida por parte de los modeladores de procesos de software.

► Ejercicios

1. ¿Cuál es la diferencia entre un modelo del proceso de software y un proceso de software?
2. Seleccionar una etapa específica de un proceso de software (por ejemplo, diseño orientado a objetos o pruebas unitarias) y especificar sus elementos de acuerdo con la figura 5.2.
3. Se tiene el siguiente proceso de negocio. Un estudiante universitario realiza una solicitud de cancelación de una asignatura en un sistema de información académico. Si la solicitud se realiza durante las primeras dos semanas del periodo académico, el sistema la autoriza en forma automática y envía una respuesta al correo electrónico del estudiante. Si el estudiante la realiza después de la segunda semana y antes de la octava, la solicitud debe ser evaluada por el director del programa y luego debe ser ratificada por el comité de la carrera. La respuesta será enviada por correo electrónico al estudiante. Las solicitudes realizadas después de la octava semana son por fuerza mayor y son evaluadas por un comité de bienestar, el cual envía su concepto al comité de la carrera, quien toma la decisión final que será enviada al estudiante. Para esta especificación, realizar un diagrama de procesos de negocio utilizando **BPMN**, diagramas de actividad **UML** e **IDEF0**.
4. Convertir el modelo de proceso de software *scrum* presentado en la figura 5.5 a notación de diagrama de actividad **UML** e **IDEF0**.
5. Elaborar un diagrama **SPEM** para las fases de análisis del sistema de información, diseño del sistema de información y construcción del sistema de información de la norma métrica V3.
6. Un estudiante requiere el préstamo de un libro en una biblioteca. Este debe solicitarlo. Luego, se enviará una respuesta sobre el estado del libro que puede ser prestado o disponible. Si este está disponible, el estudiante debe presentar la orden de préstamo y el proceso finaliza con el préstamo de este. Si el libro no está en

préstamo, el estudiante puede optar por cancelar la solicitud si está en préstamo por un mes; o si el libro está en préstamo por solo dos semanas, el estudiante optará por reservar la solicitud y pedirá reservarlo una vez sea entregado. Después de dos semanas, el estudiante recibirá el estado de disponibilidad del libro y lo obtendrá. Para la anterior especificación, realizar un diagrama **BPMN**, **EPC** o diagrama de actividad **UML**.

7. Convertir el modelo de proceso **EPC** de la figura 5.22 en una red de Petri.
8. Investigar diversas herramientas libres y comerciales que permitan desarrollar los modelos de procesos de software presentados en este capítulo.

► Referencias

- Amjad, A., Azam, F., Anwar, M. W., Butt, W. H. y Rashid, M. (2018). Event-driven process chain for modeling and verification of business requirements: A systematic literature review. *IEEE Access*, 6, 9027-9048. <https://doi.org/10.1109/ACCESS.2018.2791666>
- Atkinson, D. C., Weeks, D. C. y Noll, J. (2004). The design of evolutionary process modeling languages. En *11th Asia-Pacific Software Engineering Conference* (pp. 73-82). *IEEE*. <https://doi.org/10.1109/APSEC.2004.98>
- Bandinelli, S. C., Fuggetta, A. y Ghezzi, C. (1993). Software process model evolution in the **SPADE** environment. *IEEE Transactions on Software Engineering*, 19(12), 1128-1144. <https://doi.org/10.1109/32.249659>
- Bendraou, R., Jézéquel, J. M., Gervais, M. P. y Blanc, X. (2010). A comparison of six **UML**-based languages for software process modeling. *IEEE Transactions on Software Engineering*, 36(5), 662-675. <https://doi.org/10.1109/TSE.2009.85>
- Brcina, R. (2007). Arbeiten zur Verfolgbarkeit und Aspekte des Verfolgbarkeitsprozesses. *Softwaretechnik-Trends: Mitteilungen von mehreren Fachgruppen des Fachausschusses*, 27(1), 3-8.
- Brondani, C. H., da Cruz Mello, O. y Fontoura, L. M. (2019). A case study of a software development process model for **SIS-ASTROS**. En *SEKE* (pp. 600-776). http://ksiresearch.org/seke/seke19paper/seke19paper_98.pdf
- Broy, M. y Rumpe B. (2007). Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. *InformatikSpektrum*, 30(1), 3-18. <https://doi.org/10.1007/>
- Campos, A. L. N. y Oliveira, T. (2013). Software processes with **BPMN**: An empirical analysis. En J. Heidrich, M. Oivo, A. Jedlitschka y M. T. Baldassarre (eds.), *Product-Focused Software Process Improvement. PROFES 2013. Lecture Notes in Computer Science* (pp. 338-341). Springer. https://doi.org/10.1007/978-3-642-39259-7_29

- Cempel, W. A. y Dabal, D. (2014). **IDEFO** as a project management tool in the simulation modeling and analysis process in emergency evacuation from hospital facility: A case study. En P. Pawlewski y A. Greenwood (eds.), *Process Simulation and Optimization in Sustainable Logistics and Manufacturing. EcoProduction* (pp. 155-166). Springer. https://doi.org/10.1007/978-3-319-07347-7_11
- Conradi, R., Jaceheri, M. L., Mazzi, C., Nguyen y M. N., Aarsten, A. (1992). Design, use and implementation of **SPELL**: A language for software process modeling and evolution. En J. C. Derniame (eds.), *Software Process Technology. EWSPT 1992. Lecture Notes in Computer Science* (pp. 167-177). Springer. <https://doi.org/10.1007/BFb0017519>
- Decker, G. (2009). *Design and analysis of process choreographies* [tesis de doctorado, Universität Potsdam]. https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/3898/file/decker_diss.pdf
- DeMarco, T. (2004). *Was man nicht messen kann, kann man nicht kontrollieren*. MitpVerlag.
- Derniame, J. C., Kaba, B. A. y Wastell, D. (eds.) (1999). *Software process: Principles, methodology, and technology*. Springer.
- Dowson, M. y Fernström, C. (1994). Towards requirements for enactment mechanisms. En B. C. Warboys (eds.), *Software Process Technology. EWSPT 1994. Lecture Notes in Computer Science* (pp. 90-106). Springer. https://doi.org/10.1007/3-540-57739-4_13
- Dumas, M., La Rosa, M., Mendling, J. y Reijers, H. A. (2013). *Fundamentals of business process management*. Springer. <https://doi.org/10.1007/978-3-642-33143-5>
- Gallina, B., Pitchai, K. R. y Lundqvist, K. (2014). S-TunExSPeM: Towards an extension of spem 2.0 to model and exchange tunable safety-oriented processes. En R. Lee (eds.), *Software Engineering Research, Management and Applications. Studies in Computational Intelligence* (pp. 215-230). Springer. https://doi.org/10.1007/978-3-319-00948-3_14
- García-Borgoñón, L., Barcelona, M. A., García-García, J. A., Alba, M. y Escalona, M. J. (2014). Software process modeling languages: A systematic literature review. *Information and Software Technology*, 56(2), 103-116. <https://doi.org/10.1016/j.infsof.2013.10.001>
- García-García, J. A., Enríquez, J. G. y Domínguez-Mayo, F. J. (2019). Characterizing and evaluating the quality of software process modeling language: Comparison of ten representative model-based languages. *Computer Standards & Interfaces*, 63, 52-66. <https://doi.org/10.1016/j.csi.2018.11.008>
- Génova, G. (2012). Conceptos básicos de modelado. En *Desarrollo de software dirigido por modelos: Conceptos, métodos y herramientas* (pp. 67-80). Ra-Ma. <http://www.lcc.uma.es/~av/Publicaciones/12/LibroDSDM.pdf>
- Harel, D. y Rumpe, B. (2004). Meaningful modeling: What's the semantics of "semantics"? *Computer*, 37(10), 64-72. <https://doi.org/10.1109/MC.2004.172>
- Hauser, R. (2010). *Automatic transformation from graphical process models to executable code. ETH Zürich*. <https://doi.org/10.3929/ethz-a-006050258>
- Henderson-Sellers, B. y González-Pérez, C. (2005). A comparison of four process meta-models and the creation of a new generic standard. *Information and Software Technology*, 47(1), 49-65. <https://doi.org/10.1016/j.infsof.2004.06.001>
- Holt, J. (2004). **UML** for systems engineering: *Watching the wheels*. IET.

- Hunter, R. B. y Thayer, R. H. (eds.) (2001). *Software process improvement (practitioners)*. **IEEE** Computer Society.
- Hurtado Alegria, J. A., Bastarrica, M. C. y Bergel, A. (2011). Analyzing software process models with **AVISPA**. En *Proceedings of the 2011 International Conference on Software and Systems Process* (pp. 23-32). <https://doi.org/10.1145/1987875.1987882>
- Kaiser, G. E., Barghouti, N. S. y Sokolsky, M. H. (1990). Preliminary experience with process modeling in the marvel software development environment kernel. En *Proceedings of the 23rd International Conference on System Sciences* (pp. 131-140). **IEEE**. <https://doi.org/10.1109/HICSS.1990.205161>
- Kelemen, Z. D., Kusters, R., Trienekens, J. y Balla, K. (2013). *Selecting a process modeling language for process based unification of multiple standards and models*. https://www.academia.edu/download/40527680/Selecting_a_Process_Modeling_Language_fo20151130-12371-180bp3v.pdf
- Li, Y. B. y Mao, F. Q. (2010). Research of the verification in workflow process modeling on the application of Petri nets. En *2010 International Conference on e-Education, e-Business, e-Management and e-Learning* (pp. 21-24). **IEEE**. <https://doi.org/10.1109/IC4E.2010.71>
- Ludewig, J. y Lichter, H. (2023). *Software engineering: Grundlagen, menschen, prozesse, technischen*. Dpunkt Verlag GmbH.
- Mendling, J., Neumann, G. y Nüttgens, M. (2005). Yet another Event-Driven Process Chain. En W. M. P. van der Aalst, B. Benatallah, F. Casati y F. Curbera (eds.), *Business Process Management. BPM 2005. Lecture Notes in Computer Science* (pp. 428-433). Springer. https://doi.org/10.1007/11538394_35
- Mili, H., Tremblay, G., Jaoude, G. B., Lefebvre, É., Elabed, L. y Boussaidi, G. E. (2010). Business process modeling languages: Sorting through the alphabet soup. **ACM Computing Surveys (CSUR)**, 43(1), 1-56. <https://doi.org/10.1145/1824795.1824799>
- Moro, M. (2004). *Modellbasierte Qualitätsbewertung von Softwaresystemen*. Books on Demand GmbH.
- Nitto, E. D., Lavazza, L., Schiavoni, M., Tracanella, E. y Trombetta, M. (2002). Deriving executable process descriptions from **UML**. En *Proceedings of the 24th International Conference on Software Engineering* (pp. 155-165). <https://doi.org/10.1145/581339.581361>
- Object Management Group. (2008a). *Software & Systems Process Engineering Meta-Model Specification Version 2.0*. **OMG** Document Number: formal/2008-04-01. <https://www.omg.org/spec/SPEM/2.0/PDF>
- Object Management Group. (2008b). *Meta Object Facility (MOF) Core Specification. Version 2.5.1*. **OMG** Document Number: formal/2019-10-01. <https://www.omg.org/spec/MOF>
- Object Management Group. (2017). *OMG Unified Modeling Language (OMG UML) Version 2.5.1*. **OMG** Document Number: formal/2017-12-05. <https://www.omg.org/spec/UML/2.5.1/PDF>
- OpenUP. (2012). *Eclipse Process Framework Composer*. <http://www.utm.mx/~caff/doc/OpenUPWeb/index.htm>
- Pawel, P. (2010). *Using Petri nets to model and simulation production systems in process re-engineering (case study)*. **INTECH** Open Access Publisher. <https://www.intechopen.com/chapters/9195>

- Pereira, E. B., Bastos, R. M., Oliveira, T. C. y Móra, M. C. (2012). A set of well-formedness rules to checking the consistency of the software processes based on **SPEM** 2.0. En R. Zhang, J. Zhang, Z. Zhang, J. Filipe y J. Cordeiro (eds.), *Enterprise Information Systems. ICEIS 2011. Lecture Notes in Business Information Processing* (pp. 284-299). Springer. https://doi.org/10.1007/978-3-642-29958-2_19
- Ris-Ala, R. (2016). *Scrum Framework Drawn in BPMN*. <https://www.linkedin.com/pulse/scrum-drawn-bpmn-rafael-ris-ala-jos%C3%A9-jardim>
- Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5), 26-32. <https://doi.org/10.1109/MS.2003.1231147>
- Sutton, S. M., Heimbigner, D. y Osterweil, L. J. (1995). **APPL/A**: A language for software process programming. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 4(3), 221-286. <https://doi.org/10.1145/214013.214017>
- Van der Aalst, W. (2016). *Process mining: Data science in action*. Springer.

Métricas de procesos de software

► Objetivos

Los objetivos de este capítulo están enfocados en entender la medición del proceso de software. Al estudiar este capítulo, el lector:

- Comprenderá los conceptos básicos de medición utilizando ontologías.
- Estudiará diferentes procedimientos y estándares utilizados para realizar mediciones.
- Aprenderá a recolectar métricas del proceso de software.

La medición es un proceso de obtención experimental de uno o más valores cuantificables que pueden atribuirse razonablemente a una cantidad (Joint Committee for Guides in Metrology [JCGM], 2012). La medición se puede considerar una manifestación del tamaño, la cantidad, calidad o dimensión de un atributo particular de una entidad a medir. La asignación de números, valores o símbolos a estas entidades se debe realizar de una manera no ambigua. Sin embargo, una completa no ambigüedad es bastante difícil de lograr.

La medición es un aspecto a destacar en la realización y aplicación del método científico. Por tanto, el proceso de software requiere, como cualquier disciplina, la recolección de métricas que permitan evaluar el estado actual del proceso para poderlo

caracterizar (obtener una idea de lo que “está ocurriendo”), evaluar (juzgar algunas características del proceso de software), controlar (de acuerdo con un presupuesto y calendario estipulados), mejorar su calidad (identificar partes del proceso que pueden ser modificadas, y así obtener efectos positivos), predecir (establecer relaciones de causa-efecto entre procesos y productos) y redefinirlo con el tiempo (Farooq *et al.*, 2011).

La medición del software se ocupa de derivar un valor numérico o perfil para un atributo de un componente, sistema o proceso de software. Al comparar los valores y los estándares que se aplican a través de una organización, es posible extraer conclusiones sobre la calidad del software o valorar la efectividad de los procesos, las herramientas y los métodos de software (Sommerville, 2015).

Por consiguiente, la medición es el proceso continuo de definición, recopilación y análisis de datos sobre el proceso de desarrollo de software y sus productos para comprender y controlar tanto el proceso como sus productos, y proporcionar información significativa para mejorarlos.

► 6.1. Conceptos sobre medición

La medición se puede considerar un proceso de mapear algunos atributos de un fenómeno del mundo real a un conjunto de símbolos para los cuales se especifican operaciones bien definidas (Fairley, 2011). De esta manera, la medición se puede considerar el acto de determinar una medida o de obtener un resultado a partir de la recolección de uno o más puntos de datos (Khurana, 2007).

Las siguientes son algunas razones por las cuales se deben realizar mediciones al software (Park *et al.*, 1996):

1. Para caracterizar el esfuerzo y entender los procesos, productos, recursos y entornos, estableciendo líneas base para realizar evaluaciones futuras.
2. Para determinar el estado actual de los proyectos con respecto a su planificación.
3. Para realizar predicciones por medio de datos históricos de procesos y productos.
4. Para mejorar la calidad del proceso y del producto de software.

Además, el proceso de recolección de métricas de software requiere de una “etiqueta” adecuada para garantizar un eficaz uso de las métricas dentro de la organización. Algunos puntos a considerar dentro de la “etiqueta” de métricas de proceso son (Grady, 1992):

- ▶ Utilizar el sentido común en el momento de interpretar los datos.
- ▶ Proporcionar retroalimentación continua a individuos y grupos que recolectan métricas.
- ▶ No utilizar métricas para evaluar individuos ni para amenazarlos.
- ▶ Realizar la evaluación desde un punto de vista positivo para mejorar el proceso.
- ▶ Utilizar una amplia gama de métricas.

Se deben establecer algunas diferencias entre los términos *medida*, *métrica* e *indicadores*, ya que la mayoría de las veces se toman como iguales y hay disimilitudes entre ellos (Pressman, 2005).

- ▶ *Medida*: Proporciona una indicación cuantitativa de la extensión, cantidad, dimensión, capacidad o tamaño de algún atributo de un fenómeno del mundo real. Una medida ha sido establecida cuando un único punto de dato ha sido recolectado. Por ejemplo: el tamaño de un programa en líneas de código (Lines of Code [**LOC**]).
- ▶ *Métrica*: Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado (Institute of Electrical and Electronics Engineers [**IEEE**], 1990). Las métricas relacionan medidas de alguna forma. Por ejemplo: la productividad de un programador definida como la cantidad de líneas de código que genera por unidad de tiempo (horas). En este caso, relaciona dos medidas: **LOC** y tiempo.
- ▶ *Indicador*: Es una métrica o combinación de métricas que proporcionan comprensión de algún aspecto del proceso, proyecto o producto para mejorar alguno de sus atributos. Por ejemplo: establecimiento de rangos o intervalos de productividad para determinar si la productividad es excelente, buena, aceptable o deficiente de acuerdo con criterios como el lenguaje de programación y los años de experiencia del programador.

Las medidas se pueden clasificar como directas o derivadas (Bourque y Fairley, 2014; Fairley, 2011).

- ▶ *Medidas directas*: Se obtienen aplicando reglas de medición directamente al fenómeno de interés. Por ejemplo, contando las líneas de

código en un programa de computador utilizando reglas de conteo bien definidas. En la tabla 6.1, se presenta un listado de medidas directas para varios tipos de mediciones.

Tabla 6.1. Mediciones junto con sus medidas directas

Medición	Medida directa
Tamaño del software	Líneas de código.
Número de personas por categoría	Número de programadores; número de <i>testers</i> .
Progreso	Número de requisitos en línea base, número de requisitos modificados, número de módulos en línea base, número de defectos encontrados, número de defectos corregidos, número de semanas para lograr un hito.
Utilización de recursos	Ciclos de Central Processing Unit (CPU) utilizados, número de <i>bytes</i> de memoria utilizados.
Tiempo	Número de semanas para lograr un hito.
Calidad	Número de defectos corregidos, tiempo de respuesta del computador.

Fuente: elaboración propia con base en Fairley (2011).

- *Medidas derivadas o indirectas:* Se obtienen combinando medidas directas utilizando operaciones apropiadas en una determinada escala de medición. Ejemplo: conteo de cuántas veces ocurrió un evento, la cantidad de defectos encontrados en un producto de software, el costo y esfuerzo aplicado, y la productividad de un equipo de software (**LOC**/tiempo de desarrollo). La eficacia de un proceso solo puede evaluarse de manera indirecta. En la tabla 6.2, se presenta un listado de medidas indirectas de acuerdo con la medición.

Las buenas métricas deben facilitar el desarrollo de modelos que sean capaces de predecir el proceso, no solo describirlos. Por tanto, las métricas ideales deberían caracterizarse por ser (Mills y Shingler, 1988):

- *Simples y precisamente definibles:* Debe ser claro cómo la métrica puede ser evaluada.
- *Objetivas:* En la mayor medida posible.
- *Fácilmente obtenibles:* A un costo razonable.
- *Válidas:* Las métricas deben medir lo que intentan medir.

- *Robustas*: Relativamente insensibles a cambios insignificantes en el proceso.

Tabla 6.2. Mediciones y medidas indirectas

Medición	Medida indirecta
Tamaño	Puntos de función.
Productividad	Líneas de código desarrolladas por desarrollador por mes, puntos de función implementados por programador por semana.
Tasas de producción	Líneas de código por mes, puntos de función por semana.
Tasa de pruebas	Pruebas realizadas por personal por día.
Densidad de defectos	Defectos por miles de líneas de código, defectos por punto de función.
Eficiencia de defectos	Número de defectos corregidos por personal por día.
Efectividad de defectos	Número de defectos detectados/Número de defectos totales.
Estabilidad de requisitos	Número actual de requisitos/Número inicial de requisitos/Número actual de requisitos/Número más reciente de requisitos.
Índice de costo desempeño	Costo real/Costo presupuestado.

Fuente: elaboración propia con base en Fairley (2011).

Las métricas se pueden clasificar también como públicas o privadas. Generalmente, son privadas a nivel individual y son públicas a nivel grupal u organizacional. Las métricas de producto son privadas a nivel individual, pero se combinan para generar métricas de proyecto públicas para equipos o grupos. Por último, las métricas de proyecto se consolidan para generar métricas de proceso públicas para la organización (Pressman, 2005). La medición se puede realizar en cuatro escalas (Bourque y Fairley, 2014):

- *Nominal*: Es el nivel más bajo de medición y representa la acción de números en general. Los números se utilizan como etiquetas, pero también se pueden emplear palabras o letras. Esta escala consiste en colocar las unidades de muestreo en cualquier categoría mutuamente excluyente. Los nombres de las categorías no están relacionados entre sí. Entre algunos ejemplos, se encuentran:
 - Etapas del proceso de software (análisis, diseño, codificación, pruebas, etc.).
 - Roles en un determinado proceso de software. Por ejemplo, administrador del proyecto, analista, diseñador, etc.

- Un proyecto puede tener diez implementadores y cuatro *testers*; se puede decir que hay seis implementadores más que *testers*, pero nada sobre los niveles de habilidad de los implementadores o de los *testers*.
- *Ordinal*: Los diferentes valores obtenidos a través del proceso de medición tienen un ordenamiento implícito. Los intervalos entre valores no se especifican. Se pueden utilizar rangos como {bajo, medio, alto} con las operaciones relacionales transitivas permitidas de menor que (<), mayor que (>) e igual (=). Ejemplos:
 - Niveles de madurez en el Capability Maturity Model Integration (**CMMI**): {inicial, gestionado, definido, gestionado cuantitativamente y mejorado}.
 - Niveles de destreza de un desarrollador: {alto, medio, bajo}.
- *Intervalo*: Son medidas que tienen intervalos iguales entre dos rangos adyacentes determinados arbitrariamente. Pueden incluir un valor cero, pero es arbitrario. Es una escala cuantitativa en la que las medidas estadísticas habituales se pueden aplicar. La diferencia entre los niveles de atributos se puede calcular y es significativa. Ejemplo:
 - Fechas calendario para las etapas de un proyecto.
 - Escala Likert en encuestas y formularios.
- *Ratio*: Son mediciones que tienen intervalos iguales entre dos rangos adyacentes y un elemento cero que denota la ausencia de la cantidad que se mide. La existencia de un cero no arbitrario es obligatoria. Esta escala contiene operaciones para determinar las relaciones de igualdad, orden de clasificación, igualdad de intervalos e igualdad de razones. Ejemplos:
 - **LOC** de un programa.
 - Esfuerzo de desarrollo por fase.
- *Absoluta*: Las mediciones permiten relaciones con intervalos iguales. Incluye un elemento “cero” determinado objetivamente que denota la ausencia del fenómeno. Por ejemplo:
 - La densidad de defectos como un número entero de defectos (D) por líneas de código (**LOC**), midiendo el número total de defectos y dividiendo por el número de líneas de código en una escala de medición de ratio. Además, los defectos por línea de código se pueden convertir en defectos por mil líneas de código mediante la fórmula: $(D/LOC) * 1000$.

En la tabla 6.3, se pueden observar las cuatro escalas de medición junto con sus posibles operaciones y descripción de datos.

Tabla 6.3. Características de las escalas de medición

Tipo de escala	Características	Posibles operaciones	Descripción de datos
Nominal	Distribuciones de frecuencia entre categorías de medición	=, ≠	Categorías
Ordinal	Orden dentro de categorías, intervalos arbitrarios entre categorías	<, >	<i>Rankings</i>
Intervalo	Intervalos iguales entre medidas, elemento cero determinado arbitrariamente	+, −	Diferencias
Ratio	Intervalos iguales entre medidas, elemento cero determinado objetivamente	/	Cero absoluto
Absoluta	Similar a la escala ratio, pero con unicidad de medidas	/	Unicidad

Fuente: elaboración propia con base en Fairley (2011) y Mills y Shingler (1988).

Los métodos de medición definen actividades que asignan un valor o un símbolo a un atributo de una determinada entidad. El método de medición se define como una operación especificada con precisión que produce un número (resultado de la medición) al medir un atributo. La arbitrariedad en el método se reflejará en ambigüedad en los resultados de la medición (Bourque y Fairley, 2014).

Los métodos de medición deben ser confiables y válidos (Bourque y Fairley, 2014).

- ▶ *Confiabilidad*: La confiabilidad de un método de medición es la medida en que la aplicación del método de medición produce resultados de medición consistentes. Se refiere a la consistencia de los valores obtenidos cuando el mismo elemento se mide varias veces.
- ▶ *Validez*: Se refiere a si el método de medición realmente mide lo que pretende medir. La validez de un método de medición puede considerarse desde tres perspectivas: validez de constructo (el método de medición prueba el concepto que está destinado a medir), validez de criterio (los resultados corresponden a una prueba dife-

rente de este concepto) y validez de contenido (el método de medición es totalmente representativo de lo que se pretende medir).

Las mediciones son la base para detectar desviaciones en el rendimiento aceptable. También son la base para identificar oportunidades de mejora de procesos. Los objetivos clave de la medición de procesos son (Florak *et al.*, 1997):

- Recolectar datos que midan el desempeño de cada proceso.
- Analizar el desempeño de cada proceso.
- Conservar y utilizar los datos para:
 - Evaluar la estabilidad y capacidad del proceso.
 - Interpretar los resultados de observaciones y análisis.
 - Predecir costos y desempeños futuros.
 - Proporcionar líneas de base y puntos de referencia.
 - Detectar tendencias en las secuencias de datos.
 - Identificar oportunidades de mejora.

Las medidas se deben planificar y luego aplicar. La planificación de la medición se basa en una comprensión del proceso de software definido. Se identifican los problemas y atributos relacionados con el producto, el proceso y los recursos, y se seleccionan y definen medidas de calidad tanto del producto como del proceso. Aplicar las medidas se refiere a que durante la ejecución del proceso los datos del proceso de software y de los productos producidos se recopilan y analizan para que se puedan utilizar en el control y la mejora del proceso.

► 6.2. Ontología de medición

Ruiz *et al.* (2003) han propuesto una ontología para que investigadores y practicantes estén de acuerdo con los conceptos utilizados en medición de software. Una ontología representa una cierta visión de un dominio de aplicación, en el cual los conceptos de dominio son definidos de una forma explícita y sin ambigüedades (Deridder, 2002). La ontología propuesta fue desarrollada utilizando Unified Modeling Language (**UML**) como lenguaje de representación ontológica y una adaptación de **REFSENO** (Tautz y Von Wangenheim, 1998).

En la figura 6.1, se presenta el diagrama de clases **UML** de la ontología para la medición de software. La medición en la ontología propuesta

consiste de estructuras y relaciones entre “necesidades de información” de “objetos medibles” y sus elementos relevantes, los cuales permiten obtener “productos de información” que satisfacen las necesidades de información.

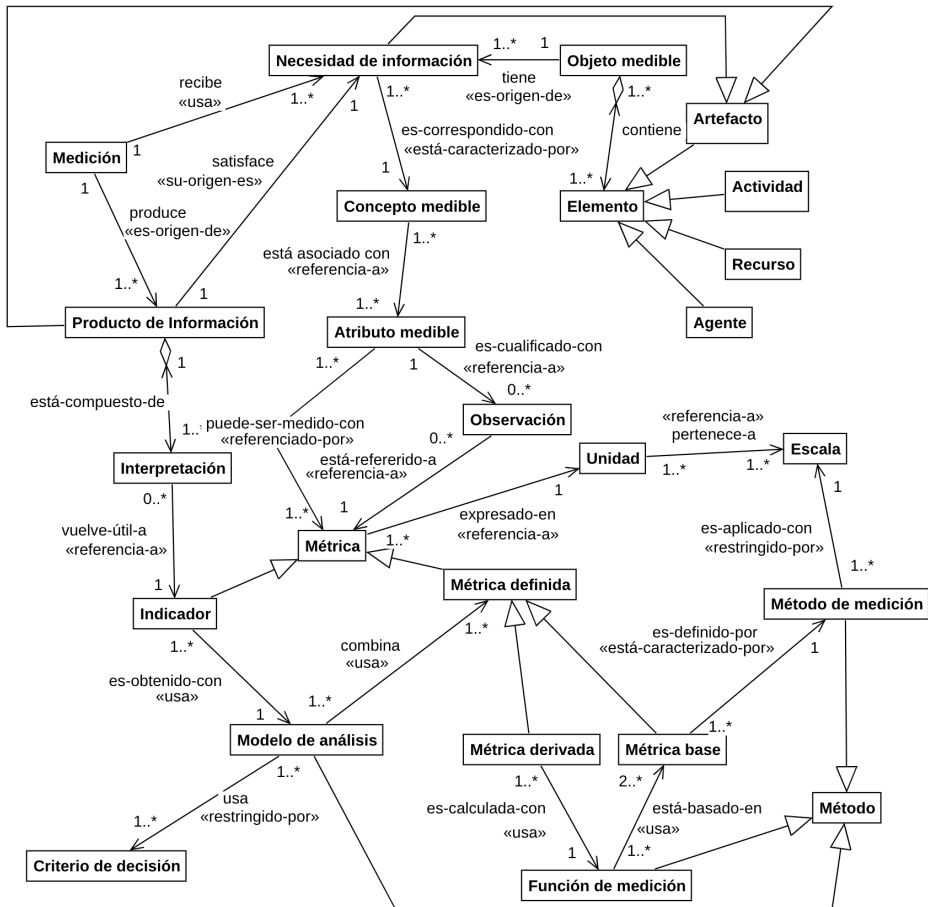


Figura 6.1. Diagrama de clases UML para la ontología de medición de software

Fuente: elaboración propia con base en Ruiz *et al.* (2003) desarrollado con la herramienta StarUML.

El punto de partida es un “concepto medible”, el cual contiene “atributos medibles” relacionados con las necesidades de información. Los atributos pueden ser medidos por medio de “métricas”. Cada “observación” permite obtener valores de un atributo en un momento dado para cada una de las

métricas asociadas. Cada métrica tiene una “unidad de medición” que pertenece a una “escala” específica. Cada métrica tiene un “valor por defecto”.

Las observaciones pueden ser directas o indirectas. Las observaciones directas utilizan “métricas base” asociadas con un atributo. Cada métrica base está definida mediante un “método de medición” específico. Los métodos pueden ser objetivos o subjetivos. Las observaciones indirectas utilizan “métricas derivadas” e “indicadores”. Las métricas derivadas están conformadas a partir de métricas base y sus valores son determinados por medio de “funciones de medición”. Los indicadores permiten verificar el cumplimiento de las necesidades de información. Los valores de un indicador pueden ser cuantitativos o cualitativos. Los valores de un indicador son calculados utilizando un “modelo de análisis” que utiliza “criterios de decisión”.

Una “interpretación” consiste de una explicación sobre los resultados de una observación de un indicador. Un “producto de información” es el resultado de un proceso de medición que satisface una necesidad de información.

La tabla 6.4 presenta el glosario de conceptos de la ontología, la tabla 6.5 muestra los atributos de los conceptos de la ontología, la tabla 6.6 presenta las relaciones de la ontología y la tabla 6.7 describe las clases de relaciones en la ontología de medición.

Tabla 6.4. Glosario de conceptos de la ontología de medición de software	
Concepto	Descripción
Actividad	Acción que se debe realizar para lograr los objetivos del objeto medible.
Agente	Persona, organización o producto de software que desempeña un papel activo en la realización de un objeto medible.
Modelo de análisis	Algoritmo o cálculo que combina una o más métricas definidas con ciertos criterios de decisión para obtener el valor de un indicador y su interpretación.
Artefacto	Parte de un objeto medible que es creado o modificado por las actividades.
Métrica base	Métrica definida en función de un atributo y su método de medición para obtener su valor. Sinónimo: métrica directa.
Criterio de decisión	Valores de umbral, objetivos o patrones utilizados para determinar la necesidad de una acción o para describir el valor de una métrica. Ayuda a interpretar los resultados de la medición.
Métrica definida	Métrica para la cual existe una fórmula matemática a fin de obtener su valor. Puede ser una métrica base o una métrica derivada.

Concepto	Descripción
Métrica derivada	Métrica definida con base en dos o más métricas base. Captura información sobre más de un atributo o sobre el mismo atributo para más de un elemento.
Elemento	Los elementos de un objeto medible pueden ser artefactos, actividades, recursos o agentes que pueden caracterizarse midiendo algunos de sus atributos.
Indicador	Métrica que proporciona una evaluación de algunos atributos, respetando ciertas necesidades de información. Es la base para tomar decisiones y analizar un objeto medible. Su valor puede ser cuantitativo o cualitativo.
Necesidad de información	Artefacto de tipo documento que permite satisfacer cierta información, incluidas las interpretaciones de las observaciones de uno o más indicadores.
Interpretación	Artefacto de tipo de documento que describe la información necesaria para gestionar un objeto medible.
Objeto medible	Objeto que se puede medir, como proyectos, procesos, productos y servicios.
Concepto medible	Idea que establece una relación abstracta entre atributos y necesidades de información. Por ejemplo: calidad, rendimiento, madurez, etc.
Función de medición	Algoritmo realizado para combinar valores o métricas base a fin de obtener el valor de una métrica derivada. La unidad y la escala de una métrica derivada dependen de cuál fue la función de medición.
Medición	Proceso que consiste en realizar las actividades necesarias para obtener los productos de información que satisfacen las necesidades de información producidas durante la gestión de un objeto medible.
Método de medición	Método que consiste en una secuencia lógica de operaciones, utilizado para cuantificar un atributo en una determinada escala. Pueden ser subjetivos (basados en el juicio humano) u objetivos (basados en reglas numéricas).
Atributo medible	Propiedad o característica de un elemento que se puede distinguir tanto cualitativa como cuantitativamente.
Método	Proceso sistemático que incluye los pasos y la heurística para permitir la realización de una o más actividades.
Métrica	Variable asociada a uno o más atributos que permite su calificación a través de observaciones (asignación de valores). Puede ser una métrica base, una métrica derivada o un indicador. Sinónimo: medida.
Observación	Acción que ocurre en un instante o un punto en el tiempo que permite obtener el valor de un atributo referido a una métrica.
Recurso	Algo que no es de características humanas, que es necesario para realizar una actividad.
Escala	Conjunto ordenado de valores continuos o discretos o un conjunto de categorías a las que se asocia una métrica para un atributo.
Unidad	Cantidad definida o adoptada por convención, utilizada para comparar cantidades de este tipo.

Fuente: elaboración propia con base en Ruiz *et al.* (2003).

Tabla 6.5. Atributos de la ontología de medición de software

Concepto	Atributo	Descripción
Artefacto	Entregable	Indica si el artefacto debe entregarse a un cliente.
	Edad	Número de años de existencia.
	Tipo	Características propias del artefacto. Por ejemplo: documento, módulo, componente, archivo, etc.
	Calidad	Medida cualitativa de la calidad.
Indicador	Nivel de exactitud	Una cuantificación de la incertidumbre o precisión inherente a la forma de obtener el valor de un indicador.
	Tipo	Características propias del valor obtenido para un indicador. Puede ser numérico cuantitativo o cualitativo.
Método de medición	Tipo	Indica las características propias de las operaciones utilizadas para cuantificar un atributo. Existen dos tipos: subjetivo (cuantificación influenciada por el juicio humano) y objetivo (cuantificación basada en reglas numéricas).
Métrica	Valor por defecto	Valor inicial que tiene una métrica antes de realizar cualquier observación de esta métrica.
Observación	Precisión	Indicador de la precisión que tiene un valor de una observación.
	Punto de tiempo	Momento o instante de tiempo cuando se realiza la observación.
	Tipo	Indica si una observación es directa (asociada a una métrica base) o indirecta (asociada a una métrica derivada o a un indicador).
	Valor	Resultado cuantitativo o cualitativo asignado a una métrica (base, derivada o indicador). Sinónimo: datos.
Escala	Tipo	Indica las características de la relación entre los valores de una escala. Puede ser: nominal, ordinal, intervalo y relación.

Fuente: elaboración propia con base en Ruiz *et al.* (2003).

Tabla 6.6. Relaciones en la ontología de medición de software

Nombre	Descripción
Vuelve útil a	Una interpretación proporciona utilidad sobre una necesidad concreta a un indicador.
Pertenece a	Una unidad de medida pertenece al conjunto de valores de una o más escalas (todas ellas de este tipo).
Puede ser medido con	Un atributo medible se puede medir a través de varias métricas.
Combina	Un modelo de análisis combina los valores de una o más métricas definidas (métricas derivadas o base) para obtener el valor de un indicador.
Contiene	Un objeto medible puede contener una colección de elementos de diferentes tipos, como artefactos, actividades, recursos y agentes.
Expresado en	Los resultados obtenidos con una métrica se expresan en una unidad de medida.
Tiene	Un objeto medible tiene algunas necesidades de información que son necesarias para satisfacer su correcta gestión.
Incluye	Un elemento relevante de un objeto medible incluye uno o más atributos medibles.
Interpreta	Una interpretación explica el resultado de una observación.
Es aplicado con	Un método de medición es aplicable con una escala específica.
Está asociado con	Un concepto medible está asociado con uno o más atributos medibles.
Está basado en	La función de medición utilizada para calcular un valor métrico se basa en el valor de dos o más métricas base.
Es calculado con	El valor de una métrica derivada se calcula con una función de medición específica.
Es calificado con	Un atributo medible se califica con una o más observaciones.
Está compuesto de	Un producto de información se compone de una o más interpretaciones.
Es correspondiente con	Una necesidad de información concreta de un objeto medible se corresponde con un concepto medible.
Está definido por	Una métrica base se define en función del método de medición utilizado para obtener su valor.
Es obtenido con	El valor, el nivel de precisión y la interpretación de un indicador se obtienen con un modelo de análisis específico.
Está referido a	Una observación obtiene el valor de un atributo referido a una métrica.
Produce	El proceso de medición produce productos de información como resultado de sus actividades.
Recibe	El proceso de medición utiliza las necesidades de información de un objeto medible como entrada de sus actividades.
Satisface	Un producto de información satisface una necesidad de información.
Usa	Un modelo de análisis utiliza uno o más criterios de decisión.

Fuente: elaboración propia con base en Ruiz *et al.* (2003).

Tabla 6.7. Clases de relaciones en la ontología de medición

Clase de relación	Nombre inverso	Propósito
Caracteriza a	Es caracterizado por	Una instancia de un concepto identifica las instancias de otro concepto.
Es origen de	Su origen es	Una instancia de un concepto se origina como consecuencia y después de la existencia de otra instancia de otro concepto.
Refiere a	Referenciado por	La definición de una instancia de un concepto se hace en referencia a instancias de otro concepto.
Restringe a	Restringido por	Los posibles estados de una instancia de un concepto están limitados por instancias de otro concepto.
Usa	Es usado por	Una instancia de un concepto usa instancias de otro concepto.

Fuente: elaboración propia con base en Ruiz *et al.* (2003).

Para entender los conceptos de la ontología de medición, se propone a continuación un ejemplo de cierta necesidad de información que será resuelta aplicando la ontología.

- *Necesidad de información:* Evaluar la efectividad de las pruebas unitarias realizadas.
- *Concepto medible:* Efectividad.
- *Objeto de medición:* Código fuente.
- *Atributos medibles:* Tamaño, defectos y tiempo.
- *Observaciones:* Mediciones realizadas al finalizar la codificación de una clase.
- *Métricas base:*
 - Tamaño de la clase.
 - Cantidad de defectos encontrados.
 - Tiempo gastado en encontrar los defectos.
- *Métodos de medición:*
 - Para el tamaño de la clase, el método de medición es un conteo manual o automatizado basado en un estándar de conteo de LOC.
 - Para la cantidad de defectos encontrados, el método es la realización de las pruebas sobre cada uno de los métodos de la clase y el registro de los defectos encontrados.

- Para el tiempo gastado en encontrar los defectos, el método de medición es cronometrar el tiempo gastado en realizar las pruebas.
- ▶ Escalas:
 - Para el tamaño de la clase en **LOC**, la escala es ratio.
 - Para la cantidad de defectos encontrados, la escala es absoluta.
 - Para el tiempo gastado en encontrar los defectos, la escala es absoluta.
- ▶ Unidad de medición:
 - Para el tamaño de la clase, la unidad de medición es la cantidad en miles de líneas de código fuente (**KLOC**).
 - Para la cantidad de defectos encontrados, la unidad de medición es la cantidad de defectos encontrados (D).
 - Para el tiempo gastado en encontrar los defectos, la unidad de medición es minutos (T).
- ▶ Métricas derivadas:
 - Densidad de defectos: Cantidad de defectos por miles de líneas de código de la clase (DD).
 - Eficiencia en la eliminación de defectos: Tasa de eliminación de defectos por unidad de tiempo (E).
- ▶ *Función de medición*:
 - Densidad de defectos: $DD = D / KLOC$.
 - Eficiencia en la eliminación de defectos: $E = D / T * 60$.
- ▶ Indicador:
 - Efectividad de pruebas unitarias.
- ▶ *Modelo de análisis*: El modelo de análisis se expresa por medio de una tabla que presenta diferentes rangos permitidos de valores de la densidad de defectos y la eficiencia de eliminación de defectos, como se puede observar en la tabla 6.8.
- ▶ *Interpretación*: Depende de los valores reales obtenidos en un proyecto real.
- ▶ *Producto de información*: Evalúa si la necesidad de información ha sido satisfecha.

Tabla 6.8. Ejemplo de modelo de análisis

Densidad de defectos	Eficiencia de eliminación de defectos	Eficiencia de pruebas (criterio de decisión)
DD <= 15 defectos/KLOC	E <= 10 defectos/hora	Baja
DD <= 15 defectos/KLOC	E > 10 defectos/hora	Media
DD > 15 defectos/KLOC	E <= 10 defectos/hora	Alta
DD > 15 defectos/KLOC	E > 10 defectos/hora	Muy alta

Fuente: elaboración propia con base en Ruiz *et al.* (2003).

► 6.3. Métricas de software

Una métrica de software es una característica de un sistema de software, documentación del sistema de software o del proceso de desarrollo que puede medirse de manera objetiva (Sommerville, 2015).

Las métricas de software pueden ser de tres tipos (Pressman, 2005; Sommerville, 2015):

- *Métricas del proceso:* Apoyan la gestión del proceso. Se recopilan a través de la realización de los proyectos de desarrollo de software y durante largos periodos. También se les denomina métricas de control. Las métricas del proceso se utilizan con propósitos estratégicos. Por ejemplo:
 - Esfuerzo y tiempo para corregir defectos encontrados.
 - Tiempo general de desarrollo.
 - Tipo de metodología utilizada.
 - Nivel promedio de experiencia del personal de desarrollo de software.
- *Métricas del proyecto:* Permiten evaluar el estado de los proyectos, valorar riesgos y ajustar el flujo de tareas de los proyectos. Cada proyecto debe medir sus entradas (recursos requeridos para realizar el trabajo), salidas (entregables o productos de trabajo creados) y resultados (efectividad de los entregables). Las métricas del proyecto se utilizan con propósitos tácticos. Por ejemplo:
 - El valor ganado planificado acumulado y el valor ganado real acumulado de un proyecto.

- *Métricas del producto*: Se asocian al propio producto de software desarrollado y a cualquier etapa de su desarrollo, desde los requisitos hasta el sistema instalado. Estas métricas ayudan a predecir las características del producto de software, por ello, también se denominan métricas de predicción. Ejemplos:
 - La complejidad ciclomática de un módulo de software.
 - El número de atributos y operaciones de una clase en un sistema orientado a objetos.

En la figura 6.2, se muestran las diferentes métricas y sus relaciones entre ellas en el contexto de una empresa desarrolladora de software. La medición tanto del proceso, de los proyectos y de los productos permitirá tomar decisiones administrativas que controlarán y mejorarán el proceso de software.

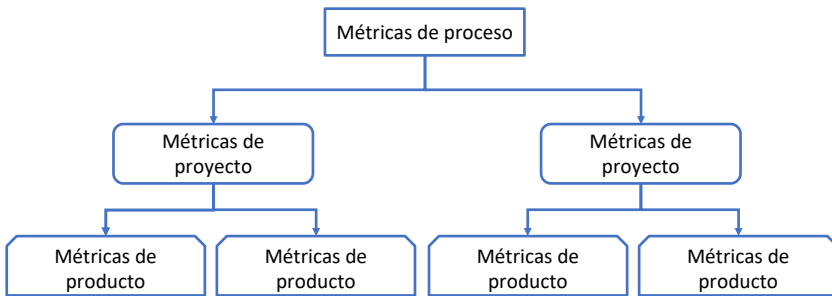


Figura 6.2. Métricas de proceso, proyecto y producto

Fuente: elaboración propia con base en Piattini *et al.* (2011).

Otra categorización de las métricas de software puede ser métricas objetivas o subjetivas (Mills y Shingler, 1988):

- *Métricas objetivas*: La recolección o medición siempre deben dar lugar a valores idénticos para una métrica dada, medida por dos o más observadores calificados. Un ejemplo de métrica de proceso objetiva es el tiempo de desarrollo.
- *Métricas subjetivas*: Incluso los observadores calificados pueden medir diferentes valores para una métrica dada, ya que su juicio subjetivo está involucrado en llegar al valor medido. Un ejemplo de métrica de proceso subjetiva es el nivel de experiencia de un programador.

► 6.4. Paradigma GQM

El paradigma objetivo-preguntas-métricas (Goal-Question-Metric [GQM]) es un enfoque que se puede utilizar para determinar las mediciones que se deben realizar en un proyecto de software (Basili *et al.*, 1994). GQM obtiene como resultado final un conjunto de métricas que responde a las siguientes preguntas:

- *Objetivos*: ¿Qué se desea lograr?
- *Preguntas*: ¿Qué preguntas se deben responder para asegurar que se logren los objetivos?
- *Métricas*: ¿Qué datos se deben recopilar y analizar para responder a las preguntas?

Por tanto, se obtiene una estrategia jerárquica de descomposición de los objetivos de medición, como se observa en la figura 6.3.

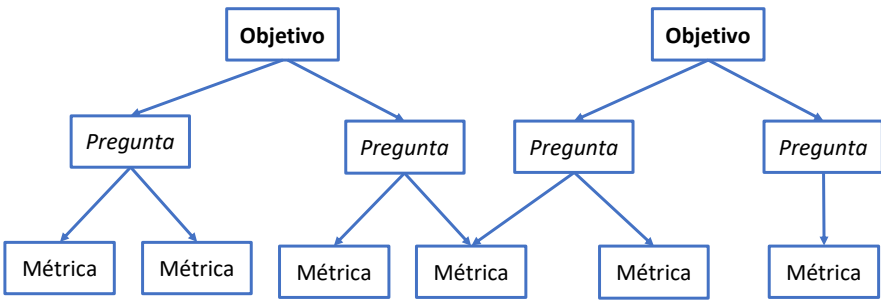


Figura 6.3. Jerarquía GQM

Fuente: elaboración propia.

A continuación, se presenta un ejemplo de la aplicación de GQM. En este caso, uno de los objetivos es reducir los defectos:

- *Objetivo*: Reducir defectos durante el desarrollo de software.
- *Preguntas*:
 - ¿Cuántos defectos se introducen durante el desarrollo del software?
 - ¿Qué porcentaje del total de defectos encontrados durante el desarrollo se introduce en cada fase del desarrollo del software?
 - ¿Qué porcentaje del total de defectos encontrados durante el desarrollo se elimina en cada fase del desarrollo del software?

- ¿Qué tipo de defectos se encuentran en cada fase del desarrollo de software?
- ¿En qué porcentaje deberían reducirse los defectos encontrados durante el desarrollo del software en los próximos 12 meses?
- ▶ Métricas:
 - Número total de defectos encontrados durante el desarrollo del software.
 - Porcentaje de defectos totales introducidos en cada fase de desarrollo de software.
 - Porcentaje de defectos totales encontrados en cada fase del desarrollo de software.
 - Tipos de defectos encontrados en cada fase de desarrollo de software.

▶ 6.5. Medición del proceso

Se pueden definir cinco perspectivas que son fundamentales para la medición del proceso (Florak *et al.*, 1997):

- ▶ *Rendimiento*: Resultados de la ejecución del proceso con respecto a atributos medibles de calidad, cantidad, costo y tiempo.
- ▶ *Estabilidad*: Comportamiento estable y predecible del proceso.
- ▶ *Conformidad*: Ejecución fiel del proceso junto con una organización y un soporte adecuados.
- ▶ *Capacidad*: El proceso entrega productos que cumplen los requisitos.
- ▶ *Mejora e inversión*: Actividades que permiten mejorar el rendimiento del proceso, reducir su variabilidad, trasladar la media a niveles más rentables y reconocer que los cambios que se han introducido están funcionando.

A continuación, se ampliarán los conceptos relacionados con cada una de estas perspectivas de medición del proceso de software.

6.5.1. Rendimiento

El desempeño del proceso se puede describir de dos maneras (Florak *et al.*, 1997):

- *Medición de los atributos de los productos que genera el proceso:* Por ejemplo, las funcionalidades, el tamaño y la velocidad de ejecución del software, la potencia de un módulo. También incluye atributos de calidad (como facilidad de uso y fiabilidad).
- *Medición de los atributos del proceso:* Por ejemplo, la cantidad de esfuerzo gastado durante la ejecución del proceso, el tiempo necesario para realizar un proceso, los tamaños y la duración de los flujos de trabajo, las características y la cantidad de recursos utilizados, y los números, los tipos y las fuentes de defectos detectados o reparados.

Para medir el rendimiento del proceso, se miden tanto atributos de calidad del producto como del proceso, y se realizan en varios tiempos para obtener registros secuenciales de su comportamiento.

6.5.2. Estabilidad

Casi todas las características de los procesos y productos muestran variaciones cuando se miden a lo largo del tiempo. Esta variación tiene dos fuentes (Florak *et al.*, 1997):

- Fenómenos que son inherentes al proceso cuyos resultados son comunes en las mediciones de un atributo.
- Variaciones que tienen causas asignables que podrían haberse evitado.

Estos conceptos se pueden presentar en forma de ecuación de la siguiente manera:

$$[\text{variación total}] = [\text{variación de causa común}] + [\text{variación de causa asignable}]$$

La variación de causa común es la variación normal del proceso. Esta variación existe debido a las interacciones normales entre los componentes del proceso (personas, computadores, material y entorno). La variación de causa común se caracteriza por tener un patrón estable y consistente en el tiempo (figura 6.4 (a)). Los resultados de la variación de causa común son aleatorios, pero varían dentro de límites predecibles y no se alejan demasiado de los valores medios (Florak *et al.*, 1997).

Las variaciones debidas a causas asignables tienen impacto en las características del producto de software y otras medidas del desempeño del proceso. Estos impactos generan cambios significativos en la variación de los valores medidos (figura 6.4 (b)). Las variaciones de causa asignable surgen de eventos que no son parte del proceso normal, ya que representan cambios repentinos en las entradas, entorno o pasos del proceso (Florak *et al.*, 1997).

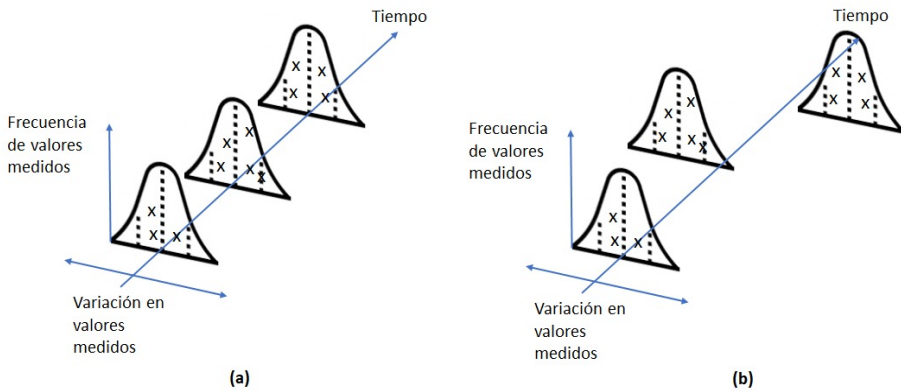


Figura 6.4. Concepto de variación controlada

Fuente: elaboración propia con base en Florak *et al.* (1997).

Se tiene un proceso estable cuando todas las causas asignables se han eliminado y se ha impedido que se repitan, obteniéndose un sistema constante de causas aleatorias. La estabilidad de un proceso con respecto a un atributo dado se determina midiendo el atributo y realizando un seguimiento de sus resultados a lo largo del tiempo.

6.5.3. Conformidad

El proceso se debe ejecutar de manera consistente. Existen tres aspectos de conformidad que pueden afectar significativamente el rendimiento del proceso (Florak *et al.*, 1997):

- ▶ *Aptitud para ejecutar el proceso:* Se aplica a personas, recursos, métodos, tecnologías y otros soportes del proceso. Por ejemplo, la falta de conocimiento sobre el proceso, capacitación insuficiente del personal, herramientas inadecuadas que permiten que el proceso no se ejecute correctamente.
- ▶ *Uso del proceso definido:* La estabilidad del proceso depende de una ejecución consistente del proceso. Si se determina que el proceso no está bajo control, una razón puede ser que el proceso no se está ejecutando como se definió.

- *Supervisión, benchmarking y evaluación del proceso*: Consiste en la supervisión del proceso a través de revisiones frecuentes y periódicas del estado del proceso, la evaluación formal de los procesos y la evaluación comparativa de proyectos.

Las medidas de conformidad no reflejan el grado en el que el proceso cumple su propósito. En su lugar, abordan los motivos por los cuales un proceso podría no estar funcionando como debería.

6.5.4. Capacidad

Cuando la variabilidad del proceso se ejecuta dentro de los límites permitidos, el proceso cumple con los requisitos del cliente. Si un proceso es estable y cumple con los requisitos, se denomina “capaz” o que “tiene capacidad”. El concepto de capacidad, por tanto, depende tanto de la estabilidad del proceso como de su capacidad para cumplir con los requisitos del cliente (Florak *et al.*, 1997).

Si la variabilidad del proceso se ejecuta fuera de los límites de la especificación, habrá ocasiones en las que el proceso no cumpla con sus requisitos. Para cambiar esta situación, debería reducirse la variabilidad, moverse la media del proceso o suavizar las especificaciones para alinear los requisitos y los límites del proceso. La figura 6.5 muestra las diferentes alternativas de variación de causa asignable.

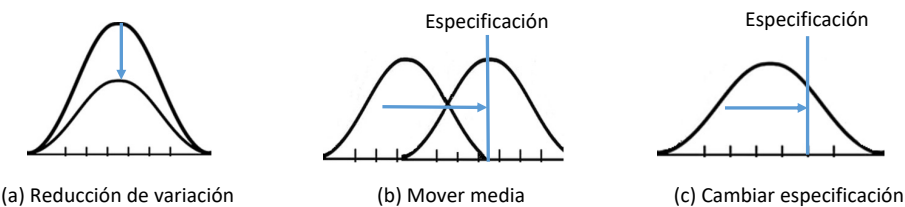


Figura 6.5. Concepto de variación de causa asignable

Fuente: elaboración propia con base en Florak *et al.* (1997).

6.5.5. Mejora e inversión

Si un proceso de software no satisface los requisitos del producto y las necesidades del negocio, o si una organización debe satisfacer las demandas

cada vez mayores de calidad y capacidad de respuesta del mercado, la organización se enfrentará a la necesidad de mejorar continuamente el rendimiento del proceso. Comprender la capacidad de los procesos de software es el primer paso para avanzar hacia la mejora del proceso (Florak *et al.*, 1997).

Además de indicadores financieros, se deben considerar medidas de satisfacción del cliente, innovación organizacional y ventaja competitiva como importantes impulsores del éxito futuro.

► 6.6. Estándar de medición del proceso ISO/IEC/IEEE 15939

El estándar **ISO/IEC/IEEE 15939: 2017** proporciona una elaboración del proceso de medición para el estándar **ISO/IEC 15288** e **ISO/IEC 12207**. El proceso de medición es aplicable a las disciplinas de gestión e ingeniería de sistemas y software. El proceso se describe a través de un modelo que define las actividades del proceso de medición que se requieren para especificar qué información de medición se requiere, cómo se deben aplicar las medidas y los resultados del análisis, y cómo determinar si estos son válidos. El proceso de medición es flexible y adaptable a las necesidades de diferentes usuarios (**ISO/IEC/IEEE 15939**, 2017).

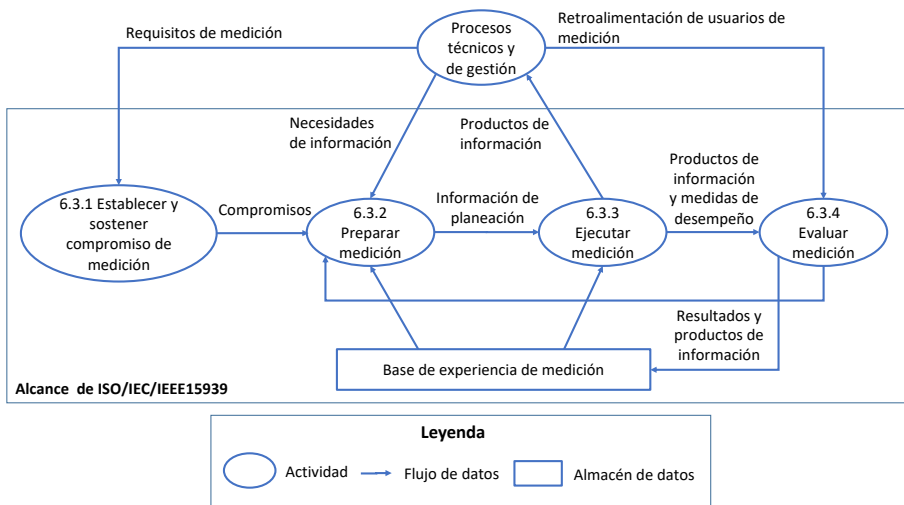


Figura 6.6. Modelo del proceso de medición

Fuente: elaboración propia con base en ISO/IEC/IEEE 15939 (2017).

El propósito del proceso de medición es recopilar, analizar e informar datos e información objetivos para respaldar una gestión eficaz y demostrar la calidad de los productos, servicios y procesos (ISO/IEC/IEEE 15288, 2015; ISO/IEC/IEEE 12207, 2017).

En la figura 6.6, se presenta el modelo del proceso de medición, el cual está conformado por las siguientes actividades: establecer y mantener el compromiso de medición, preparar, ejecutar y evaluar la medición.

En las tablas 6.9, 6.10, 6.11 y 6.12, se presentan las subtareas correspondientes a cada tarea del proceso de medición ISO/IEC/IEEE 15939.

Tabla 6.9. Establecer y mantener el compromiso de medición

Tareas	Subtareas
Aceptar los requisitos de medición	Identificar el alcance de la medición
	Establecer un compromiso de gestión
	Comunicar el compromiso con la unidad organizacional
Asignar recursos	Asignar la responsabilidad de la medición
	Proporcionar recursos para planificar, realizar y evaluar mediciones

Fuente: elaboración propia con base en ISO/IEC/IEEE 15939 (2017).

Tabla 6.10. Preparar la medición

Tareas	Subtareas
Definir la estrategia de medición	No incluye subtareas
Describir las características de la organización que son relevantes para la medición	No incluye subtareas
Identificar y priorizar las necesidades de información	Identificar necesidades de información
	Priorizar las necesidades de información identificadas
	Seleccionar las necesidades de información a ser dirigidas
	Registrar y comunicar las necesidades de información seleccionadas
Seleccionar y especificar medidas que satisfagan las necesidades de información	Identificar medidas candidatas que satisfagan las necesidades de información seleccionadas
	Seleccionar medidas a partir de las medidas candidatas
	Especificar medidas seleccionadas

Tareas	Subtareas
Definir procedimientos de recolección, análisis, acceso e informes de datos	Definir procedimientos para la recolección, almacenamiento y verificación de datos
	Definir procedimientos para el análisis de datos e informes de productos de información
	Definir procedimientos de gestión de configuración
Definir criterios para evaluar los elementos de información y el proceso de medición	Definir criterios para evaluar productos de información
	Definir criterios para evaluar el proceso de medición
Identificar y planificar los sistemas o servicios que se utilizarán	No incluye subtareas
Revisar, aprobar y proporcionar recursos para tareas de medición	Revisar y aprobar la planificación de mediciones
	Proporcionar recursos para implementar y evaluar tareas de medición
Adquirir e implementar tecnologías de soporte	Evaluar y seleccionar tecnologías de soporte apropiadas
	Adquirir e implementar tecnologías de soporte

Fuente: elaboración propia con base en ISO/IEC/IEEE 15939 (2017).

Tabla 6.11. Ejecutar la medición

Tareas	Subtareas
Integrar procedimientos para la generación, la recopilación, el análisis y los informes de datos en los procesos relevantes	Integrar la generación y recopilación de datos en los procesos relevantes
	Integrar el análisis de datos y los informes en los procesos relevantes
	Comunicar los procedimientos de datos a los proveedores de datos y partes interesadas
Recopilar, almacenar y verificar datos	Recolectar datos
	Almacenar los datos recopilados, incluida la información de contexto
	Verificar los datos recopilados
Analizar datos y desarrollar elementos de información	Analizar los datos recopilados
	Desarrollar elementos de información
	Revisar elementos de información
Registrar resultados e informar a los usuarios de la medición	Registrar resultados
	Informar a los usuarios de la medición

Fuente: elaboración propia con base en ISO/IEC/IEEE 15939 (2017).

Tabla 6.12. Evaluar la medición

Tareas	Subtareas
Evaluar los productos de información y el proceso de medición	Evaluar los productos de información según los criterios de evaluación especificados
	Evaluar el proceso de medición contra los criterios de evaluación especificados
	Almacenar lecciones aprendidas en la base de experiencia de medición
Identificar posibles mejoras	Identificar posibles mejoras a los productos de información
	Identificar posibles mejoras a la medición
	Comunicar mejoras potenciales

Fuente: elaboración propia con base en ISO/IEC/IEEE 15939 (2017).

► 6.7. Métricas de proceso de software

Las métricas de proceso de software son fundamentales para conocer el estado actual del proceso y serán la base para las actividades posteriores de ejecución, control y mejora del proceso. Hay varias razones para medir atributos de los procesos de software (Fairley, 2011):

- Proporcionar indicadores frecuentes de progreso.
- Proporcionar alertas tempranas de problemas.
- Permitir el análisis de tendencias para los proyectos.
- Permitir estimaciones del costo final y la fecha de finalización de los proyectos.
- Construir un repositorio de datos históricos de proyectos para la organización.

Además, las definiciones estandarizadas y las reglas de conteo para la medición de los procesos de software y los productos de trabajo son necesarias para proporcionar resultados de medición estandarizados en todos los proyectos dentro de una organización, poblar repositorios de datos y construir modelos predictivos basados en datos acumulados (Bourque y Fairley, 2014).

La medición del proceso de software está relacionada con la eficiencia y la efectividad de un proceso, actividad o tarea de software (Bourque y Fairley, 2014):

- ▶ *Eficiencia de un proceso, actividad o tarea de software*: Es la relación entre los recursos realmente consumidos y los recursos esperados o deseados que se consumirán al realizar un proceso, actividad o tarea de software:
 - El esfuerzo es la medida de recursos para la mayoría de los procesos, actividades y tareas de software. Se mide en unidades como horas-persona, días-persona, semanas-persona, meses de esfuerzo o en unidades monetarias equivalentes.
- ▶ *Efectividad del proceso*: Es la relación entre la salida real y la salida esperada producida por un proceso, actividad o tarea de software. Algunos ejemplos son la cantidad real de defectos detectados y corregidos durante las pruebas de software relacionada con la cantidad esperada de defectos que se detectarán y corregirán (con base en datos históricos para proyectos similares):
 - La productividad en la realización de un proceso, actividad o tarea es la proporción de salida producida dividida por los recursos consumidos. Ejemplos como la cantidad de defectos de software descubiertos y corregidos divididos por horas de esfuerzo por persona.

Pueden recopilarse tres tipos de métricas de proceso (Sommerville, 2015):

- ▶ *El tiempo que tarda en completarse un proceso particular*: Puede ser el tiempo total dedicado al proceso, tiempo calendario, tiempo empleado en el proceso por ciertos empleados en particular, etc.
- ▶ *Los recursos requeridos para un proceso particular*: Los recursos pueden incluir esfuerzo total en días-persona, costos de viaje o recursos de cómputo.
- ▶ *El número de ocurrencias de un evento particular*: Los ejemplos de eventos que pueden monitorizarse incluyen el número de defectos descubiertos durante la inspección del código, el número de cambios solicitados a los requisitos y el número promedio de líneas de código modificadas en respuesta a un cambio de requisitos.

En la tabla 6.13, se presenta un listado de entidades susceptibles de medición organizadas por su tipo y categorías. En la tabla 6.14, se presenta un listado de atributos medibles asociados con las entidades del proceso de software presentadas.

Tabla 6.13. Entidades medibles en un proceso de software

Tipo de entidades	Categorías de entidades	Entidades
Objetos recibidos o usados	Productos y subproductos	
	Ideas y conceptos	
	Recursos	Personal, instalaciones, herramientas, materias primas, energía, fondos, tiempo
	Guías y direcciones	Políticas, procedimientos, metas, restricciones, reglas, leyes, regulaciones, entrenamiento
Actividades y sus elementos	Procesos y controladores	Análisis de requisitos, diseño, codificación, pruebas, control de configuración, cambio de control, problemas
	Administración	Revisión, inspección, integración
	Rutas de flujo	Rutas de productos, rutas de recursos, rutas de datos, rutas de control
	Estructuras	Colas, pilas, contenedores
	Planificación	Logro de hitos
	Planificación vs. real	Variación de actividades planificadas vs. reales, variación de esfuerzo planificado vs. real, variación de tamaño del producto planificado vs. real
	Progreso	Productos de trabajo terminados, aceptados y en línea base
	Madurez del proceso	Se basa en el nivel CMMI alcanzado por la organización
Objetos consumidos	Recursos	Esfuerzo total y por fases, materias primas, energía, dinero, tiempo
Objetos mantenidos	Personal	
	Instalaciones	
	Herramientas	
	Materiales	
	Trabajo en proceso	
	Datos	
	Conocimiento	
	Experiencia	
Objetos producidos	Productos	Requisitos, especificaciones, diseños, módulos, casos de prueba, resultados de prueba, componentes probados, documentación, defectos, informes de defectos, peticiones de cambio
	Subproductos	Conocimiento, experiencia, habilidades, mejora del proceso, datos, <i>goodwill</i> , clientes satisfechos

Fuente: elaboración propia con base en Fairley (2011), Florak *et al.* (1997) y Xu *et al.* (2016).

Tabla 6.14. Atributos medibles asociados con entidades del proceso de software

Tipo de entidades	Entidades	Atributos
Objetos recibidos o usados	Cambios	Tipo, fecha, tamaño, cantidad de peticiones recibidas.
	Requisitos	Requisitos implementados, número de requisitos identificados, porcentaje de requisitos trazados al diseño y código.
	Problemas	Tipo, fecha, tamaño, origen, severidad, número de problemas recibidos.
	Fondos	Dinero, presupuesto, Estado
	Personal	Años de experiencia, tipo de educación, porcentaje de capacitación, códigos de empleo
	Herramientas	Accesibilidad, adecuación, utilidad, habilidades
	Instalaciones y medio ambiente	Metros cuadrados por empleado, nivel de ruido, Iluminación, número de personal en cubículos, de empleados compartiendo cubículo, inversión en herramientas por empleado, horas de uso del computador.
Actividades y sus elementos	Facilidades	Espacio, computadores, soporte técnico, suficiencia.
	Rutas de flujo	Tiempo de procesamiento, tasas de rendimiento, desviaciones, retrasos, atrasos.
	Longitud, tamaño	Colas, búfers, pilas.
	Riesgo	Estado de los factores de riesgo y actividades de mitigación (Fairley, 2011).
	Procedimientos	Cobertura, suficiencia, calidad, documentación.
	Plan de trabajo	Objetivos, <i>Work Breakdown Structure</i> (WBS), aplicabilidad, facilidad de entendimiento.
Objetos consumidos	Esfuerzo	Número de horas de desarrollo, de horas de retrabajo, de horas de soporte, de horas de reunión.
	Tiempo	Fecha y hora de inicio de tareas, fecha y hora de finalización de tareas, duración del proceso, tiempo de espera.
	Dinero	Costo hasta la fecha, variación de costo, costo de la reelaboración, gastos para diversos tipos de recursos.

Tipo de entidades	Entidades	Atributos
Objetos producidos	Estado de las unidades de trabajo	Número de unidades diseñadas, de unidades codificadas, de unidades probadas.
	Tamaño de las unidades de trabajo	Número de requisitos, de puntos de función, de líneas de código, de módulos, de objetos.
	Cantidad de salida	Número de aprobaciones, de defectos encontrados.
	Resultados de la prueba	Número de casos de prueba pasados, porcentaje de cobertura de las pruebas.
	Arquitectura del programa	<i>Fan in, fan out.</i>
	Cambios	Tipo (correctivo, perfectivo, etc.), fecha, tamaño, esfuerzo gastado.
	Problemas y defectos	Número de informes, densidad de defectos, tipo, origen, distribución por tipo, distribución por origen, número de problemas y defectos abiertos, de problemas y defectos cerrados.
	Atributos de calidad del producto	Defectos, confiabilidad, disponibilidad, tiempo de respuesta, rendimiento.
	Utilización de recursos críticos	Porcentaje de memoria, de capacidad de CPU, de capacidad de entrada/salida (E/S).

Fuente: elaboración propia con base en Fairley (2011) y Florak *et al.* (1997).

► 6.8. Métricas de modelos de procesos SPEM

En la literatura, se han propuesto diferentes métricas aplicadas a los modelos de procesos de software. En el capítulo cinco, se presentó una notación estandarizada denominada Software Process Engineering Metamodel (**SPEM**) para realizar esos modelos. Canfora *et al.* (2005) han propuesto una colección de métricas que cuantifican modelos de procesos descriptivos. Las métricas definidas están orientadas a las propiedades estructurales del modelo general del proceso de software. En la tabla 6.15, se presenta un conjunto de métricas representativas para modelos de procesos de software. Cabe anotar que esta colección de métricas está orientada a evaluar la facilidad de mantenimiento de los modelos de procesos de software.

Tabla 6.15. Métricas de modelos de procesos SPEM

Métrica	Definición
NA	Número de actividades del modelo de proceso de software.
NWP	Número de productos de trabajo del modelo de proceso de software.
NPR	Número de roles que participan en el proceso.
NDWPIIn	Número de dependencias de entrada de los productos de trabajo con las actividades en el proceso.
NDWPOut	Número de dependencias de salida de los productos de trabajo con las actividades en el proceso.
NDWP	Número de dependencias entre los productos de trabajo y las actividades. $NDWP(PM) = NDWPIIn(PM) + NDWPOut(PM)$
NDA	Número de dependencias de precedencia entre actividades.
NCA	Acoplamiento de actividades en el modelo de procesos. $NCA(PM) = NA(PM)/NDA(PM)$
RDWPIIn	Tasa entre dependencias de entrada de productos de trabajo con actividades y número total de dependencias de productos de trabajo con actividades. $RDWPIIn(PM) = NDWPIIn(PM)/NDWP(PM)$
RDWPOut	Tasa entre dependencias de salida de productos de trabajo con actividades y número total de dependencias de productos de trabajo con actividades. $RDWPOut(PM) = NDWPOut(PM)/NDWP(PM)$
RWPA	Tasa de productos de trabajo y actividades. $RWPA(PM) = NWP(PM)/NA(PM)$
RRPA	Tasa de roles de proceso y actividades. $RRPA(PM) = NPR(PM)/NA(PM)$

Fuente: elaboración propia con base en Canfora *et al.* (2005).

En la figura 6.8, se presenta un ejemplo de modelo de procesos de software representado con **SPEM**. La tabla 6.16 muestra los valores de las métricas obtenidos para el modelo representado en la figura 6.7.

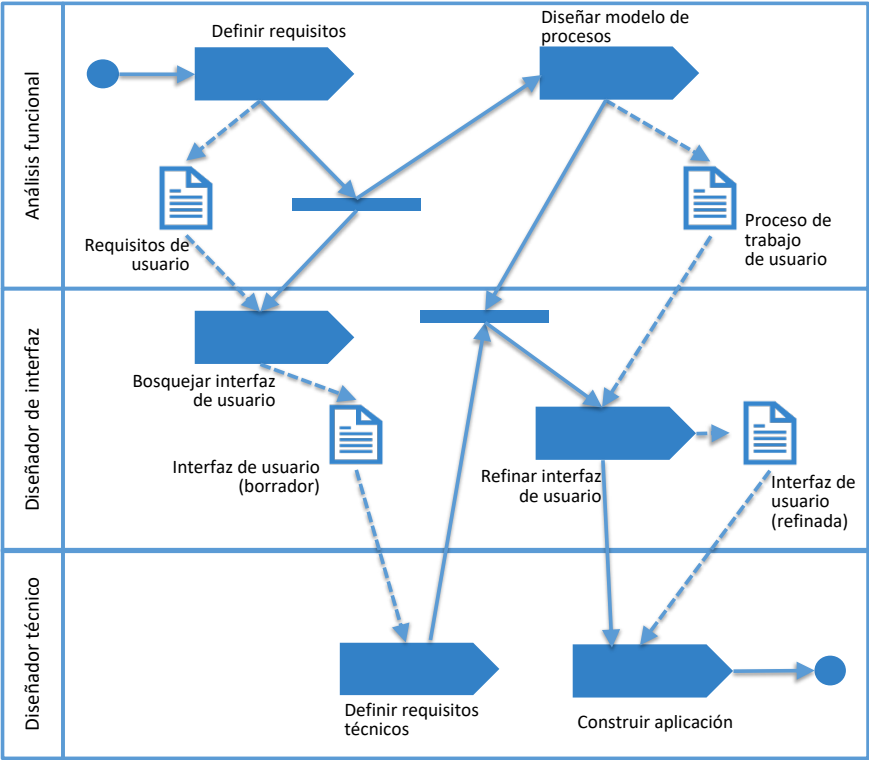


Figura 6.7. Ejemplo de un modelo de procesos SPEM

Fuente: elaboración propia con base en Menéndez Domínguez y Castellanos Bolaños (2015) desarrollado con Enterprise Architect.

Tabla 6.16. Valores obtenidos para las métricas del modelo de procesos SPEM

Métrica	Valor	Métrica	Valor	Métrica	Valor	Métrica	Valor
NA	6	NDWPIIn	4	NDA	7	RDWPOut	4/8 = 0,5
NWP	4	NDWPOut	4	NCA	6/7 = 0,857	RWPA	4/6 = 0,67
NPR	3	NDWP	8	RDWPIIn	4/8 = 0,5	RRPA	3/6 = 0,5

Fuente: elaboración propia con base en Canfora *et al.* (2005).

► 6.9. Técnicas de medición del proceso de software

Las técnicas de medición del proceso de software se utilizan para recopilar datos de proceso y datos de los productos de trabajo, transformar los datos en información útil y analizar la información para identificar las actividades de proceso que son candidatas para la mejora (Bourque y Fairley, 2014):

- *Técnicas cuantitativas*: Se utilizan para recopilar y analizar datos en forma numérica. A los procesos se les puede aplicar técnicas matemáticas y estadísticas. Se pueden utilizar herramientas básicas para el control de calidad a fin de analizar datos de medición cuantitativos (por ejemplo, hojas de verificación, diagramas de Pareto, histogramas, diagramas de dispersión, tablas de ejecución, tablas de control y diagramas de causa y efecto). Estas herramientas se describirán en el próximo capítulo.
- *Técnicas cualitativas*: Incluyen entrevistas, cuestionarios y juicios de expertos. Se pueden utilizar para complementar las técnicas de medición cuantitativas. Las técnicas de consenso grupal, incluida la técnica Delphi, se pueden utilizar para obtener un consenso entre grupos de *stakeholders*.

► Desafíos

Hay muchas ventajas en la medición del software a desarrollar junto con la identificación de críticas con respecto a su uso y aplicación. En general, se cree que los desarrolladores de software son reacios a aplicar mediciones del software y se resisten a realizar mediciones y utilizar métricas durante el desarrollo de proyectos.

Las principales limitaciones de las métricas de software en la industria del software están relacionadas con su automatización, la cual es una tarea difícil y la aplicación de las métricas puede ser difícil de implementar para diferentes entornos (Srinivasan, 2015).

La recolección de métricas no se puede realizar de manera manual debido al gran esfuerzo que se requiere en grandes proyectos. Por tanto, se requiere un proceso automatizado con herramientas de soporte adecuadas a los diferentes entornos de desarrollo. Las ejecuciones de métricas de software consumen mucho tiempo y son costosas y, a veces, arrojan conclusiones engañosas. Es difícil comprender los resultados de las métricas

de software, especialmente si involucran diferentes entornos y procesos de desarrollo. Además, la medición precisa de métricas requiere especialistas certificados en métricas de software en la industria.

Los programas de medición eficientes a veces fallan debido a tres cuestiones (Noor *et al.*, 2020):

- Conceptos erróneos del principio básico de medición y de los pasos que se deben realizar.
- La concepción errónea de información relacionada con la estimación de métricas se suma a los síntomas inesperados que socavan el crecimiento y el éxito de la organización.
- Desprecio por los factores humanos, es decir, los administradores a veces interpretan las medidas desde un punto de vista únicamente tecnológico y es posible que no consideren aspectos como grado de compromiso, enfoques mentales, experiencias y nivel de comunicación.

Los desafíos fundamentales para establecer un programa efectivo de uso y aplicación de métricas del proceso de software se orientan al establecimiento de unidades de medición consistentes en que el sistema de cálculo que se utilice no debe obtener recuentos diferentes que hagan difícil contrastar aspectos del proceso (Khurana, 2017). Además, los desarrolladores deben comprender qué información se recopila, por qué se recopila y cómo se utiliza, así como facilitar la participación de los desarrolladores en la creación de métricas.

► Ejemplos

Se presentarán algunos ejemplos de métricas orientadas al proceso ágil *scrum*. En este proceso, se realizan varios eventos en cada *sprint* para monitorear el desempeño del equipo. Las métricas están distribuidas por eventos de *scrum*: planificación del *sprint* (tabla 6.17), reuniones diarias (tabla 6.18), revisión del *sprint* (tabla 6.19) y retrospectiva del *sprint* (tabla 6.20).

Tabla 6.17. Métricas de planificación del *sprint*

Métrica	Proceso de medición	Descripción
Estimación de esfuerzo	Determinar puntos de historia por historia de usuario para planificar el <i>sprint</i> .	Tiene como objetivo estimar el esfuerzo requerido por <i>sprint</i> .
Puntos de historia	Describir el nivel de dificultad de una historia.	Proceso de medición para predecir el tamaño del software y esfuerzo requerido.
Esfuerzo de tareas	Estimar el esfuerzo para cada tarea en función de las horas del programador.	Unidad de medida para estimar el esfuerzo para implementar historias de usuario.
Fecha prevista y de fin de las tareas	Se determinan después de considerar la prioridad de una tarea en un <i>sprint</i> .	Tiempo fijo para completar una tarea.
Velocidad	Sumatoria de todos los trabajos terminados.	El número de trabajos completados durante el <i>sprint</i> .

Fuente: elaboración propia con base en Kurnia *et al.* (2018).

Tabla 6.18. Métricas de reuniones diarias

Métrica	Proceso de medición	Descripción
Contribución	Contar los integrantes que asisten y participan en la reunión.	Evaluar la participación y calidad del compromiso durante la reunión diaria.
Tasa de trabajo gastado y restante	Trabajo invertido en día <i>i</i> para cada tarea <i>j</i> en el trabajo pendiente del <i>sprint</i> /Trabajo restante en día <i>i</i> para cada tarea en el trabajo pendiente del <i>sprint</i> .	La relación entre el número de tareas (por día) realizadas y las tareas restantes (por día).
Incumplimiento de estándares	Realizar un seguimiento de los estándares incumplidos por <i>sprint</i> .	Los estándares de calidad del código que debe cumplir el equipo.
<i>Burndown chart</i>	Trabajo terminado al día en un <i>sprint</i> contra el número de horas de trabajo restantes.	Un gráfico que muestra cuánto trabajo queda en el <i>sprint</i> .

Fuente: elaboración propia con base en Kurnia *et al.* (2018).

Tabla 6.19. Métricas de revisión del *sprint*

Métrica	Proceso de medición	Descripción
Tiempo de corrección de defectos	Contar el tiempo de reparación de errores cometidos en un <i>sprint</i> .	Mide el tiempo que se tarda en corregir los errores encontrados durante el <i>sprint</i> .
Valor de negocio entregado	Medido en puntos o número de historias para describir el valor de negocio logrado.	Mide cuánto valor otorga la empresa a una historia de usuario.
Satisfacción del cliente	Basado en la demostración de características del producto.	Evaluación cualitativa de la satisfacción del cliente con base en determinados indicadores.
Defectos por iteración	Calculado todo defecto que se introdujo durante el <i>sprint</i> .	El defecto que se encontró y no debe acumularse en el <i>sprint</i> .
Entrega a tiempo	Proporción de funciones realizadas en el programa de lanzamiento planificado.	Indica el valor real para el cliente mediante la entrega de funciones específicas.
Cumplimiento del alcance	Relación entre el número de historias implementadas y planificadas.	Indica el compromiso acordado en la planificación del <i>sprint</i> .
Cantidad de historias	Se calcula como un conteo simple o peso por complejidad de la historia.	Seguimiento del progreso del proyecto en función del número de historias aceptadas.
Porcentaje de trabajo encontrado	Muestra el progreso del proyecto por estado de las tareas (no iniciada, en curso y completada).	Una herramienta para seguir el progreso de los miembros del equipo.
Capacidad de trabajo	El valor de la capacidad de trabajo debe ser igual o mayor que la velocidad.	El número de tareas reportadas durante el <i>sprint</i> con respecto a la función completada o no.

Fuente: elaboración propia con base en Kurnia *et al.* (2018).

Tabla 6.20. Métricas de retrospectiva del *sprint*

Métrica	Proceso de medición	Descripción
Gestión del valor ganado	Calcula el índice de rendimiento de cronograma y el índice de rendimiento de costos requeridos.	Una técnica para medir el desempeño y el progreso del proyecto de manera objetiva.
Satisfacción del trabajo	Realización de una encuesta relacionada con el trabajo realizado.	Satisfacción personal del desarrollador con su trabajo.

Fuente: elaboración propia con base en Kurnia *et al.* (2018).

► Resumen

La realización de mediciones es un aspecto fundamental para entender y mejorar el proceso de software. El proceso de medición inicia con la recolección de medidas sobre algún aspecto a medir, se agrupan en métricas y se generan indicadores para comprender y mejorar aspectos del proceso.

Las medidas pueden ser directas o indirectas. Las medidas directas se aplican directamente a la situación a medir. Las escalas indirectas combinan medidas directas utilizando operaciones apropiadas. Las escalas de medición pueden ser nominales (basadas en asignar etiquetas a los valores a medir), ordinales (tienen un ordenamiento implícito), intervalo (los rangos adyacentes entre dos intervalos tienen la misma distancia), ratio (como la escala de intervalo, pero con un elemento cero) o absoluta (permite relaciones entre las medidas).

Ruiz *et al.* (2003) definieron una ontología de medición basada en UML con los principales conceptos, atributos y relaciones subyacentes en la medición de software.

Las métricas de software se pueden clasificar en métricas del producto (asociadas al producto de software desarrollado), proyecto (valoran el estado de un proyecto) o proceso (se consolidan las métricas de proyectos permitiendo gestionar el proceso).

El paradigma GQM es un enfoque de medición basado en objetivos que se descomponen en preguntas y, a su vez, estas permiten definir métricas para responder a las preguntas definidas.

La medición del proceso de software se define con base en cinco perspectivas: rendimiento (cantidad, calidad, costo y tiempo del proceso), estabilidad (proceso predecible), conformidad (idoneidad del proceso), capacidad (proceso de entrega productos que satisfacen los requisitos) y mejora e inversión (mejorar la variabilidad del proceso y gestionar los cambios).

El estándar de medición ISO/IEC/IEEE 15939 establece un proceso de medición aplicable a la ingeniería de software que define cuatro actividades: establecer y sostener un compromiso de medición, preparar la medición, ejecutar la medición y evaluar la medición.

Con respecto a las métricas del proceso de software, se cuenta con un listado de entidades y atributos susceptibles de ser medidos y recolectados durante el desarrollo de software. Los tipos de entidades pueden ser objetos recibidos o usados, actividades y sus elementos, objetos consumidos, objetos mantenidos y objetos producidos.

La medición de modelos de procesos de software se puede realizar aplicando una colección de métricas definidas para modelos de procesos de software utilizando **SPEM**.

Por último, se pueden aplicar tanto técnicas cuantitativas como cualitativas para recolectar las métricas del proceso de software.

► Ejercicios

1. Explicar las diferencias conceptuales entre métricas de proceso y métricas de producto.
2. Determinar una colección de medidas directas e indirectas para la fase de análisis del proceso de desarrollo de software. Para las medidas determinadas, se debe indicar su tipo de escala.
3. Un proceso de software está compuesto de fases. Se desea evaluar el cumplimiento de sus fases de acuerdo con el tiempo estimado de desarrollo de cada una ellas. Establecer para esas fases diferentes medidas, métricas e indicadores que puedan contrastar el tiempo estimado con el tiempo real de desarrollo.
4. Aplicar el paradigma GQM para el cumplimiento de los siguientes objetivos:
 - Evaluar el cumplimiento del desarrollo de un elemento del *product backlog* durante la ejecución de un *sprint* en un proyecto *scrum*.
 - Mantener un proyecto con los tiempos de desarrollo reales iguales o menores a los estimados.
 - Reducir la cantidad de defectos detectados en la fase de pruebas de aceptación.
 - Maximizar la satisfacción del cliente.
5. Investigar métricas de proceso que se pueden aplicar a procesos ágiles como, *scrum* y eXtreme Programming (**xP**).
6. Determinar el valor de las métricas **NA**, **NWP**, **NPR**, **NDWPin**, **NDWPout**, **NDWP**, **NCA**, **RDWPin**, **RDWPout**, **NDA**, **RWPA**, **RRPA**, para los diagramas **SPEM** presentados en las figuras 5.6, 5.7, 5.8 y 5.9.
7. Describir las principales actividades del modelo de medición **ISO/IEC/IEEE 15939**.

► Referencias

- Basili, V. R., Caldiera, G. y Rombach, H. D. (1994). The goal question metric approach. En *Encyclopedia of software engineering* (pp. 528-532). Wiley & Sons Inc.
- Bourque, P. y Fairley, R. E. (eds.) (2014). *Guide to the software engineering body of knowledge, version 3.0*. **IEEE** Computer Society.
- Canfora, G., García, F., Piattini, M., Ruiz, F. y Visaggio, C. A. (2005). A family of experiments to validate metrics for software process models. *Journal of Systems and Software*, 77(2), 113-129. <https://doi.org/10.1016/j.jss.2004.11.007>
- Deridder, D. (2002). A concept-oriented approach to support software maintenance and reuse activities. En *Workshop on Knowledge-Based Object-Oriented Software Engineering at 16th European Conference on Object-Oriented Programming (ECOOP 2002)*. Springer. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7785ac1e75fc7b343776cbb98c598e1b1a0be565>
- Fairley, R. E. (2011). *Managing and leading software projects*. John Wiley & Sons.
- Farooq, S. U., Quadri, S. M. K. y Ahmad, N. (2011). Software measurements and metrics: Role in effective software testing. *International Journal of Engineering Science and Technology*, 3(1), 671-680. https://www.academia.edu/download/52482421/SOFTWARE_MEASUREMENTS_AND_METRICS_ROLE_I20170404-6019-9p9zbx.pdf
- Florak, W. A., Park, R. E. y Carleton, A. (1997). *Practical software measurement: Measuring for process management and improvement*. No. **CMU/SEI-97-HB-003**. Software Engineering Institute, Carnegie Mellon University. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=76aafd5d0ed49263488bca95f00f1fdad3729bec>
- Grady, R. B. (1992). *Practical software metrics for project management and process improvement*. Prentice-Hall.
- Institute of Electrical and Electronics Engineers. (1990). 610.12-1990 - **IEEE Standard Glossary of Software Engineering Terminology**. <https://doi.org/10.1109/IEEESTD.1990.101064>
- ISO/IEC/IEEE 12207**. (2017). **ISO/IEC/IEEE 12207:2017 Systems and software engineering - Software life cycle processes**.
- ISO/IEC/IEEE 15288**. (2015). **ISO/IEC/IEEE 15288:2015 Systems and software engineering - System life cycle processes**.
- ISO/IEC/IEEE 15939**. (2017). **ISO/IEC/IEEE 15939:2017 Systems and software engineering - Measurement process**.
- Joint Committee for Guides in Metrology. (2012). **JCGM 200:2012: International vocabulary of metrology. Basic and general concepts and associated terms (VIM)**. https://www.bipm.org/utis/common/documents/jcgm/JCGM_200_2012.pdf
- Kurnia, R., Ferdiana, R. y Wibirama, S. (2018). Software metrics classification for agile scrum process: A literature review. En *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)* (pp. 174-179). **IEEE**. <https://doi.org/10.1109/ISRITI.2018.8864244>
- Menéndez Domínguez, V. H. y Castellanos Bolaños, M. E. (2015). **SPEM: Software process engineering metamodel**. *Archivo de la Revista Latinoamericana de Ingeniería de Software*, 3(2), 92-100. <https://doi.org/10.18294/relais.2015.92-100>

- Mills, E. E. y Shingler, K. H. (1988). *Software Metrics: SEI Curriculum Module SEI-CM-12-1.1*. Software Engineering Institute, Carnegie Mellon University. https://insights.sei.cmu.edu/documents/1537/1988_007_001_15608.pdf
- Noor, H., Hayat, D. B., Hamid, A., Wakeel, T. y Nasim, R. (2020). Software metrics: Investigating success factors, challenges, solutions and new research directions. *International Journal of Scientific & Technology Research*, 9(8), 38-44.
- Park, R. E., Goethert, W. B. y Florac, W. A. (1996). *Goal-driven software measurement: A guidebook*. No. **CMU/SEI-96-HB-002**. Software Engineering Institute, Carnegie Mellon University. https://insights.sei.cmu.edu/documents/1623/1996_002_001_16436.pdf
- Piattini Velthuis, M. G., García Rubio, F. O., García Rodríguez de Guzmán, I. y Pino, F. J. (2011). *Calidad de sistemas de información*. Ra-Ma.
- Pressman, R. S. (2005). *Software engineering: A practitioner's approach*. McGraw-Hill.
- Ruiz, F., Genero, M., García, F., Piattini, M. y Calero, C. (2003). A proposal of a software measurement ontology. En *Proceedings of the Conference on Computer Science and Operational Research*. Springer. https://www.academia.edu/download/68115968/A_proposal_of_a_Software_Measurement_Ont20210715-13490-bwjmn6.pdf
- Sommerville, I. (2015). *Software engineering: Always learning*. Pearson.
- Srinivasan, K. P. (2015). Unique fundamentals of software measurement and software metrics in software engineering. *International Journal of Computer Science & Information Technology (IJCSIT)*, 7(4), 29-43. <https://www.airccse.org/journal/jcsit/7415ijcsit03.pdf>
- Tautz, C. y Von Wangenheim, C. (1998). **REFSENO**: A representation formalism for software engineering ontologies. Technical report No. 015.98/E, version 1.1. Fraunhofer **IESE**. <https://publica-rest.fraunhofer.de/server/api/core/bitstreams/05029db1-0b3f-408e-b786-468127baee2d/content>
- Xu, R., Xue, Y., Nie, P., Zhang, Y. y Li, D. (2006). Research on CMMI-based software process metrics. En *First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)* (vol. 2, pp. 391-397). **IEEE**. <https://doi.org/10.1109/IMSCCS.2006.260>

Control de procesos de software

► Objetivos

Los objetivos de este capítulo están enfocados en conocer los fundamentos del control de procesos de software. Al finalizar, el lector estará en capacidad de:

- Entender los conceptos fundamentales del control de procesos de software.
- Aplicar diferentes gráficos de control para verificar la estabilidad y el rendimiento de un proceso de software.
- Realizar seguimiento de proyectos de desarrollo de software que se ajustan a un determinado proceso de software.

Los conceptos y métodos asociados con la gestión de procesos y la mejora continua han ganado mayor importancia en la comunidad de software. Estas prácticas representan una manera de actuar y comprender los datos producidos por los procesos durante su ejecución que resultan en una mejor calidad, mayor productividad y productos competitivos (Florac *et al.*, 1997).

Más allá de la medición del proceso, se requiere la medición del comportamiento que se está ejecutando en un periodo razonable. El control estadístico del proceso puede contribuir a lograr este objetivo. Los principios del control estadístico del proceso sostienen que, al establecer y mantener niveles estables de variabilidad, los procesos producirán resultados predecibles; entonces, se puede decir que los procesos están bajo control estadístico.

El uso de este tipo de datos permite un estilo de gestión científica en el que las decisiones se toman “con base en hechos, en lugar de conjeturas, y se pueden producir mejores productos con menos desperdicio y retrabajo” (Fine, 1997).

► 7.1. Control estadístico del proceso

La base del control estadístico de procesos (**CEP**) se refiere a que, si un proceso se utiliza de manera consistente, mostrará a su vez resultados consistentes en atributos clave del proceso, como calidad, productividad, etc. (Card y Glass, 1990). La coherencia no significa que siempre se lograrán los mismos resultados, estos variarán, ya que hay algunas variaciones normales en el rendimiento que son inherentes a todos los procesos (Jalote y Saxena, 2002).

El control estadístico de procesos es una técnica de gestión cuantitativa esencial para medir, controlar y reducir la magnitud de las variaciones del proceso. El **CEP** es una técnica para controlar la calidad y el rendimiento de los procesos basada en el análisis estadístico de la variación de los resultados obtenidos al medir los atributos del proceso. El análisis estadístico de esta variación permite identificar problemas y proponer acciones para que los procesos se vuelvan estadísticamente estables (Florac y Carleton, 1999).

El **CEP** ha mostrado ser muy efectivo en contextos de manufactura y recientemente ha sido extendido al dominio de los procesos de software. El **CEP** es un enfoque basado estadísticamente que ayuda a determinar si un proceso es estable. Por medio del uso de pocos puntos de datos, se pueden determinar puntos de control superiores e inferiores para la variabilidad aceptable del desempeño del proceso (Caivano, 2005).

En la figura 7.1, se muestra un diagrama de máquinas de estado de un ciclo de control. Un ciclo de control está conformado por los siguientes estados: a partir del estado “Proceso bajo control”, un ciclo de control permanece en este estado durante un tiempo; luego, si en algún momento está fuera de control, pasa al estado “Fuera de control” y permanece en él hasta un punto que cae fuera de los límites de control; a partir de ahí pasa al estado de “Reparación del proceso y producto”; cuando esas reparaciones finalizan, vuelve al estado “Proceso bajo control”. Un ciclo de control es un intervalo entre dos reparaciones sucesivas.

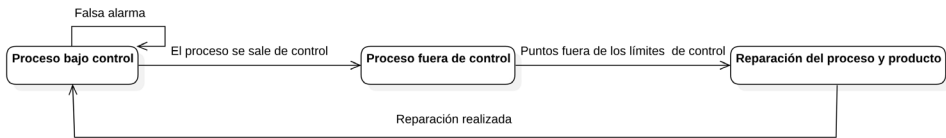


Figura 7.1. Diagrama de estados de un ciclo de control

Fuente: elaboración propia con base en Caivano (2005) desarrollado con la herramienta StarUML.

Por tanto, un ciclo en la vida del proceso consta de tres etapas. Estas son: proceso que opera en un estado bajo control (antes de la llegada de una causa asignable), proceso que opera en un estado fuera de control (entre la llegada y la detección de la causa asignable) y búsqueda y reparación de la causa asignable.

El **CEP** ha sido adoptado por organizaciones desarrolladoras de software para mejorar la gestión y evaluación de procesos. El **CEP** utiliza varios gráficos de control junto con indicadores para establecer límites operacionales de variación aceptable, y monitorear y evaluar la evolución del desempeño del proceso.

► 7.2. Gráficos de control

Los gráficos de control son herramientas ideales para analizar el comportamiento del proceso, ya que miden el rendimiento de este a lo largo del tiempo (Florac *et al.*, 2000). Las organizaciones de software están comenzando a apreciar el valor que los gráficos de control agregan cuando se utilizan para proporcionar información cuantitativa sobre el comportamiento de los procesos de software. Los gráficos de control se centran en estudiar el comportamiento del proceso desde el punto de vista de su variación y estabilidad. A continuación, se describirán dos aspectos fundamentales de los gráficos de control: la variación y la estabilidad.

7.2.1. Variación

Los resultados producidos por un proceso pueden caracterizarse por algunos atributos de calidad, cuyos valores generalmente muestran alguna variación. La base para los gráficos de control es el reconocimiento de dos tipos de variación: variación de causa común y variación de causa asignable (Florac *et al.*, 1997):

- *La variación de causa común:* Es la variación en el rendimiento del proceso debido a la interacción normal entre los componentes del proceso (personas, máquinas, material, entorno y métodos utilizados). A veces, se las denomina “causas naturales”. Esta variación se debe a una causa común aleatoria, pero variará dentro de límites predecibles. Estas causas son inherentes al proceso y están presentes todo el tiempo. La variación en el rendimiento del proceso es predecible y los resultados inesperados son raros. Si la variación en el rendimiento de un proceso solo se debe a causas comunes, se dice que el proceso está bajo control estadístico o que es un proceso estable.
- *La variación de causas asignables:* Surge de eventos que no son parte del proceso normal y lo pueden hacer inestable marcando impactos en las características del producto. Estos impactos cambian significativamente los patrones de variación. A veces, se las denomina “causas especiales”. Representan cambios anormales repentinos o persistentes en uno o más de los componentes del proceso, ocurren a veces y pueden prevenirse. Si algunos de los valores caen por fuera de los límites o muestran un patrón no aleatorio, una causa asignable se asume que se presenta. Estos cambios pueden ser aspectos como entradas al proceso, el entorno, los pasos del proceso en sí mismos o la forma en que se ejecutan los pasos del proceso. Se dice que un proceso que está operando en presencia de causas asignables es un proceso fuera de control.

Se dice que un proceso está bajo control estadístico si toda la variación en los atributos es causada por causas naturales. Al analizar la variabilidad del proceso a lo largo del tiempo, es posible identificar la aparición de causas especiales de variación, que son variaciones en el rendimiento del proceso debido a eventos que no son normales al proceso (Shewhart, 1926). Estas causas deben investigarse y eliminarse del proceso para restaurar su estabilidad.

Un ejemplo de presencia de causas asignables de variación se ilustra en la figura 7.2. Antes del instante t_1 , el proceso está bajo control, es decir, solo variaciones de causa común están presentes y tanto la media (μ) como la desviación estándar del proceso (σ) están con sus valores bajo control. En el instante t_1 , se produce una causa asignable que cambia la media del proceso a un nuevo valor $\mu_1 > \mu_0$. En el instante t_2 , ocurre otra causa asignable que resulta en $\mu = \mu_0$, pero ahora la desviación estándar del proceso

se ha desplazado a un valor mayor $\sigma_1 > \sigma_0$. En el instante t_3 , se presenta otra causa assignable que produce que tanto la media del proceso como la desviación estándar tomen valores fuera de control (Montgomery, 2012).

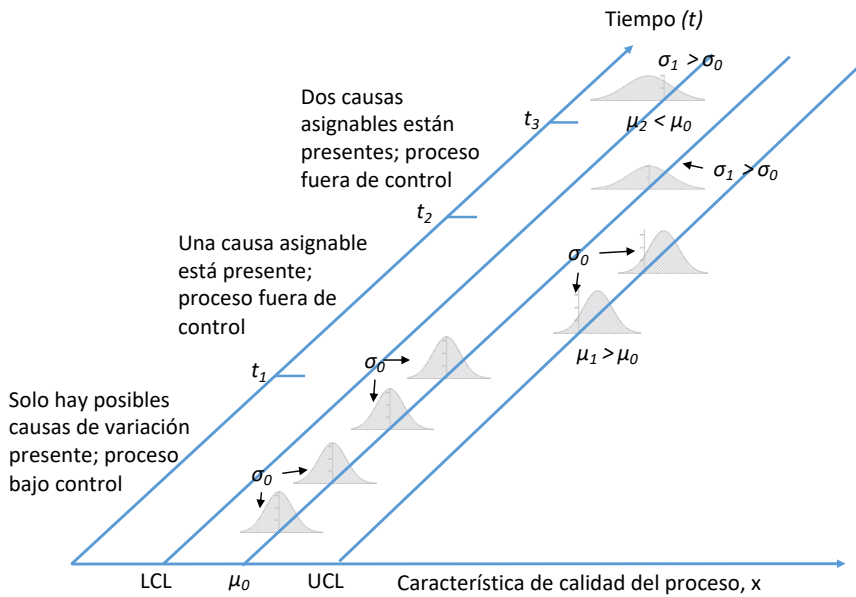


Figura 7.2. Variación de causas assignables

Fuente: elaboración propia con base en Montgomery (2012).

Un objetivo principal del control estadístico del proceso es detectar rápidamente la aparición de variaciones de causas assignables en el proceso (Jalote y Saxena, 2002).

7.2.2. Estabilidad

La medición de la variación de un atributo de proceso y el seguimiento de los resultados a lo largo del tiempo determinan la estabilidad de un proceso con respecto al atributo. Si una o más mediciones están fuera del rango de variación aleatoria o existen patrones sistemáticos, el proceso puede que no sea estable. Por tanto, se deben buscar las causas assignables y eliminar las que se encuentren si se quiere lograr un estado de operación estable y predecible (Florac *et al.*, 1997).

Cuando un proceso es estable, más del 99% de la variación del rendimiento de este caerá dentro de 3 sigmas (desviación estándar) de la media. Cuando la variación del proceso cae fuera de los límites de 3 sigmas, es muy probable que la variación sea causada por una anomalía en este. Cuando un proceso es estable, o casi, los límites de 3 sigmas determinan la cantidad de variación que es normal o natural para este.

► 7.3. Tipos de gráficos de control

Los gráficos de control más comunes fueron propuestos por Walter A. Shewhart. A continuación, se presentarán cada uno de los gráficos de control junto con ejemplos y usos en el control de procesos de software.

7.3.1. Gráficos de control de Shewhart

Para mantener un proceso operando bajo control estadístico, es esencial monitorear su desempeño e identificar cuándo se sale de control. Los gráficos de control de Shewhart (Shewhart, 1926) son las herramientas de **CEP** más utilizadas por la industria manufacturera para determinar si un proceso es estable.

Estos gráficos desarrollados por Shewhart en la década de 1920 se consideran herramientas importantes para el **CEP** y utilizan técnicas estadísticas para detectar fuentes inusuales de variación en un proceso de producción (Chang y Tong, 2013). Los gráficos de control son herramientas comunes que se han utilizado para monitorear los procesos de fabricación; son visualizaciones gráficas de una característica de calidad que se ha medido o calculado a partir de una muestra frente al número o el tiempo de esta. Representan una interesante herramienta para monitorear y evaluar si un proceso está bajo control estadístico. Se consideran el instrumento principal para analizar la variación del rendimiento del proceso. Los gráficos de control también se pueden usar para estimar los parámetros de un proceso de producción y determinar su capacidad. En un gráfico de control, se elige algún atributo de calidad y se trazan los valores del atributo para las muestras tomadas de la producción en algunos intervalos de tiempo.

Los gráficos de control tienen varios beneficios (Salazar, 2019):

- Ayudan a visualizar la variación.
- Apoyan la búsqueda y corrección de problemas cuando ocurren.

- Predicen los rangos esperados de resultados.
- Analizan patrones de variación del proceso por causas especiales o comunes.

En la figura 7.3, se presenta un ejemplo de gráfico de control típico. El eje y de un gráfico de control denota una característica de calidad o una característica particular del proceso que se controla y se marca en unidades en las que se expresa el valor de prueba. El eje x consiste en intervalos de tiempo o números de muestra. Existe una línea central que representa el promedio del valor de la métrica estudiada o también puede indicar un valor nominal (Şengöz, 2018).



Figura 7.3. Ejemplo de gráfico de control

Fuente: elaboración propia.

Los límites de control se eligen de modo que, si el proceso está bajo control, casi todos los puntos de muestra caerán entre ellos. El límite superior identifica el límite de control superior (Upper Control Limit [UCL]), mientras el inferior designa el límite de control inferior (Lower Control Limit [LCL]), respectivamente. Los datos recopilados se trazan en secuencia y, a continuación, se interpreta el patrón que aparece en el gráfico (Şengöz, 2018). Esos límites permiten filtrar la variación de causa asignable de la variación de causa común. Los límites se derivan empíricamente de las mediciones de la variación en el rendimiento del proceso a lo largo del tiempo (Florac *et al.*, 1997).

Todos los gráficos de control clásicos tienen una línea central y límites de control en ambos lados de la línea central. Tanto la línea central como los límites representan estimaciones que se calculan a partir de un conjunto de observaciones recopiladas mientras se ejecuta el proceso. La línea

central y los límites de control están destinados a revelar lo que el proceso realmente puede hacer (su nivel actual de rendimiento).

Mientras los puntos se encuentren dentro de los límites de control, se supone que el proceso está bajo control. Sin embargo, si un punto se encuentra fuera de los límites de control, se interpreta como evidencia de que el proceso está fuera de control y, por tanto, se requieren acciones para encontrar y eliminar las causas asignables responsables de este comportamiento.

Se recomienda conectar los puntos del gráfico de control con segmentos de línea recta, para que sea más fácil visualizar cómo la secuencia de puntos evoluciona con el tiempo.

Los límites de control tradicionales son ± 3 sigmas, donde sigma es la desviación estándar de la estadística trazada en el gráfico (figura 7.4). Cuando un límite de control se encuentra más allá del rango de resultados posibles, por ejemplo, un valor negativo para el tamaño del producto, el número de defectos o el rango de subgrupos, este límite de control generalmente se omite.

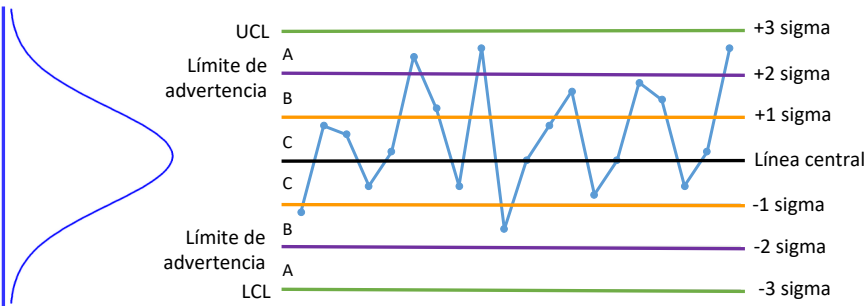


Figura 7.4. Límites 3 sigma de control del proceso

Fuente: elaboración propia.

Los límites de control generalmente se establecen en 3 sigma alrededor de la media. Es decir, el **UCL** es el valor medio de las muestras más 3 sigma, mientras el **LCL** es el valor medio menos 3 sigma. Con estos límites de control, las posibilidades de que un punto caiga fuera de los límites de control, incluso cuando no hay una causa especial, es de solo el 0,27 % (Montgomery, 2012). Por tanto, cuando un punto cae fuera de los límites de control, es muy probable que alguna causa asignable esté presente en el proceso.

Hay una estrecha relación entre la curva de distribución normal y un gráfico de control. Los gráficos de control se construyen sobre la base de expandir los límites sigma por encima y por debajo de la media. En la figura 7.4, se pueden observar los valores “límite de advertencia” que corresponden a 2 sigmas.

Hay varios cuadros de control que se pueden usar de acuerdo con los tipos de datos, el número de observaciones disponibles y la distribución del conjunto de datos. Hay gráficos basados en el índice de localización, es decir, media, mediana, etc., y otros basados en el índice de dispersión, es decir, desviación estándar, varianza, etc. (Caivano, 2005). También existen varios gráficos de control Shewhart, para datos variables y de atributos.

En los procesos de software, debido a la escasez de datos y dado que las mediciones a menudo ocurren solo como valores individuales, los gráficos de control más utilizados son los gráficos de control X y R (Caivano, 2005). En general, en los procesos de software, si un punto cae fuera de los límites de control, se pueden tomar medidas para mejorar el producto de trabajo o cambiar el proceso para eliminar la causa especial detectada.

7.3.2. Gráficos de control para variables

Los gráficos de control para variables permiten estudiar la calidad de las características numéricas. Se utilizan ampliamente porque permiten un control más efectivo y proporcionan más información sobre el rendimiento de los procesos. Proporcionan al usuario una estimación de la tendencia central y la distribución del caso estudiado. Los gráficos de control más utilizados, como se indicó, son el gráfico de control de mediciones individuales (X) y el gráfico de control de rangos (R). Además, también existen el gráfico de control de promedios ($\bar{X}$), y el gráfico de control de desviación estándar (s).

Gráfico de control de individuos (gráfico de control X)

Los gráficos de control de individuos se utilizan en casos en los que solo se debe definir un valor medido en una escala numérica, es decir, contar, calcular o utilizar un instrumento de prueba. Cada punto de datos en el gráfico de control X representa un valor individual de una característica medible.

A continuación, se presenta un procedimiento para la generación de este tipo de gráfico de control:

1. Se deben tener al menos 10 valores, aunque 20 es mejor; pero, si hay un gran intervalo de tiempo entre la obtención de los valores, también pueden servir 8 valores.
2. Se calcula el valor promedio de X ($\bar{X}$).
3. El valor absoluto de las diferencias entre dos valores consecutivos de X se llama rango móvil. El rango móvil (**RM**) y el promedio de **RM** ($\overline{RM}$) son calculados.
4. Los límites **UCL** y **LCL** son calculados utilizando la fórmula $\bar{x} \pm 2,66 \overline{RM}$.
5. Las líneas del gráfico de control se dibujan con la línea central ($\bar{x}$), **UCL** y **LCL**.
6. Los valores utilizados en el cálculo se trazan en el gráfico.
7. Los valores que aparecen a su debido tiempo son dibujados en el mismo cuadro e interpretados.

El siguiente ejemplo presenta el uso del gráfico de control X para un proceso de inspección de código. El atributo definido para las inspecciones es la densidad de defectos detectados durante la inspección (defectos/**LOC**). El gráfico de control se presenta en la figura 7.5.



Figura 7.5. Gráfico de control de inspecciones del proceso

Fuente: elaboración propia con base en Jalote y Saxena (2002).

La densidad de defectos para la inspección 18 es mayor que el límite de control superior. Por ejemplo, se puede encontrar que la causa asignable es que se introdujeron nuevos estándares de codificación y, por ello, el

número de defectos es demasiado alto. La acción u oportunidad de mejora que sigue a esta identificación podría ser capacitar a los programadores en los nuevos estándares de codificación (Jalote y Saxena, 2002).

Gráfico de control de promedios ($\bar{x}$) (gráfico de control $\bar{x}$)

Los gráficos de control de promedios son uno de los gráficos de control más utilizados. Los cambios en los promedios muestran la variación observada en la tendencia central de un subgrupo al siguiente.

Durante el muestreo, se toman n datos para analizar el caso en estudio en intervalos específicos. El gráfico de control de promedios utiliza los promedios de las muestras tomadas. Los promedios de las muestras poseen una distribución normal. La interpretación del gráfico depende de encontrar cuán cerca están los promedios de las muestras medidas al valor nominal o promedio.

Los pasos de preparación para un gráfico de control de promedios son:

1. Debe existir al menos 10 grupos de medición diferentes de un tamaño de muestra de 5 valores, aunque 12 valores es mejor.
2. La media del tamaño de la muestra se calcula para cada grupo de medición diferente, en que cada media del tamaño de la muestra se indica como $\bar{x}$.
3. El rango es la diferencia entre el valor máximo y mínimo en una muestra. Se calcula el rango (R) para cada tamaño de muestra.
4. Los promedios de los valores de $\bar{x}$ ($\bar{\bar{x}}$) y R ($\bar{R}$) son calculados.
5. Los límites **UCL** y **LCL** son calculados mediante la fórmula $\bar{\bar{x}} \pm \bar{R} A_2$.
6. Las líneas del gráfico de control se dibujan con la línea central ($\bar{\bar{x}}$), **UCL** y **LCL**.
7. Los valores calculados en el paso 2 ($\bar{x}$) se colocan en el gráfico.
8. Los valores que aparecen a su debido tiempo son dibujados en el mismo cuadro e interpretados.

En la figura 7.6, se muestra un ejemplo de gráfica de control de promedios en que se especifica para una colección de datos sobre la cantidad de horas diarias promedio gastadas para un determinado servicio. Los datos se agrupan por semana para generar los datos promedio. Según la figura 7.6, los datos representan un proceso que no está fuera de control, ya que está dentro de los límites de control establecidos. Cada número de semana representa un grupo de datos con 5 valores (de lunes a viernes).

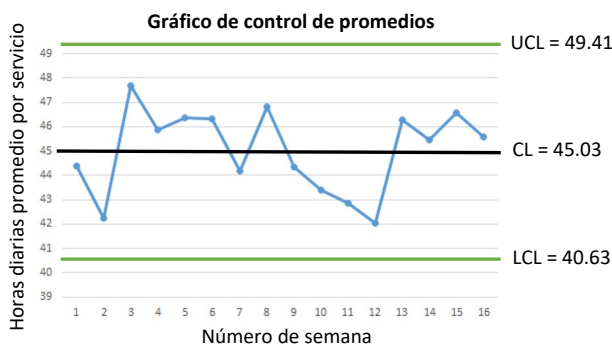


Figura 7.6. Gráfico de control de promedios

Fuente: elaboración propia con base en Florac *et al.* (1997).

Gráfico de control de rangos (R) (gráfico de control R)

El gráfico de control de rangos expresa que el rango de cada subgrupo (la diferencia entre el valor más pequeño y el más grande en este subgrupo) no excedió los límites establecidos. El rango de los valores dentro de cada subgrupo se puede calcular y trazar.

Estos gráficos se utilizan junto con los gráficos de control de promedios. Parte de la información estadística se pierde cuando se usan tamaños de muestra pequeños y se utiliza la media de los valores. La preparación del gráfico de control R se vuelve tan importante para que la información perdida pueda aparecer. Los rangos no poseen una distribución normal como las medias de las muestras. Cada punto en el gráfico de control R representa el rango móvil de dos puntos de datos sucesivos.

Mientras un gráfico de control $\bar{x}$ brinda información sobre el comportamiento de los valores medios de una muestra, un gráfico de control R brinda información sobre las diferencias en las muestras.

En la figura 7.7, se muestra la gráfica de control R que complementa la gráfica de control de promedios de la figura 7.6. En este caso, los rangos de las muestras se mantienen dentro de los límites aceptables y las horas diarias promedio de servicio son estables.

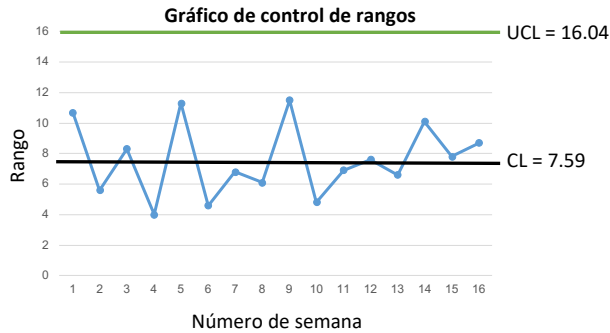


Figura 7.7. Gráfico de control de rangos

Fuente: elaboración propia con base en Florac *et al.* (1997).

Gráfico de control de desviación estándar (s) (gráfico de control s)

Los cuadros de control preparados con los valores de desviación estándar se denominan cuadros de control de desviación estándar. Estos gráficos se utilizan junto con los gráficos de control de promedios. Cuando se usa la media de los valores, parte de la información estadística se pierde. Se prefiere usar los gráficos de control (s) en los casos en los que faltan datos en las muestras, generalmente cuando el tamaño de la muestra es grande. El gráfico de control (s) proporciona información sobre la variación general de los valores con respecto al valor promedio de las muestras.

Los gráficos de control (s) representan el comportamiento general de la distribución en la población. En una muestra de valores, con el mismo R, (s) puede ser alto o bajo. Esto significa que R da una interpretación específica de un caso, pero (s) da una interpretación general al respecto.

En la figura 7.8, se presenta la gráfica de control de promedios como la gráfica de desviación estándar para cuatro grupos de módulos para los cuales se determinó la métrica *fan out*. Se puede observar que en la gráfica de control de promedios el equipo 1 obtuvo un promedio por fuera del límite de control inferior. Por tanto, se podría concluir que existen diferencias reales entre los equipos. En la gráfica de control de desviación estándar (s), la dispersión de los datos es muy baja y se puede concluir que son esencialmente los mismos.

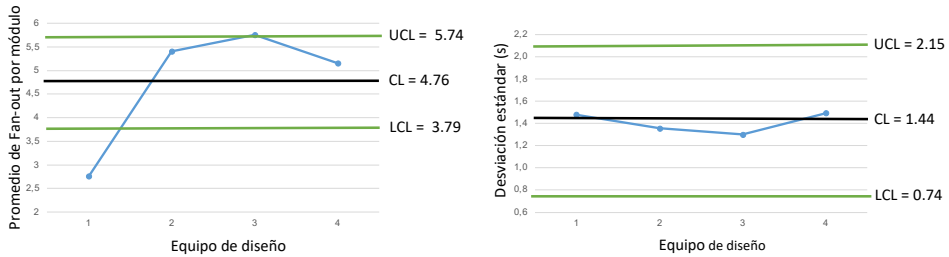


Figura 7.8. Gráfico de control de desviación estándar

Fuente: elaboración propia con base en Florac *et al.* (1997).

7.3.3. Gráficos de control para atributos

Los gráficos de control para atributos se utilizan cuando los datos no se presentan midiendo en una escala numérica, sino que se definen como conforme o no conforme con las especificaciones. Luego, se obtienen diferentes proporciones adecuadas para cada caso, y se dibujan los respectivos gráficos de control. Los gráficos más utilizados son el gráfico de control para la fracción no conforme (p), el gráfico de control para el número de elementos no conformes (np), el gráfico de control para las conformidades por unidad (u) y el gráfico de control para las no conformidades (c).

Gráfico de control para fracciones no conformes (p) (gráfico de control p)

Los gráficos de control p se obtienen dividiendo la cantidad de elementos conformes entre la cantidad total de producción. Los gráficos de control p siguen una distribución binomial. El gráfico p es el gráfico de atributos más utilizado. La p representa la fracción o el porcentaje del número de elementos que son inacceptables (o defectuosos). La decisión de aceptar o no el proceso puede basarse en una característica o varias, pero un elemento defectuoso se cuenta como defectuoso solo una vez, aunque pueda tener varios defectos.

En la figura 7.9, se presenta un ejemplo de gráfico de control (p) con diferentes fracciones de producto de software no conforme de acuerdo con la especificación de requisitos de software. En este caso, varios valores p están fuera de los límites de control. Esta configuración de valores p muestra que el proceso no está bajo control estadístico. Como hay tres o más valores fuera de los límites de control, el proceso está fuera de control. Las causas de este comportamiento deben determinarse para resolver tales problemas.

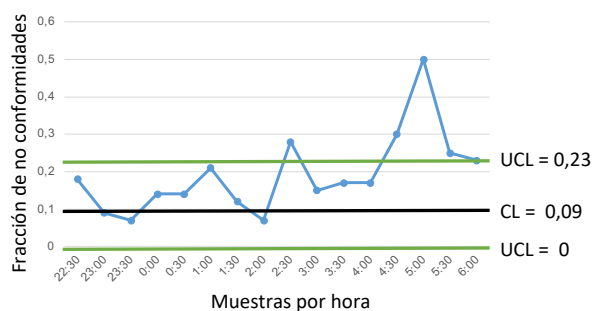


Figura 7.9. Gráfico de control para fracciones no conformes

Fuente: elaboración propia con base en Florac y Carleton (1999).

Gráfico de control para elementos no conformes (np) (gráfico de control np)

Los gráficos de control preparados con el número de elementos no conformes se denominan gráficos de control np . Este tipo de gráfico es sensible a los cambios en la cantidad de elementos defectuosos en el proceso de medición. El np en los gráficos np representa el número medio de éxito de una distribución binomial. El eje vertical representa el número de defectos para cada subgrupo y el eje horizontal la designación de subgrupo. Los límites de control se calculan sobre la base de la distribución binomial y una aproximación basada en el teorema del límite central. Un ejemplo se presenta en la figura 7.10 donde el número de no conformidades presentada en una colección de 10 muestras está dentro de los límites aceptables, excepto la muestra número 5.

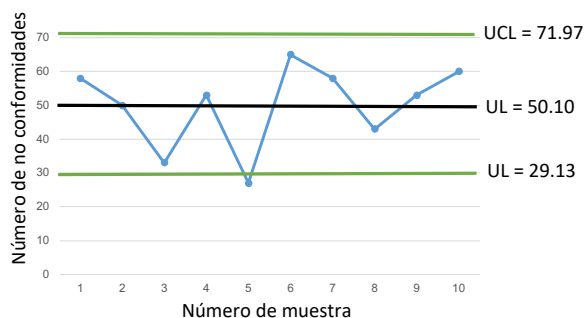


Figura 7.10. Gráfico de control para elementos no conformes np

Fuente: elaboración propia con base en Florac y Carleton (1999).

Gráfico de control de conformidad por unidad (u) (gráfico de control u)

El gráfico de control (u) se utiliza generalmente para trazar tasas individuales de no conformidades por unidad. Es utilizado con datos recopilados en subgrupos de diferentes tamaños y muestra cómo se mide el proceso por el número de no conformidades por elemento o grupo de elementos.

El número de no conformidades se divide por unidad para encontrar el valor “por unidad” de los defectos y llevarlos todos a la misma base de comparación. El gráfico (u) se utiliza para determinar si el proceso es estable y predecible.

Este gráfico es útil cuando el proceso es demasiado complejo para ser considerado simplemente conforme o no conforme. Es apropiado cuando el área de oportunidad para un defecto varía de un subgrupo a otro. Por tanto, el desplazamiento de las líneas **UCL** y **LCL** depende del tamaño del subgrupo. Los defectos por mil líneas de código fuente, los defectos por punto de función y las fallas del sistema por día en operación son ejemplos de datos de atributos que son candidatos para los gráficos de control (u).

Los datos en un gráfico de control (u) al seguir la distribución de Poisson deben cumplir las siguientes condiciones (Florac y Carleton, 1999):

- Deben basarse en recuentos de eventos discretos.
- Los eventos discretos deben ocurrir dentro de una región finita bien definida de espacio, tiempo o producto.
- Los eventos deben ocurrir independientemente uno del otro.
- Los eventos deben ser raros en relación con la oportunidad de que ocurran.

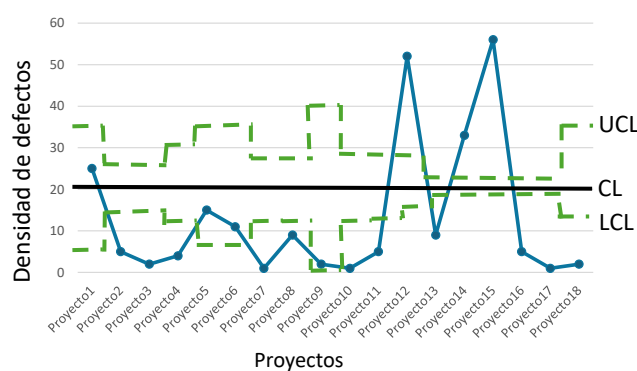


Figura 7.11. Gráfico de control de conformidad por unidad u

Fuente: elaboración propia con base en Florac y Carleton (1999).

La figura 7.11 muestra un gráfico (u) con los valores de densidad de defectos de un cierto grupo de documentos de diseño al final de la fase de implementación. Los límites de control superior e inferior varían para cada proyecto y la mayoría de ellos están fuera de esos límites. Por tanto, el proceso realizado en los proyectos no es estable y está fuera de control.

Gráfico de control de no conformidades (c) (gráfico de control c)

Los gráficos de control preparados con el número de no conformidades por unidad constante se denominan gráficos de control (c). En estos casos, la unidad será constante para todos los datos recopilados.

El gráfico de control (c) es sensible a los cambios en la cantidad de elementos defectuosos en el proceso de medición y muestra cómo el número de defectos o no conformidades para un proceso está cambiando con el tiempo. Cuando cambia el número de elementos probados dentro del subgrupo, se debe usar un gráfico de control (u), ya que muestra el número de defectos por unidad en lugar de los defectos totales.

En la figura 7.12, se presenta un ejemplo de gráfico de control de no conformidades para páginas web vistas por usuarios por mes. Solo se presentó un único proyecto cuya cantidad de páginas vistas superó el umbral superior establecido.

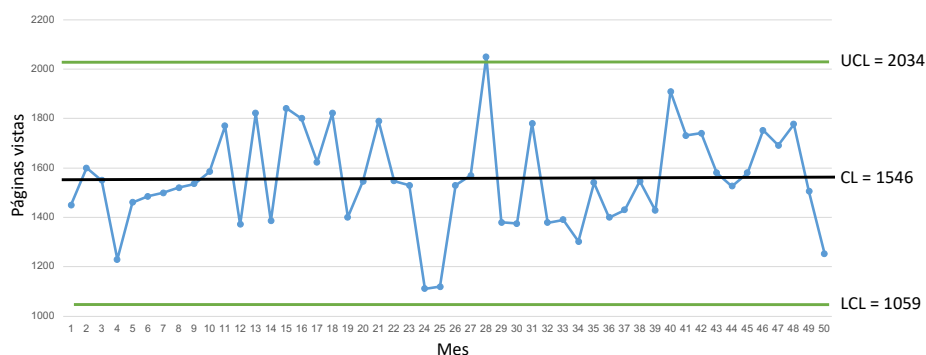


Figura 7.12. Gráfico de control para no conformidades

Fuente: elaboración propia con base en Florac y Carleton (1999).

7.3.4. Detección de inestabilidades y procesos fuera de control

Para verificar si el proceso tiene causas especiales de variación, existen varias pruebas de estabilidad para detectar patrones inusuales y comportamientos no aleatorios. El manual de Western Electric (WE) cita cuatro pruebas (Wheeler, 1993, 1995):

- *Prueba 1:* Un solo punto cae fuera de los límites de control de 3 sigmas.
- *Prueba 2:* Al menos dos de tres valores sucesivos caen en el mismo lado y más de dos unidades sigma lejos de la línea central.
- *Prueba 3:* Al menos cuatro de cinco valores sucesivos caen en el mismo lado y más de una unidad sigma lejos de la línea central.
- *Prueba 4:* Al menos ocho valores sucesivos caen en el mismo lado de la línea central.

La figura 7.13 muestra un ejemplo de verificación de estas pruebas.

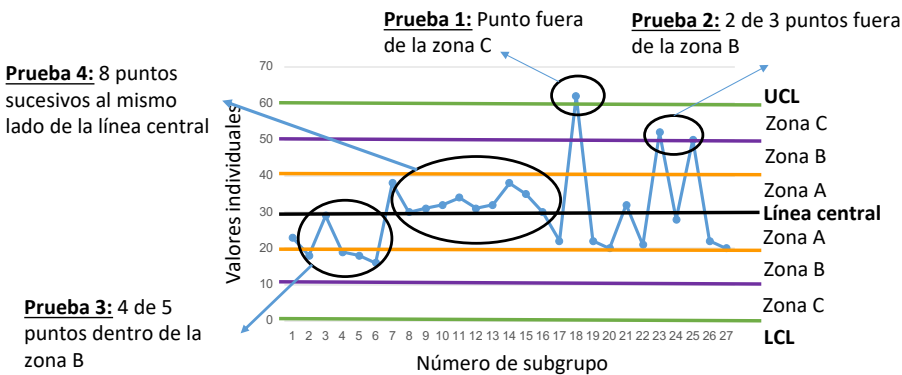


Figura 7.13. Pruebas para analizar la estabilidad de un proceso

Fuente: elaboración propia con base en Gonçalves *et al.* (2012).

La figura 7.14 muestra los pasos involucrados en la investigación de la estabilidad de un proceso proporcionando un contexto para discutir el uso de mediciones de productos o procesos para evaluar la estabilidad del proceso. Los pasos en la figura 7.14 son los siguientes:

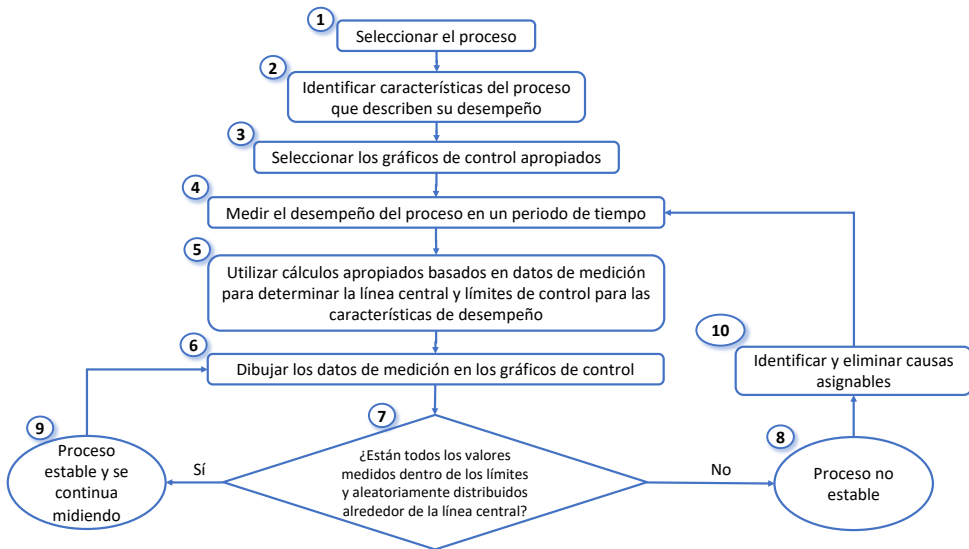


Figura 7.14. Pasos para usar gráficos de control en evaluación de la estabilidad del proceso

Fuente: elaboración propia con base en Florac *et al.* (1997).

1. Seleccionar el proceso para evaluar su estabilidad.
2. Identificar las características del producto o proceso que describen el rendimiento del proceso.
3. Seleccionar los tipos apropiados de gráficos de control.
4. Medir las características del producto o proceso durante un periodo. El número de mediciones y el periodo dependerán del proceso y del tipo de gráfico de control utilizado.
5. Utilizar los cálculos apropiados, aplicados a los datos de medición, para establecer las líneas centrales y los límites de variación para el proceso normal de actuación. Los cálculos específicos variarán según el tipo de gráfico de control utilizado.
6. Trazar los datos de medición obtenidos del paso 4 en los gráficos de control.
7. Comparar los valores trazados en el paso 6 con los valores de la línea central y los límites establecidos en el paso 5.
8. Si todos los valores trazados se distribuyen aleatoriamente por encima y por debajo de las líneas centrales y dentro de los lími-

tes de control, concluir que el proceso ha sido estable al menos durante el periodo o los eventos cubiertos por las mediciones. Continuar midiendo y trazando los datos para asegurarse de que el proceso se mantiene estable.

9. Si algún valor trazado excede los límites, o si el patrón de valores exhibe otro comportamiento no aleatorio, concluir que el proceso no es estable. Las razones de tales observaciones deben ser investigadas. Si se encuentra una causa asignable y el resultado es perjudicial, arreglar el proceso para que no vuelva a suceder. Si el resultado es beneficioso, intentar hacer que la causa sea parte del proceso.
10. Una vez que se hayan eliminado todas las causas asignables, los límites de control deben recalcularse. Esto puede significar que se necesitarán observaciones adicionales para determinar límites confiables para el proceso corregido.

Existen dos tipos de errores posibles, provenientes de la estadística e independiente de los límites de control establecidos (Montgomery, 2012):

- *Error tipo I:* Ocurre cuando un punto cae fuera de los límites de control debido a la variación debida a causas naturales. Esto obliga a buscar una causa asignable (cuando no existe) y agrega un costo adicional al proceso.
- *Error tipo II:* Se produce cuando un punto cae dentro de los límites de control, aunque hay alguna causa asignable en el proceso. En este caso, se ignora este punto en lugar de tomar medidas correctivas y esto conlleva un costo adicional en los procesos posteriores.

El costo en la etapa de bajo control (figura 7.1) se debe a un error de tipo I que puede ocurrir y el costo en una etapa fuera de control se debe a errores de tipo II. En la tercera etapa, el costo se debe a la reparación del proceso.

► 7.4. Seguimiento del proceso

Para garantizar una correcta y eficaz ejecución del proceso de software, se requiere algún nivel de medición y control sobre cada uno de los siguientes atributos (Fairley, 2011):

- *Esfuerzo:* Cantidad de trabajo invertido en diversas actividades laborales.

- ▶ *Calendario*: Logro de hitos medidos objetivamente.
- ▶ *Costo*: Gastos incurridos para varios tipos de recursos.
- ▶ *Progreso*: Productos de trabajo definidos, completados y aceptados.
- ▶ *Características del producto*: Requisitos funcionales implementados.
- ▶ *Atributos de calidad del producto*: Defectos, confiabilidad, disponibilidad, tiempo de respuesta, rendimiento, etc.
- ▶ *Riesgo*: Estado de los factores de riesgo junto con sus actividades de mitigación.

Generalmente, los valores medidos de estos atributos se comparan con los valores planificados para cada intervalo de medición especificado en el plan de proyecto. El control se ejerce cuando los valores reales se desvían de los valores planificados o especificados en más de una cantidad aceptable.

El esfuerzo, costo y calendario tienen atributos comunes y cada uno tiene atributos únicos. Los atributos comunes de esfuerzo, calendario y costo a medir y controlar incluyen la cantidad y el porcentaje de cada uno que se está gastando en diversas actividades del proyecto, incluida la gestión, el análisis, el diseño y la implementación del proyecto, así como la verificación, validación, y otras actividades de apoyo.

Las posibilidades de acción correctiva, cuando los valores reales de los atributos del proyecto no son los planificados o esperados, incluyen (Fairley, 2011):

- ▶ Extender los plazos de entrega.
- ▶ Agregar más recursos o utilizar mejores recursos.
- ▶ Mejorar varios elementos del proceso de desarrollo.
- ▶ Limitar los requisitos del producto.

Los recursos que podrían mejorarse, agregarse o reemplazarse incluyen (Fairley, 2011):

- ▶ Personas (se debe considerar la ley de Brooks al agregar más personal).
- ▶ Componentes software (por ejemplo, reingeniería de componentes de software para mejorar el rendimiento).
- ▶ Componentes de *hardware* (por ejemplo, más memoria o un procesador más rápido).
- ▶ Herramientas de software (por ejemplo, una herramienta de pruebas).

A continuación, se presentarán dos técnicas muy utilizadas en la industria para realizar seguimiento del proceso: el seguimiento binario y la técnica del valor ganado.

7.4.1. Seguimiento binario

El seguimiento binario es una técnica fundamental para seguir con precisión el estado de un proyecto. Los diferentes paquetes de trabajo se consideran “entregables binarios” (DeMarco, 1982). Cada paquete de trabajo, solicitud de cambio e informe de defectos produce uno o más productos de trabajo que deben satisfacer los criterios de aceptación antes que las nuevas versiones se conviertan en líneas base. Cada paquete de trabajo se considera con un 0 % completado hasta que su producto de trabajo asociado cumpla con sus criterios de aceptación; cuando los cumpla, el producto de trabajo se cuenta como 100 % completo. Por tanto, el seguimiento es binario.

Los paquetes de niveles superiores son promedios de los niveles inmediatamente inferiores hasta llegar al paquete de trabajo principal del proyecto. El seguimiento binario, como se muestra en las figuras 7.15 y 7.16, proporciona un seguimiento preciso del progreso real. Las X de los paquetes de trabajo indican que el trabajo se ha completado en un 100 %. Los ejemplos de seguimiento informan el progreso como el 41,5 % completado (figura 7.15) y el 71 % completado si es más detallado (figura 7.16). Los paquetes de trabajo con niveles más detallados proporcionan una mayor precisión de medida o disminuyen el riesgo de microgestión.

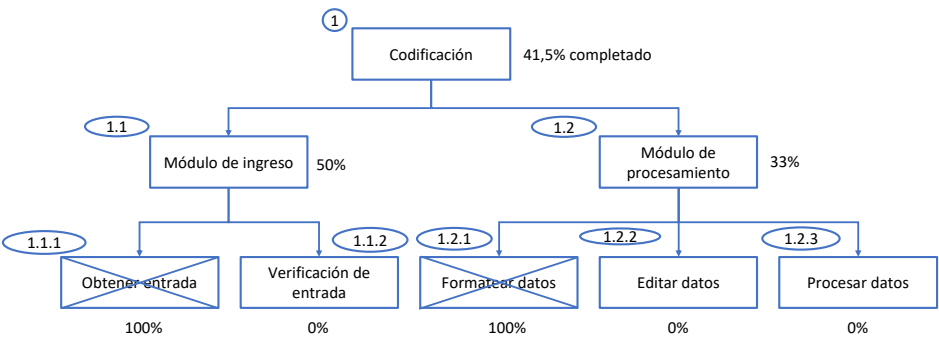


Figura 7.15. Seguimiento binario de paquetes de trabajo

Fuente: elaboración propia con base en Fairley (2011).

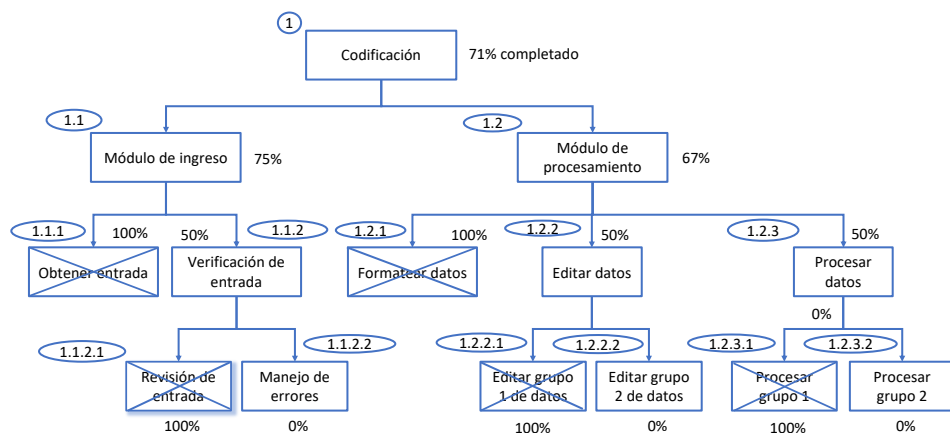


Figura 7.16. Seguimiento binario detallado de paquetes de trabajo

Fuente: elaboración propia con base en Fairley (2011).

La técnica del seguimiento binario puede proporcionar información del estado exacto para proyectos de software, y así prevenir el síndrome del 95% completado. Este síndrome indica que “el producto está completo en un 95% cuando se acerca la fecha de finalización programada, y así permanece por mucho tiempo”, como se observa en la figura 7.17.

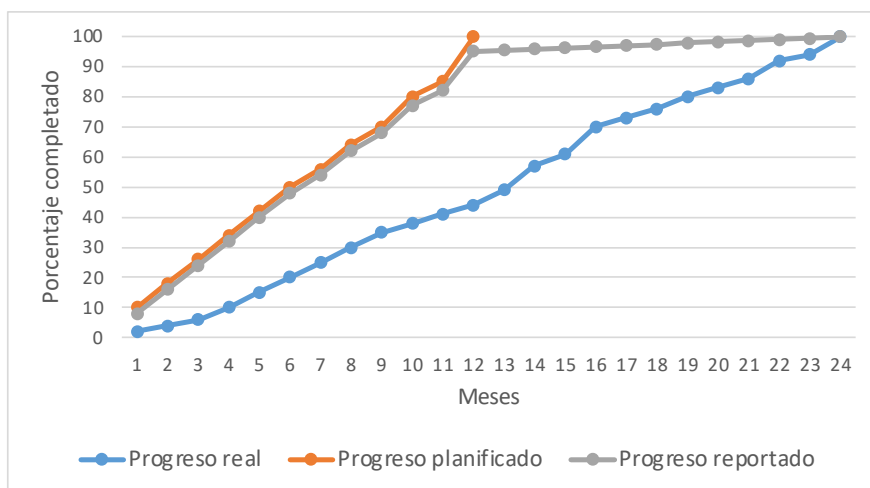


Figura 7.17. Síndrome del 95% completado

Fuente: elaboración propia con base en Fairley (2011).

7.4.2. Valor ganado

El método de valor ganado establece un valor para cada tarea sin considerar su tipo o duración. Siempre que se termina una tarea se obtiene un determinado valor ganado; si la tarea no se ha terminado, su valor ganado será cero (Humphrey, 2005). Para evaluar el progreso con respecto al plan, se compara el valor ganado real acumulado a la fecha con el valor ganado planificado acumulado a la fecha. Si el primero es igual o supera al segundo, el proyecto va bien o está adelantado; si no, el proyecto se encuentra atrasado.

El enfoque basado en el valor ganado para informar sobre la finalización de paquetes de trabajo es el siguiente (Fairley, 2011):

1. Se especifica el costo planificado y la duración de las tareas que se completarán en el proyecto.
2. Cuando se completa una tarea, acreditar el costo planificado para la tarea como su “valor ganado”.
3. Para todas las tareas completadas hasta la fecha, se debe comparar la suma de los valores ganados (es decir, el costo planificado para todas las tareas) con la suma de los costos reales de las tareas.
4. Para todas las tareas completadas hasta la fecha, comparar el número de tareas que deberían haberse completado con las que se han completado.

Si el costo real acumulado para completar todas las tareas hasta la fecha es mayor que el costo planificado para estas (paso 4), el proyecto supera el presupuesto. Por el contrario, si el costo real es menor al costo planificado, el proyecto está por debajo del presupuesto. De manera similar, si se han completado menos tareas hasta la fecha planificada, el proyecto está retrasado, o si se han completado más tareas de las planificadas, el proyecto está adelantado con respecto al calendario.

A continuación, se presenta un ejemplo de seguimiento de un proyecto basado en el valor ganado. Se presentan dos plantillas: la plantilla de planificación de tareas y la plantilla de planificación de calendario. El propósito de la plantilla de planificación de tareas es estimar el tiempo de desarrollo para cada tarea del proyecto, calcular el valor planificado para cada tarea del proyecto y estimar la fecha de finalización planificada para cada tarea. El propósito de la plantilla de planificación de calendario es registrar las horas estimadas y reales gastadas por periodo calendario y relacionar el valor planificado de las tareas con el calendario.

Tabla 7.1. Plantilla de planificación de tareas

ID tarea	Tarea	Datos planificados				Datos reales			
		Tiempo estimado (horas)	Tiempo estimado acumulado (horas)	Valor planificado	Valor planificado acumulado	Tiempo real (horas)	Tiempo real acumulado (horas)	Valor ganado	Valor ganado acumulado
1	Planificación	16	16	3.4	3.4	20	20	3.4	3.4
2	Análisis	52	68	7.2	10.6	45	65	7.2	10.6
3	Diseño arquitectónico	21	89	3.8	14.4	25	90	3.8	14.4
4	Diseño detallado	80	169	12.7	27.1	71	161	12.7	27.1
5	Codificación paquete A	69	238	10.1	37.2	60	221	10.1	37.2
6	Pruebas paquete A	25	263	5.3	42.5	55	276	5.3	42.5
7	Codificación paquete B	85	348	18.0	60.5				
8	Pruebas paquete B	38	386	6.1	66.6				
9	Codificación paquete C	94	480	10.1	76.7				
10	Pruebas paquete C	40	520	4.4	81.2				
11	Pruebas de integración	96	616	6.3	87.5				
12	Pruebas de aceptación	72	688	7.4	94.9				
13	Implantación	58	746	5.1	100				
Total		746		100					

Fuente: elaboración propia.

Tabla 7.2. Plantilla de planificación de calendario

Semana	Datos planificados				Datos reales			
	Tiempo estimado (horas)	Tiempo estimado acumulado (horas)	Valor planificado	Valor planificado acumulado	Tiempo real (horas)	Tiempo real acumulado (horas)	Valor ganado	Valor ganado acumulado
1	25	25	3.4	3.4	20	20	3.4	3.4
2	30	55	7.2	10.6	30	50	0.0	3.4
3	32	87	3.8	14.4	30	80	7.2	10.6
4	34	121	0.0	14.4	20	100	3.8	14.4
5	25	146	12.7	27.1	28	128	0.0	14.4
6	32	178	0.0	27.1	38	166	12.7	27.1
7	35	213	15.4	42.5	40	206	0.0	27.1
8	30	243	0.0	42.5	20	226	10.1	37.2
9	35	278	0.0	42.5	30	256	0.0	37.2
10	30	308	18.0	60.5	20	276	5.3	42.5
11	35	343	6.1	66.6				
12	40	383	10.1	76.7				
13	38	421	10.8	87.5				
14	30	451	7.4	94.9				
15	32	483	5.1	100.0				

Fuente: elaboración propia.

En la plantilla de planificación de tareas (tabla 7.1), se estiman las horas de cada tarea y se calcula el valor planificado de cada tarea dividiendo el tiempo de cada tarea entre el total del proyecto (746 horas) y multiplicando por 100. En las columnas de la derecha (datos reales), se registra el tiempo real que se tardó realizando cada tarea. Si una tarea se finalizó con éxito, su valor ganado asume el valor planificado asignado. Se puede observar que, aunque puede haber tareas con mayor o menor tiempo estimado, se les asigna el mismo valor planificado. Es decir, el valor ganado es indiferente del tiempo real de la tarea. En el ejemplo, el proyecto aún no ha terminado y se han terminado las primeras seis tareas.

En la plantilla de planificación de calendario (tabla 7.2), se realiza una estimación del tiempo de trabajo por semanas. Si en una determinada semana se termina una o varias tareas, se le asigna el valor ganado de las actividades terminadas. De igual manera, en las columnas de datos reales se registra el tiempo real trabajado por semana y se registra el valor ganado de cada tarea. Si en una determinada semana no se terminan tareas, el valor ganado para esa semana es cero (semanas 2, 5 y 7). Se puede observar que al finalizar la décima semana se tiene un valor ganado de 42,5 y el valor planificado para dicha semana es de 60,5. Por tanto, el proyecto se encuentra atrasado.

En la figura 7.18, se puede observar cómo evolucionó el valor ganado real con respecto al planificado y su estado actual en la semana 10. El proyecto comenzó a atrasarse desde la segunda semana, luego en la cuarta semana se igualó al plan, pero luego volvió a atrasarse. El estado actual del proyecto tiene un atraso del $(60,5 - 42,5) / 42,5 * 100 = 42,35 \%$.

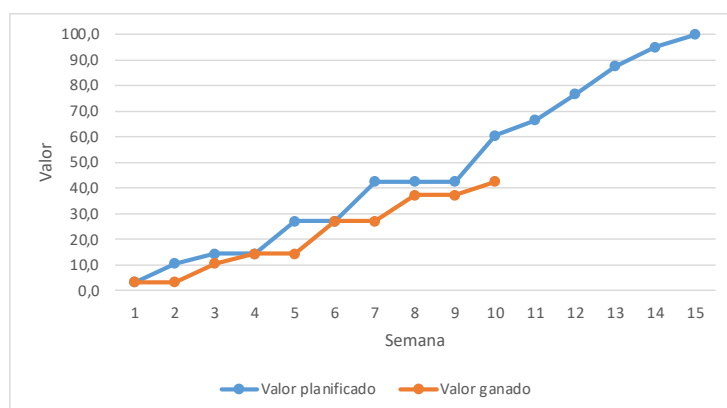


Figura 7.18. Ejemplo de seguimiento utilizando valor ganado

Fuente: elaboración propia.

► **Desafíos**

La tabla 7.3 resume los desafíos identificados por diferentes estudios al implementar el CEP en procesos de software. En consecuencia, las actividades humanas intensivas, las métricas de diversidad, las múltiples causas comunes y una pequeña cantidad de datos de proceso son características de la industria del software que no pueden modificarse fácilmente para técnicas cuantitativas específicas.

Tabla 7.3. Desafíos al implementar un CEP en procesos de software

Autores	Descripción
Baldassarre <i>et al.</i> (2004)	• Existen diferencias fundamentales entre el desarrollo de software y los procesos de manufactura. • Existen demasiados atributos y variables en el ciclo de vida del software.
Caivano (2005)	• Dificultad para elegir indicadores adecuados a fin de implementar CEP en procesos de software.
Komuro (2006)	• Características del proceso (por ejemplo, intensivo en recursos humanos y centrado en procesos). • Múltiples causas comunes de variación en los procesos de software. • Dificultad para obtener un gran conjunto de datos homogéneos.
Manlove y Kan (2007)	• El software es producido por personas, no por máquinas. • Demasiadas fuentes de variabilidad en el entorno de desarrollo de software. • Múltiples causas comunes de variación en el CEP. • El comportamiento del indicador de proceso varía con el ciclo de desarrollo.
Raczynski y Curtis (2008)	• Fuentes heterogéneas de variación. • Existen muy pocos puntos de datos al construir límites de control válidos. • Se deben mantener demasiados gráficos de control al desglosar datos por atributos.
Sargut y Demirörs (2006)	• Múltiples causas comunes de variación en procesos de software. • Selección de métricas para implementar procesos de software. • Compensación entre el número de puntos de datos y la profundidad del análisis.
Tarhan y Demirörs (2006)	• Muestreo racional de los datos del proceso. • Dificultad para elegir indicadores adecuados para implementar el CEP en procesos de software. • Insuficiencia de datos.
Weller y Card (2008)	• Pocos datos para construir gráficos de control. • Múltiples causas comunes de variación en procesos de software.

Fuente: elaboración propia con base en Montgomery (2012).

Los gráficos de control de Shewhart requieren una gran cantidad de observaciones para estimar la media del proceso y la variación del proceso, y así establecer los límites de control. Muchos estudios han indicado que el uso de 20 a 30 subgrupos con cuatro o cinco observaciones es adecuado (Montgomery, 2012).

► Ejemplos

A continuación, se presentarán varios ejemplos del uso de gráficas de control del proceso de software. Cada gráfica muestra valores de diferentes atributos de entidades usadas, consumidas y producidas del proceso de software.

Las entidades recibidas o usadas hacen referencia a eventos que puedan ocurrir en la fase de planificación o diseño de software y se pueden medir con atributos, como los cambios en los casos de uso y cambios en los requisitos funcionales. En la figura 7.19, se muestra un gráfico de control X sobre cómo evoluciona el porcentaje de cambios no resueltos en varios módulos de software que aún no se han realizado en la fase de diseño de software. La gráfica presenta algunos valores por fuera de los límites aceptables (prueba 1).

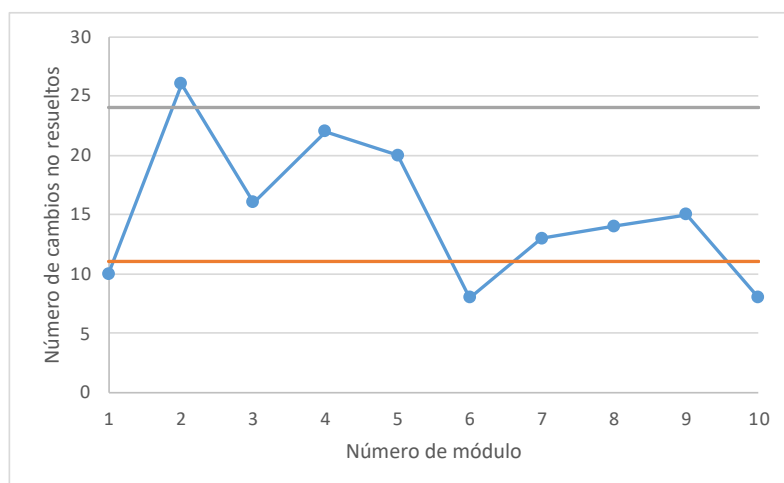


Figura 7.19. Gráfico de control X para cambios no resueltos

Fuente: elaboración propia.

Otro atributo para entidades recibidas es el grado de requerimientos de software desarrollados. En la figura 7.20, se presenta un gráfico de control X sobre el porcentaje de requerimientos no resueltos en módulos de software. El comportamiento de los valores recolectados está dentro de los límites permitidos; por tanto, el proceso desde el punto de vista de los requerimientos es estable.

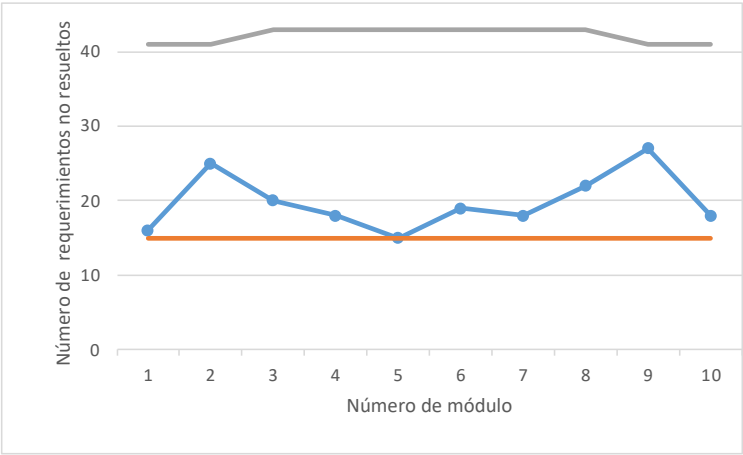


Figura 7.20. Gráfico de control X para requerimientos no resueltos
Fuente: elaboración propia.

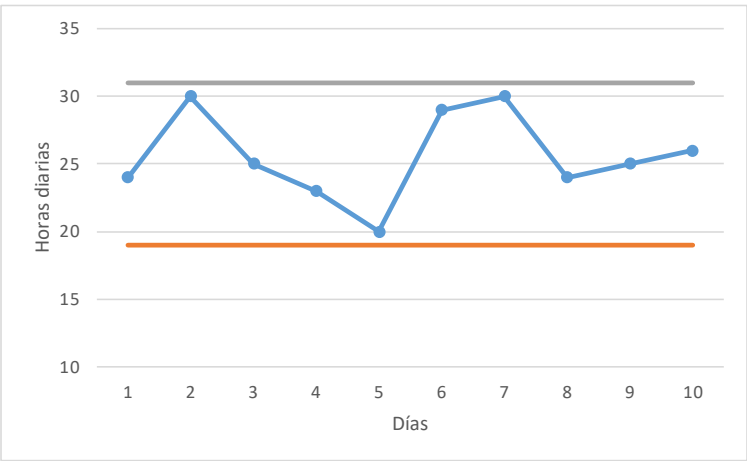


Figura 7.21. Gráfico de control X para esfuerzo en horas de trabajo por día
Fuente: elaboración propia.

Las entidades consumidas se refieren a actividades o recursos necesarios para el desarrollo de software, por ejemplo, el esfuerzo del equipo de desarrollo y el costo del desarrollo de software.

El esfuerzo se mide en horas que cada día pueden ser dedicadas al desarrollo, retrabajo, soporte, preparación o reunión. Un ejemplo de gráfico de control X generado para este atributo se muestra en la figura 7.21. Los valores de esfuerzo están dentro de los límites aceptables.

El atributo costo se aplica a un determinado proceso o actividad. En la figura 7.22, se presenta un gráfico de control X para el costo de diferentes procesos durante el desarrollo de un proyecto de software. El proceso es ligeramente inestable, ya que un dato se encuentra por debajo del límite de control inferior (prueba 1).

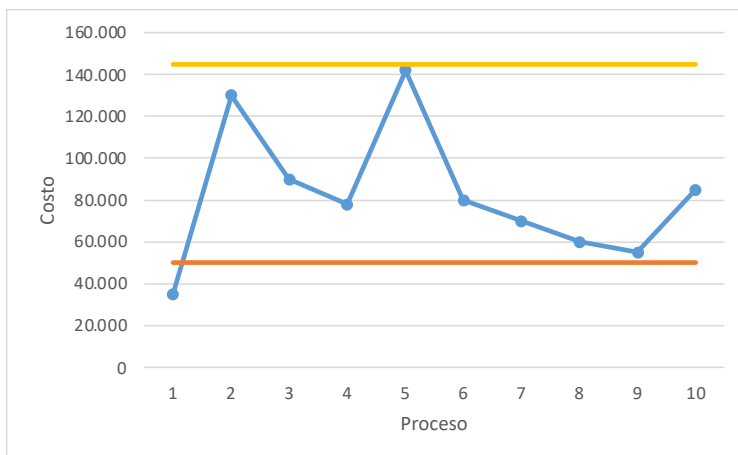


Figura 7.22. Gráfico de control X para el costo de procesos

Fuente: elaboración propia.

Con respecto a entidades producidas, se refieren a los objetos que puede producir el desarrollo de software. Algunos atributos medibles pueden ser el tamaño de los módulos o las unidades de trabajo producidas y los resultados de las pruebas del sistema.

La figura 7.23 presenta un ejemplo de gráfico de control X para los tamaños de los módulos dados en miles de líneas de código fuente. El proceso es ligeramente inestable debido a que dos puntos de datos están por debajo del límite de control inferior.

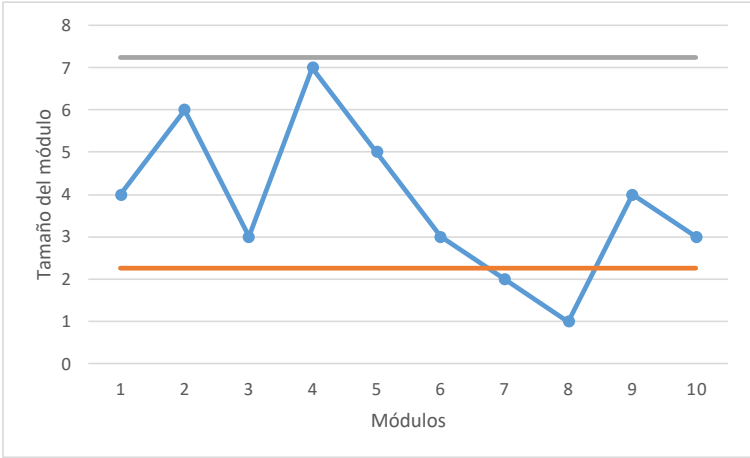


Figura 7.23. Gráfico de control X para el costo de procesos
Fuente: elaboración propia.

El atributo resultados de pruebas expresa un porcentaje de pruebas no aprobadas en cada módulo. En la figura 7.24, se presenta un ejemplo de gráfico de control X para el porcentaje de pruebas no aprobada para un conjunto de módulos. El proceso desde el punto de vista de las pruebas es estable.

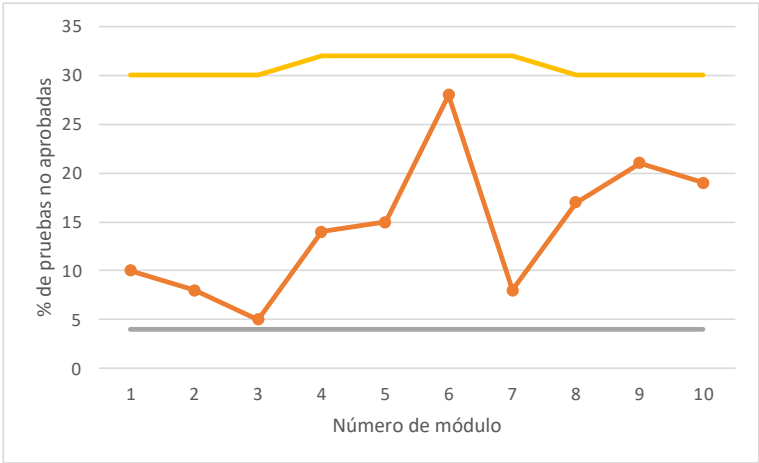


Figura 7.24. Gráfico de control X para pruebas no aprobadas
Fuente: elaboración propia.

► Resumen

El control estadístico del proceso se refiere a un conjunto de técnicas utilizadas para ayudar a entender, analizar e interpretar información numérica que permite identificar y rastrear la variación en los procesos. El control estadístico del proceso utiliza diferentes gráficos de control para evaluar esta variación.

Los gráficos de control tienen dos tipos principales según la forma en que se obtienen los valores utilizados. Si estos se obtienen midiendo valores numéricos, se denominan gráficos de control para las variables. Si se obtienen derivando proporciones, se denominan gráficos de control para los atributos. Estos cuadros se aplican a diferentes casos de procesos específicos, de modo que cada uno puede evaluarse por sí solo.

Cada tipo principal tiene diferentes gráficos de control, particularmente para el caso estudiado. Los tipos más importantes de gráficos de control para variables son principalmente.

- Gráfico de control de mediciones individuales.
- Gráfico de control de promedios.
- Gráfico de control de rangos.
- Gráfico de control de desviación estándar.

Los principales tipos de gráficos de control para los atributos son:

- Gráficos de control para fracciones no conformes.
- Gráfico de control para elementos no conformes.
- Gráfico de control de conformidad por unidad.
- Gráfico de control de no conformidades.

El seguimiento binario de los paquetes de trabajo es una técnica que evita el síndrome del 95% completado de los proyectos de software. A cada paquete de trabajo de bajo nivel se asigna un 0% o 100% de completitud. Los paquetes de niveles superiores son promedios de los niveles inmediatamente inferiores hasta llegar al paquete de trabajo principal del proyecto.

El valor ganado basado en el seguimiento binario de los paquetes de trabajo completados proporciona informes concisos del costo real versus el costo planificado, el calendario y el trabajo completado.

► Ejercicios

- 1. ¿Cuáles son los requisitos para realizar un buen control de un proceso de software?
- 2. Explicar las diferencias entre los gráficos de control para variables y atributos.
- 3. Un proyecto de software va en la semana 10, el valor planificado es de 80,5, pero el valor planificado real es de 73,2. Se tenía programado finalizar en la semana 12. Si se mantiene la tasa actual de trabajo, con cuántas semanas de retraso se finalizará el proyecto.
- 4. Se tienen los siguientes datos de defectos por semana encontrados durante la fase de pruebas de un proyecto de software. Desarrollar un gráfico de control X para esos datos y determinar la estabilidad del proceso.

Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10
23	35	20	18	21	28	19	32	40	15

- 5. Se tienen los siguientes datos de tiempos de desarrollo de elementos de un *sprint backlog* durante el desarrollo de un proyecto *scrum*. Desarrollar un gráfico de control de promedios para estos datos y determinar la estabilidad del proceso.

Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10
59	64	63	58	59	61	59	62	68	60
62	60	60	65	61	58	57	61	65	59
63	62	59	63	60	62	66	69	57	65
67	62	62	60	61	59	64	63	64	55
56	60	58	62	63	62	65	62	63	61

- 6. Se tienen los siguientes datos de promedios y rangos de puntos de historia de usuario por mes en cinco proyectos *scrum*. Desarrollar un gráfico de control de rangos y determinar la estabilidad del proceso.

Mes	1	2	3	4	5	6	7	8
Promedio	738	762	751	763	757	754	762	761
Rango	14	21	17	53	29	49	71	62

7. Establecer posibles límites de control de proceso para las siguientes métricas de un proceso de software: tiempos de entrega, tiempos de respuesta a soporte del cliente, ataques de ciberseguridad y costo de reparación de defectos.

► Referencias

- Baldassarre, T., Boffoli, N., Caivano, D. y Visaggio, G. (2004). Managing Software Process Improvement (SPI) through statistical process control (SPC). En F. Bomarius y H. Iida (eds.), *Product Focused Software Process Improvement. PROFES 2004. Lecture Notes in Computer Science* (pp. 30-46). Springer. https://doi.org/10.1007/978-3-540-24659-6_3
- Caivano, D. (2005). Continuous Software Process Improvement through statistical process control. En *Ninth European Conference on Software Maintenance and Reengineering* (pp. 288-293). **IEEE**. <https://doi.org/10.1109/CSMR.2005.20>
- Card, D. N. y Glass, R. L. (1990). *Measuring software design quality*. Prentice-Hall.
- Chang, C. W. y Tong, L. I. (2013). Monitoring the software development process using a short-run control chart. *Software Quality Journal*, 21, 479-499. <https://doi.org/10.1007/s11219-012-9182-y>
- DeMarco, T. (1986). *Controlling software projects: Management, measurement, and estimates*. Prentice Hall.
- Fairley, R. E. (2011). *Managing and leading software projects*. John Wiley & Sons.
- Fine, E. S. (1997). What is wrong with SPC? *Quality*, 36(10), 22-24.
- Florac, W. A. y Carleton, A. D. (1999). *Measuring the software process: Statistical process control for Software Process Improvement*. Addison-Wesley.
- Florac, W. A., Carleton, A. D. y Barnard, J. R. (2000). Statistical process control: Analyzing space shuttle onboard software process. *IEEE Software*, 17(4), 97-106. <https://doi.org/10.1109/52.854075>
- Florac, W. A., Park, R. E. y Carleton, A. (1997). *Practical software measurement: Measuring for process management and improvement*. No. **CMU/SEI-97-HB-003**. Software Engineering Institute, Carnegie Mellon University. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=76aafd5d0ed49263488bca95f00f1fdad3729bec>
- Gonçalves, L., Lima, L., Reis, R. Q., Nascimento, L. y Ribeiro, T. (2012). Support for statistic process control of software process. En *2012 XXXVIII Conferencia Latinoamericana en Informática (CLEI)* (pp. 1-10). **IEEE**. <https://doi.org/10.1109/CLEI.2012.6426915>
- Humphrey, W. S. (2005). *PSP (sm): A self-improvement process for software engineers*. Addison-Wesley.
- Jalote, P. y Saxena, A. (2002). Optimum control limits for employing statistical process control in software process. *IEEE Transactions on Software Engineering*, 28(12), 1126-1134. <https://doi.org/10.1109/TSE.2002.1158286>
- Komuro, M. (2006). Experiences of applying SPC techniques to software development processes. En *Proceedings of the 28th international conference on Software engineering* (pp. 577-584). <https://doi.org/10.1145/1134285.1134367>

- Khurana, R. (2007). *Software engineering: Principles and practices*. Vikas.
- Manlove, D. y Kan, S. H. (2007). Practical statistical process control for software metrics. *Software Quality Professional Magazine*, 9(4), 15-26.
- Montgomery, D. C. (2012). *Statistical quality control*. Wiley Global Education.
- Raczynski, B. y Curtis, B. (2008). Software data violate **SPC**'s underlying assumptions. *IEEE Software*, 25(3), 48-50.
- Salazar, R. (2019). Quality Control Charts: x-bar chart, R-chart and Process Capability Analysis. *Towards data science*. <https://towardsdatascience.com/quality-control-charts-x-bar-chart-r-chart-and-process-capability-analysis-96caa9d9233e>
- Sargut, K. U. y Demirörs, O. (2006). Utilization of statistical process control (**SPC**) in emergent software organizations: Pitfalls and suggestions. *Software Quality Journal*, 14, 135-157. <https://doi.org/10.1007/s11219-006-7599-x>
- Şengöz, N. G. (2018). Control charts to enhance quality. En L. Kounis (ed.), *Quality management systems: A selective presentation of case-studies showcasing its evolution* (pp. 153-194). IntechOpen.
- Shewhart, W. A. (1926). Quality control charts. *Bell System Technical Journal*, 5, 593-603. <https://doi.org/10.1002/j.1538-7305.1926.tb00125.x>
- Tarhan, A. y Demirörs, O. (2006). Investigating suitability of software process and metrics for statistical process control. En I. Richardson, P. Runeson y R. Messnarz (eds.), *Software Process Improvement. EuroSPI 2006. Lecture Notes in Computer Science* (pp. 88-99). Springer. https://doi.org/10.1007/11908562_9
- Weller, E. y Card, D. (2008). Applying **SPC** to software development where and why. *IEEE Software*, 25(3), 48-50.
- Wheeler, D. J. (1993). *Understanding variation: The key to managing chaos*. **SPC** Press.
- Wheeler, D. J. (1995). *Advanced topics in statistical process control*. **SPC** Press.

Mejora del proceso de software

► Objetivos

Los objetivos de este capítulo están enfocados en estudiar los conceptos subyacentes a la mejora del proceso de software. Al finalizar, el lector estará en capacidad de:

- Entender en qué consiste la mejora del proceso de software.
- Conocer diferentes modelos existentes para promover y realizar la mejora del proceso de software.
- Comprender la forma en que se evalúa la capacidad de un proceso de software.

Actualmente, en las empresas de desarrollo de software es un desafío entregar software de alta calidad de manera consistente a tiempo y dentro del presupuesto. Se presentan problemas con el presupuesto, retrasos en el cronograma de los proyectos, entrega tardía del software, sobrecostos, problemas de calidad con el software entregado, quejas de los clientes y baja moral del personal (O'Regan, 2017). Además, es esencial que el software sea seguro, confiable, de alta calidad y adecuado para su propósito de manera consistente y puntual a sus clientes. En otras palabras, las empresas necesitan trabajar continuamente de manera más inteligente para mejorar sus negocios y ofrecer soluciones superiores a sus clientes.

Las iniciativas de mejora de procesos de software deben estar alineadas con los objetivos del negocio y desempeñan un papel clave para ayudar a las empresas a alcanzar sus objetivos estratégicos. La incorporación de prácticas de mejora de procesos permite a las empresas resolver cuestiones clave para eliminar problemas de calidad y examinar críticamente sus procesos actuales, y así determinar en qué medida satisfacen sus necesidades, identificar cómo se pueden mejorar los procesos y dónde se pueden minimizar o eliminar los defectos.

► 8.1. Conceptos sobre mejora del proceso de software

Autores pioneros en calidad de procesos como Deming y Juran argumentaron que los procesos de alta calidad son esenciales para la entrega de un producto de alta calidad. Además, las mejoras en la calidad conducen a reducciones de costos y una mayor productividad, ya que se dedica menos tiempo a reelaborar los productos defectuosos.

El trabajo de Deming y Juran fue aplicado más tarde al campo de la calidad del software por Watts Humphrey en el Software Engineering Institute (SEI), lo que condujo al nacimiento del campo de mejora del proceso de software.

La mejora del proceso de software (Software Process Improvement [SPI]) es un proceso de aprendizaje organizacional continuo para obtener y mantener una ventaja competitiva en entornos empresariales y de software en rápido desarrollo (Lee *et al.*, 2018). Una iniciativa SPI consiste en un conjunto de prácticas y actividades que están diseñadas para mejorar los procesos de las organizaciones de software a través de la evaluación de sus prácticas actuales y la forma en que se desarrollan los productos y servicios de software (Mejía *et al.*, 2017).

Además, las iniciativas SPI se han convertido en una herramienta esencial para que los gerentes de proyecto y los ingenieros de software logren sus objetivos, mantengan su competitividad y proporcionen un retorno de la inversión a las organizaciones (Kuhrmann *et al.*, 2015). Además, se logra un software de mayor calidad y su entrega en forma más oportuna (Pressman, 2005). SPI puede implicar la evaluación del estado actual (por ejemplo, el nivel de madurez) seguido del desarrollo de un proceso de software futuro u objetivo que implemente las estrategias de mejora y aplique medidas correctivas al proceso (Qumer *et al.*, 2007).

Para aplicar **SPI**, se deben definir los procesos correctos y realizarles seguimiento de manera consistente. Implica capacitar al personal en los nuevos procesos, refinar los procesos y mejorarlos continuamente. La necesidad de una iniciativa **SPI** a menudo surge debido a la comprensión de que la organización es débil en algunas áreas y que necesita mejorar para lograr sus objetivos de negocio de manera más efectiva (O'Regan, 2017).

En la tabla 8.1, se muestran algunos ejemplos de atributos de proceso que pueden ser objeto de mejora.

Tabla 8.1. Atributos del proceso de software

Característica del proceso	Cuestiones clave
Comprensión	¿En qué medida el proceso está definido explícitamente y qué tan fácil es entender la definición del proceso?
Estandarización	¿Hasta qué punto el proceso se basa en el proceso genérico estándar?
Visibilidad	¿Las actividades del proceso culminan en resultados claros de modo que el avance del proceso se observa externamente?
Medible	¿El proceso incluye recolección de datos u otras actividades que permitan medir las características del proceso o del producto?
Soporte	¿En qué medida pueden usarse herramientas de software para apoyar las actividades del proceso?
Aceptabilidad	¿El proceso definido es aceptable y útil para los ingenieros responsables de elaborar el producto de software?
Fiabilidad	¿El proceso está diseñado de tal forma que se evitan o se detectan errores de proceso antes de que deriven en errores de producto?
Robustez	¿El proceso puede continuar a pesar de problemas inesperados?
Facilidad de mantenimiento	¿El proceso puede evolucionar para reflejar los requisitos cambiantes de la organización o mejoras identificadas en el proceso?
Rapidez	¿Qué tan rápido puede completarse el proceso de entrega de un sistema a partir de una especificación dada?

Fuente: elaboración propia con base en Pressman (2005).

Los beneficios de **SPI** incluyen, entre otros, mejoras en la calidad, reducciones en el costo de la mala calidad, mejoras en la productividad, reducciones en el costo del desarrollo de software, mejoras en la entrega a tiempo, consistencia mejorada en el presupuesto y la entrega programada, mejoras en la satisfacción del cliente y mejoras en la moral de los empleados. Además, favorece la entrega constante de software de alta calidad, mejores resultados financieros y una mayor satisfacción del cliente (O'Reagan, 2017).

Igualmente, el crecimiento profesional, el aumento del reconocimiento profesional, la planificación de proyectos, el monitoreo de los riesgos del proyecto, el apoyo técnico, la adopción de las tecnologías actuales, el liderazgo y el compromiso sólidos se encuentran entre los factores mejor calificados que contribuyen al éxito de las iniciativas de **SPI** (Niazi *et al.*, 2018).

► 8.2. Ciclo de mejora del proceso de software

Antes de comenzar una iniciativa **SPI**, se debe analizar el proceso de software actual. El proceso se debe valorar identificando sus debilidades y posibles cuellos de botella (Pressman, 2005). El análisis del proceso puede enfocarse considerando algunas de las características del proceso descritas en la tabla 8.1.

La tabla 8.2 muestra algunos de los aspectos del proceso que se pueden investigar durante el análisis del proceso (Pressman, 2005).

Las técnicas usadas comúnmente para el análisis del proceso son (Pressman, 2005):

- *Cuestionarios y entrevistas*: Se pregunta en forma escrita u oral a los usuarios de los procesos de software sobre cómo ejecutan el proceso, qué problemas se presentan y cuáles posibles mejoras se pueden proponer.
- *Estudios etnográficos*: Se observa y estudia cualitativamente a grupos de usuarios del proceso mientras trabajan en situaciones de la vida real. De esta manera, se revelan aspectos no susceptibles de detectarse con cuestionarios y entrevistas.

Durante el análisis de procesos, se revisan los modelos de procesos de software construidos en la organización (por ejemplo, modelos Software Process Engineering Metamodel [**SPEM**] o Business Process Model

and Notation [BPMN]). Básicamente, se deben revisar que los modelos se ejecuten realmente como están definidos. Otros aspectos a evaluar son:

- ▶ Las actividades realmente son ejecutadas por los roles definidos en el modelo de procesos.
- ▶ Se generan los productos de trabajo indicados en el modelo de procesos.
- ▶ Se realiza un análisis de la eficiencia y efectividad de las actividades del modelo de procesos.

El modelo de procesos incluye, además del comportamiento normal, flujos de actividad que reflejen excepciones o condiciones de error durante la ejecución del proceso de software.

Tabla 8.2. Aspectos del proceso de software

Aspectos del proceso	Preguntas
Estandarización	El proceso debe estar estandarizado o adaptado a diferentes tipos de productos de software que desarrolla la organización.
Prácticas de ingeniería de software	El proceso debe incluir buenas prácticas de ingeniería de software (por ejemplo, revisiones/inspecciones, gestión de configuración, etc.)
Restricciones organizacionales	Restricciones relacionadas con el proyecto o los clientes del software (por ejemplo, aspectos de seguridad y confiabilidad).
Comunicaciones	Gestión de las comunicaciones dentro del proceso y solución a los posibles conflictos y cuellos de botella que puedan surgir (por ejemplo, retrasos en entrega de artefactos o productos de trabajo).
Introspección	Existencia de una reflexión continua de los usuarios del proceso sobre su mejora (por ejemplo, la retrospectiva del <i>sprint</i> en <i>scrum</i>).
Aprendizaje	Mecanismos para que los usuarios del proceso aprendan sobre los procesos que utilizan (por ejemplo, cursos, manuales, etc.)
Soporte de herramientas	Uso de herramientas que permitan aprender, utilizar y mejorar el proceso (por ejemplo, herramientas de modelado de procesos, librerías de activos de proceso, etc.)

Fuente: elaboración propia con base en Pressman (2005).

La iniciativa SPI consiste en varios ciclos de mejora y cada ciclo implementará una o más áreas de proceso. Se pueden emplear pequeños ciclos de mejora para implementar los resultados de una evaluación o sugerencias de mejora del personal (O'Regan, 2017). El ciclo de mejora continua se describe en la figura 8.1.

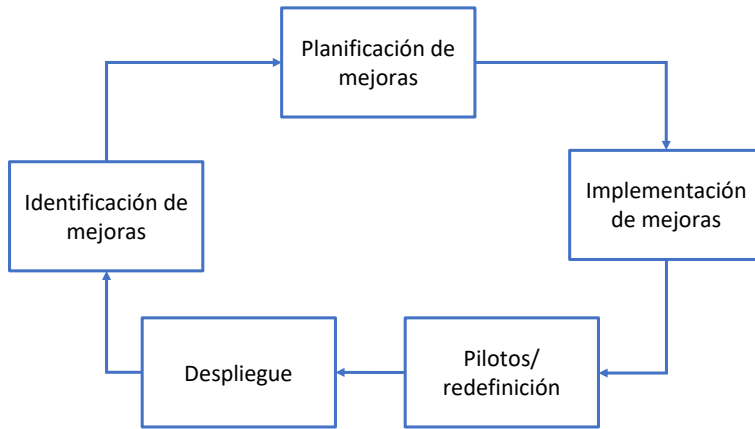


Figura 8.1. Ciclo de mejora continua del proceso de software

Fuente: elaboración propia con base en O'Regan (2017).

8.2.1. Identificación de mejoras

El cambio al proceso implica hacer modificaciones al proceso existente. En esta etapa, se utilizan los resultados del análisis del proceso para identificar las mejoras planteadas al proceso actual. La identificación de mejoras puede incluir la sugerencia de nuevos procesos, estructuras de proceso y métodos y herramientas para enfrentar los problemas identificados (Pressman, 2005).

Las mejoras provienen de diversas fuentes: sugerencias del personal, lecciones aprendidas de proyectos, revisiones periódicas del proceso o recomendaciones originadas de evaluaciones (O'Regan, 2017).

Existe una correlación entre la medición y la mejora de procesos de software. Los resultados de las fases de medición y control de procesos (capítulos seis y siete) obtienen aspectos del proceso que se pueden mejorar a partir de información objetiva que permite planificar, identificar y llevar a cabo de una manera eficiente las acciones de mejora correspondientes (Piattini Velthuis y Garzás Parra, 2007).

Al evaluar un proceso de software, se deben considerar los siguientes requisitos fundamentales (Piattini Velthuis y Garzás Parra, 2007):

- Los procesos deben producir los resultados esperados.
- Los procesos deben estar basados en una correcta definición.

- Los procesos deben ser mejorados en función de los objetivos de negocio, muy cambiantes ante la gran competitividad de las empresas.

8.2.2. Planificación de mejoras

Se debe definir el alcance del plan **SPI**. Las mejoras identificadas se deben priorizar para su implementación, de acuerdo con criterios como necesidades de mejorar áreas de proceso específicas, costos de introducir cambios, etc.

Luego, se prepara el plan y cronograma del proyecto para un ciclo de mejora y se asignan los recursos suficientes. Un plan de acción es suficiente para pequeñas iniciativas de mejora. Es esencial que una iniciativa **SPI** sea tratada como un proyecto estándar. Por tanto, se debe considerar:

- La alta gerencia debe estar cien por ciento comprometida con el éxito de la iniciativa.
- Se debe incluir un gerente de proyecto para gestionar la iniciativa.
- Es necesario que todo el personal tenga claro que la iniciativa de mejora es una prioridad para la organización.
- Los empleados deben recibir capacitación adecuada sobre la mejora del proceso.

8.2.3. Implementación de mejoras

Las mejoras consistirán en nuevos procesos, estándares, plantillas, procedimientos, listas de verificación de pautas y herramientas para respaldar el proceso. Se introducen cambios al proceso y se integran con otras actividades de este. Además, se deben seleccionar y aprobar los programas piloto iniciales del proceso de software.

8.2.4. Planes piloto/refinación

Se realizan programas piloto a los nuevos procesos seleccionados antes de su implementación para garantizar que sean adecuados para su propósito. Con la ejecución de los programas piloto, se obtiene una retroalimentación necesaria que permitirá redefinir los procesos de software analizados. Los cambios del proceso propuestos nunca serán completamente efectivos en el momento de introducirlos. Se requiere una refinación para descubrir problemas menores, y luego proponer e introducir modificaciones al proceso.

8.2.5. Despliegue

Los usuarios de los procesos modificados necesitan comprender los cambios que se propusieron y cómo realizar los procesos nuevos y modificados. Por tanto, el personal debe ser entrenado en los nuevos procesos de software. El personal recibe soporte durante el despliegue y se realizan auditorías del proceso mejorado.

8.2.6. Repetir el ciclo

La mejora del proceso es continua y tan pronto como se completa un ciclo de mejora se considera su efectividad y un nuevo ciclo de mejora inicia.

Al realizar cambios al proceso para mejorarlo, se pueden encontrar dificultades por parte de los usuarios del proceso. Algunas pueden ser (Pressman, 2005):

- *Resistencia al cambio*: Los usuarios del proceso pueden oponerse a la introducción de los cambios y, en algunos casos, podrían obstruir de manera deliberada estos. Se deben realizar capacitaciones y tener una retroalimentación continua para garantizar la resolución de problemas que se presenten generando confianza en los nuevos procesos.
- *Persistencia del cambio*: Aunque haya capacitación y utilización efectiva del proceso por parte de sus usuarios, es común que las innovaciones a los procesos se dejen de aplicar después de un corto tiempo, el entusiasmo inicial desaparece, la inconformidad con el nuevo proceso surge y los procesos vuelvan a su estado anterior.

► 8.3. Modelos de mejora de procesos de software

La mayoría de los modelos SPI tienen sus orígenes en el paradigma de planificar-hacer-verificar-actuar (Plan, Do, Check, Act [**PDCA**]) de Shewhart-Deming, que enfatiza la necesidad de una mejora cíclica y continua del proceso. Los modelos **SPI** se pueden dividir en dos categorías principales: modelos *top-down*, o prescriptivos (como **CMMI** o **ISO/IEC 15504**) y modelos *bottom-up*, o inductivos (como Quantum Information Processing [**QIP**]). Los primeros modelos se basan en las mejores prácticas prescritas sobre cómo se debe desarrollar software. Por otro lado, los segundos modelos se basan en una comprensión profunda de los procesos actuales (Pernstål *et al.*, 2016).

La mayoría de estos modelos orientan sobre qué hacer, pero no sobre cómo se debe proceder a la implementación real. Son criticados por ser demasiado extensos y consumir recursos, y por no considerar las necesidades específicas de la organización (Kuilboer y Ashrafi, 2000; Zahran, 1998).

Un modelo de mejora **SPI** define (Pressman, 2005):

1. Un conjunto de características que deben cumplirse si quiere lograrse un proceso de software efectivo.
2. Un método para evaluar si las características están presentes.
3. Un mecanismo para resumir los resultados de cualquier evaluación.
4. Una estrategia para ayudar a una organización a implementar características del proceso que sean débiles o que faltan.

La figura 8.2 proporciona una visión general de un modelo **SPI**.

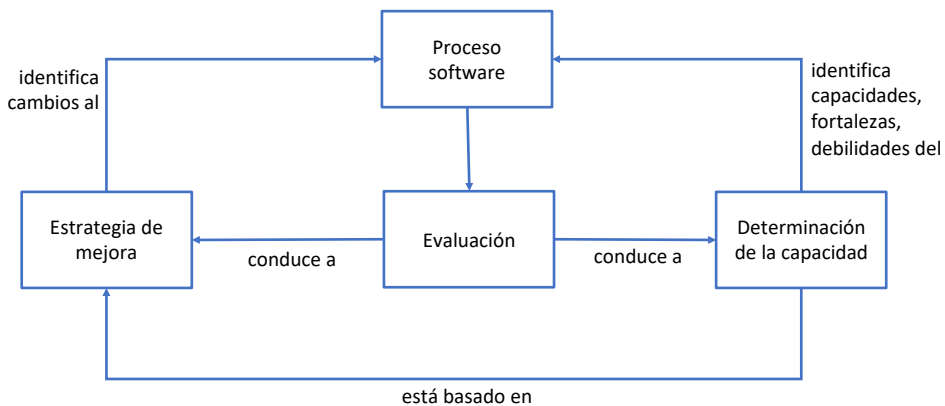


Figura 8.2. Elementos de un modelo **SPI**

Fuente: elaboración propia con base en Pressman (2005).

A continuación, se presentará una breve descripción de los principales modelos para la mejora del proceso de software utilizados ampliamente en la industria.

8.3.1. Modelo **QIP**

El paradigma de mejora de la calidad (Quality Improvement Paradigm [**QIP**]) se origina a partir de los paradigmas de la calidad de la fabricación,

como el ciclo Plan, Do, Check, Act (PDCA) y la gestión de la calidad total (Total Quality Management [TQM]). QIP es un marco metodológico básico de un sistema de infraestructura de procesos de software llamado la fábrica de experiencias (Basili *et al.*, 1994) (apartado 11.8). Se basa en el principio de que la disciplina del software es evolutiva y experimental. El ciclo QIP consta de seis pasos (figura 8.3):

1. Caracterizar el entorno.
2. Establecer un conjunto de metas cuantificables sobre la base de la caracterización inicial y de las capacidades de la organización.
3. Elegir el proceso sobre la base de la caracterización del entorno y de los objetivos organizacionales.
4. Ejecutar el proceso y proporcionar retroalimentación del proyecto.
5. Analizar los datos recopilados de cada proyecto y hacer recomendaciones para el futuro.
6. Empaquetar la experiencia en forma de modelos actualizados o conocimiento estructurado.

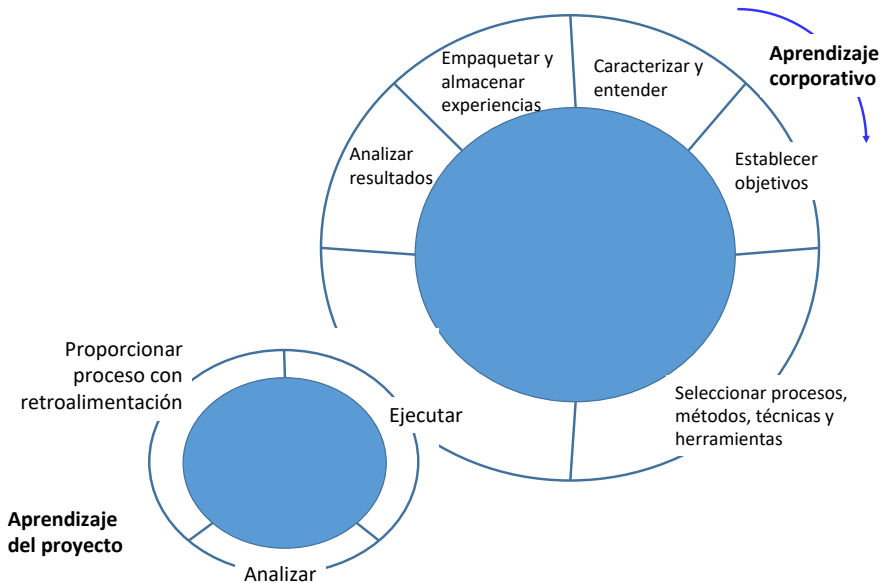


Figura 8.3. Modelo QIP

Fuente: elaboración propia con base en Basili *et al.* (1994).

Estos pasos se realizan en dos niveles, formando un ciclo de aprendizaje corporativo y un ciclo de aprendizaje de proyectos. El ciclo corporativo tiene como finalidad iniciar el proceso de cada proyecto, analizando y comparando los datos de ejecución del proceso con el conocimiento disponible y acumulando la experiencia reutilizable. El ciclo de aprendizaje del proyecto asegura la retroalimentación proporcionada al proyecto durante la fase de ejecución. La mejora del proceso se basa en el análisis de los indicadores cuantitativos del proyecto.

8.3.2. Modelo IDEAL

El modelo **IDEAL** es un programa **SPI** genérico propuesto por el **SEI** que sirve como una hoja de ruta para iniciar y gestionar iniciativas **SPI** (McFeeley, 1996). Es un modelo de cinco fases (iniciación, diagnóstico, establecimiento, actuación, aprendizaje, o Initiating, Diagnosing, Establishing, Acting, Learning [**IDEAL**]) ejecutado en un ciclo continuo en el que cada fase presenta un conjunto de tareas que se practican durante la implementación de un programa **SPI**. La estructura del modelo con sus diferentes fases se observa en la figura 8.4.

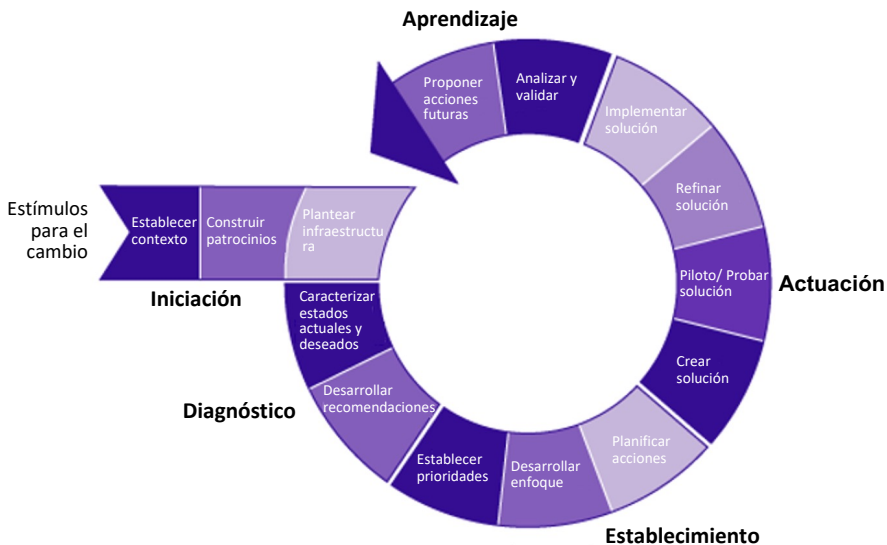


Figura 8.4. Modelo IDEAL

Fuente: elaboración propia con base en McFeeley (1996).

Las fases del modelo **IDEAL** son (McFeeley, 1996):

- *Iniciación*: Se establece la infraestructura de mejora inicial identificando un grupo directivo de gestión (Management Steering Group [**MSG**]) y un grupo de proceso de ingeniería de software (Software Engineering Process Group [**SEPG**]). Se crea un plan **SPI** para guiar a la organización a través del proceso de cambio y se definen los objetivos generales de **SPI**. Esta fase se concentra en establecer el contexto **SPI** de la empresa, así como los patrocinadores y la infraestructura **SPI**.
- *Diagnóstico*: El plan **SPI** se inicia de acuerdo con la visión de la organización, el plan estratégico de negocios, las lecciones aprendidas de esfuerzos de mejora anteriores, los problemas de negocio que enfrenta la organización y los objetivos a largo plazo. Esta fase se concentra en evaluar y caracterizar el estado actual y deseado de la organización, así como en desarrollar las recomendaciones y los resultados de los documentos que servirán de base para el plan de acción de **SPI**.
- *Establecimiento*: Se abordan los problemas que llevaron a la organización a un proceso **SPI**. Se desarrollan estrategias para alcanzar las soluciones, se completa el borrador del plan de acción **SPI**, se definen metas y métricas medibles, y se brinda capacitación para los grupos de trabajo técnico. Esta fase se centra en establecer prioridades **SPI**; desarrollar estrategias, objetivos medibles, métricas y recursos, y planificar las acciones de mejora.
- *Actuación*: Se crean las soluciones a los problemas, se ponen a prueba y se implementan en la organización. Luego, se realiza el despliegue y la institucionalización del proceso mejorado. Esta fase se centra en planificar, ejecutar y seguir el plan de acción; crear las soluciones; realizar un programa piloto y pruebas de la solución; refinar la solución con base en los resultados obtenidos por el programa piloto/pruebas realizadas, e implementar la solución.
- *Aprendizaje*: Se prepara a la organización para próximos programas de **SPI**. Las lecciones aprendidas, las soluciones desarrolladas y las métricas y los objetivos recopilados se agregan a una base de datos para servir como fuente de información para proyectos futuros. Se realizan correcciones o ajustes a la estrategia, métodos o infraestructura en función de las evaluaciones de la información recopilada. Esta fase se concentra en analizar y validar las lecciones

aprendidas que se transmitirán al nuevo ciclo de mejora, y proponer acciones futuras de mejora organizacional.

El modelo **IDEAL** tiene un enfoque muy técnico con mayor énfasis en qué implementar que en cómo implementar (Kandt, 2003). Hace énfasis en los métodos y las herramientas para implantar **SPI** sobre los individuos y las interacciones, y este enfoque puede socavar el éxito de la iniciativa (Ferreira y Wazlawick, 2011).

8.3.3. Modelo **CMMI**

El **CMMI** tiene como objetivo final facilitar que una organización desarrolle productos o soluciones mejorando su capacidad para gestionar el desarrollo, la adquisición y el mantenimiento de sus productos o servicios (Chaudhary y Chopra, 2017).

CMMI es un modelo de mejora de capacidades que se puede adaptar para resolver cualquier problema de rendimiento en cualquier nivel de las organizaciones. El modelo proporciona pautas y recomendaciones para diagnosticar problemas y mejorar el rendimiento (**SEI**, 2010). **CMMI** fue desarrollado por el **SEI** en la Carnegie Mellon University y actualmente aborda tres áreas de interés importantes:

CMMI para desarrollo (**CMMI-DEV**): Orientado al desarrollo de productos y servicios. Se utiliza para mejorar el desarrollo de soluciones.

CMMI para servicios (**CMMI-SVC**): Orientado al establecimiento y la gestión de servicios. Se utiliza para mejorar la entrega de servicios y la creación de sistemas de servicios para la operación de soluciones.

CMMI para adquisición (**CMMI-ACQ**): Adquisición de productos y servicios. Se utiliza para mejorar la compra/adquisición de productos, servicios y soluciones.

8.3.3.1. Componentes del área del proceso

Los componentes del modelo **CMMI** y sus relaciones se encuentran resumidos en la figura 8.5.

Los componentes elementales del modelo son (**SEI**, 2010):

- ▶ *Áreas de proceso*: Es un grupo de prácticas relacionadas que dan cumplimiento a una colección de metas para mejorar una determinada área de proceso.

- *Declaraciones del propósito*: Describen la finalidad del área de proceso.
- *Notas introductorias*: Describen los conceptos principales de un área de proceso.
- *Áreas de proceso relacionadas*: Establecen referencias a otras áreas de proceso.
- *Objetivos específicos*: Describen características para satisfacer un área de proceso.
- *Objetivos genéricos*: Describen características que se aplican a múltiples áreas de proceso.
- *Prácticas específicas*: Descripción de una actividad para lograr un objetivo específico asociado.
- *Ejemplos de productos de trabajo*: Enumeran resultados de muestra de una práctica específica.
- *Subprácticas*: Descripción detallada que ayuda a implementar prácticas específicas o genéricas.
- *Prácticas genéricas*: Se aplican a múltiples áreas de proceso y están asociadas a un objetivo genérico.
- *Elaboraciones de la práctica genérica*: Permiten orientar la forma en que se pueden aplicar las prácticas genéricas a las áreas de proceso.

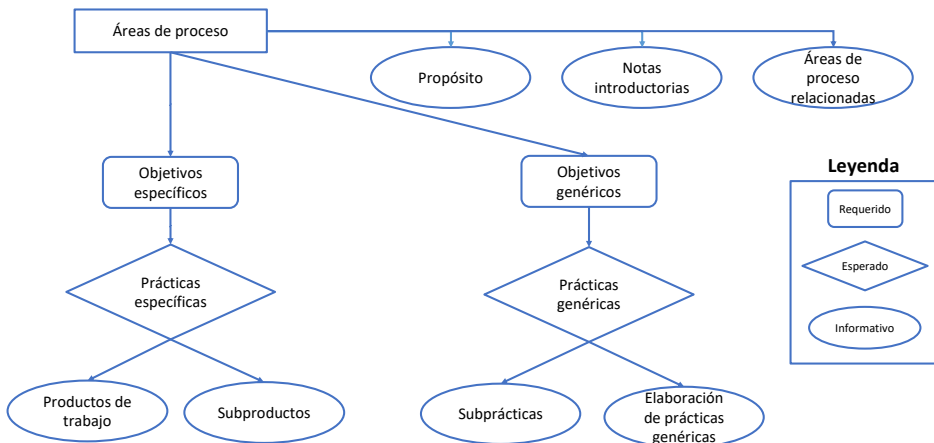


Figura 8.5. Componentes del modelo CMMI

Fuente: elaboración propia con base en SEI (2010).

8.3.3.2. Áreas de proceso

CMMI-DEV 1.3 contiene 22 áreas de proceso agrupados en cuatro categorías. En la tabla 8.3, se presenta una descripción de cada una de estas áreas.

Tabla 8.3. Áreas de proceso **CMMI**

Categoría	Área de proceso	Propósito
Gestión de proyectos	Planificación del proyecto (PP)	Establecer y mantener planes que definan las actividades del proyecto.
	Gestión integrada de proyectos (IPM, por sus siglas en inglés)	Establecer y gestionar el proyecto y la participación de las partes interesadas relevantes de acuerdo con un proceso integrado y definido.
	Monitorización y control de proyectos (PMC, por sus siglas en inglés)	Proporcionar una comprensión del progreso del proyecto, y así tomar las acciones correctivas apropiadas cuando el rendimiento del proyecto se desvíe del plan.
	Gestión de requisitos (REQM, por sus siglas en inglés)	Gestionar los requisitos de los productos y los componentes de producto del proyecto.
	Gestión de riesgos (RSKM, por sus siglas en inglés)	Identificar problemas potenciales para planificar las actividades de gestión de riesgos y mitigar los impactos sobre la consecución de objetivos.
	Gestión de acuerdo con proveedores (SAM, por sus siglas en inglés)	Gestionar la adquisición de productos y servicios de proveedores.
	Gestión cuantitativa de proyectos (QPM, por sus siglas en inglés)	Gestionar cuantitativamente el proyecto para lograr los objetivos de calidad y de rendimiento del proceso en el proyecto.
Ingeniería	Desarrollo de requisitos (RD, por sus siglas en inglés)	Analizar y establecer los requisitos de cliente, de producto y de componentes de producto.
	Solución técnica (TS, por sus siglas en inglés)	Seleccionar, diseñar e implementar soluciones para los requisitos.
	Integración de producto (PI, por sus siglas en inglés)	Ensamblar el producto a partir de sus componentes asegurando que el producto se comporta correctamente.
	Validación (VAL)	Demostrar que un producto o componente cumple con su uso previsto cuando se utiliza en un entorno específico.
	Verificación (VER)	Asegurar que los productos de trabajo seleccionados cumplen los requisitos especificados.

Categoría	Área de proceso	Propósito
Gestión de procesos	Gestión del rendimiento de la organización (OPM, por sus siglas en inglés)	Gestionar proactivamente el rendimiento de la organización para satisfacer sus objetivos de negocio.
	Definición de procesos de la organización (OPD, por sus siglas en inglés)	Establecer y mantener un conjunto de activos de proceso de la organización, estándares del entorno de trabajo y reglas y guías para los equipos.
	Enfoque en procesos de la organización (OPF, por sus siglas en inglés)	Planificar, implementar y desplegar las mejoras de proceso de la organización, basadas en una comprensión de sus fortalezas y debilidades.
	Formación en la organización (OT, por sus siglas en inglés)	Desarrollar las habilidades y los conocimientos de las personas para que puedan desempeñar sus roles eficaz y eficientemente.
	Rendimiento de procesos de la organización (OPP, por sus siglas en inglés)	Establecer y mantener una comprensión cuantitativa del rendimiento de los procesos para dar soporte a la consecución de los objetivos de calidad y de rendimiento del proceso.
Soporte	Análisis causal y resolución (CAR, por sus siglas en inglés)	Identificar las causas de los resultados seleccionados y actuar para mejorar el rendimiento del proceso.
	Gestión de configuración (CM, por sus siglas en inglés)	Establecer y mantener la integridad de los productos de trabajo aplicando actividades de gestión de configuración.
	Análisis de decisiones y resolución (DAR, por sus siglas en inglés)	Analizar las posibles decisiones utilizando un proceso de evaluación formal que evalúa las alternativas identificadas frente a unos criterios establecidos.
	Medición y análisis (MA)	Desarrollar y mantener la capacidad de medición utilizada para dar soporte a las necesidades de información de la gerencia.
	Aseguramiento de la calidad del proceso y del producto (PPQA, por sus siglas en inglés)	Proporcionar al personal y a la gerencia una visión objetiva de los procesos y de los productos de trabajo asociados.

Fuente: elaboración propia con base en [SEI](#) (2010).

CMMI cuenta con dos tipos de representaciones para describir el camino de evolución de la mejora de procesos: la representación por etapas y la representación continua. A continuación, se describen los dos tipos de representaciones.

8.3.3.3. Representación por etapas

Esta representación permite a las organizaciones mejorar incrementalmente un conjunto de áreas de proceso relacionadas, está asociada con el nivel de madurez (nivel en un conjunto de áreas de proceso en las que se alcanzan todos sus objetivos) y se enfoca en la madurez global de la organización (SEI, 2010).

Esta representación contiene cinco niveles de madurez que permiten mejorar los procesos. Los niveles de madurez se numeran del 1 al 5 y se miden evaluando el cumplimiento de los objetivos específicos y genéricos asociados con las áreas de procesos que conforman un nivel especificado (figura 8.6).

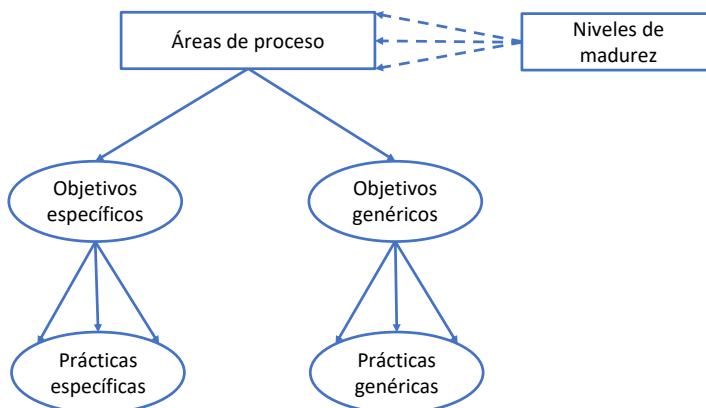


Figura 8.6. Estructura de la representación por etapas

Fuente: elaboración propia con base en SEI (2010).

Los cinco niveles de madurez de la representación por etapas del modelo **CMMI** son (SEI, 2010):

- ▶ *Nivel de madurez 1, inicial:* Los procesos de software son caóticos. El éxito depende de la competencia y la heroicidad del personal y no del uso de procesos previamente definidos y probados.
- ▶ *Nivel de madurez 2, gestionado:* Los procesos se planifican y ejecutan de acuerdo con políticas y planes establecidos; los proyectos utilizan personal cualificado para desarrollar productos; se involucra a las partes interesadas relevantes, y se monitorizan, controlan y revisan los proyectos.

- *Nivel de madurez 3, definido:* Los procesos están bien caracterizados, entendidos y utilizados. Los procesos se describen por medio de estándares, procedimientos, herramientas y métodos. Además, se establece un conjunto de procesos estándar de la organización. Los proyectos adaptan el conjunto de procesos estándar a sus características particulares de acuerdo con unas determinadas guías de adaptación.
- *Nivel de madurez 4, gestionado cuantitativamente:* La organización establece objetivos cuantitativos para medir la calidad y el rendimiento del proceso, y los utilizan como criterios en la gestión de los proyectos. La calidad y el rendimiento del proceso se interpretan en términos estadísticos y se gestionan durante la vida de los proyectos.
- *Nivel de madurez 5, en optimización:* Una organización mejora continuamente sus procesos por medio de una comprensión cuantitativa de sus objetivos de negocio y necesidades de rendimiento.

Cada nivel de madurez debe cumplir un conjunto de áreas de proceso. En la figura 8.7, se presentan los niveles de madurez con las respectivas áreas de proceso que se deben cumplir.

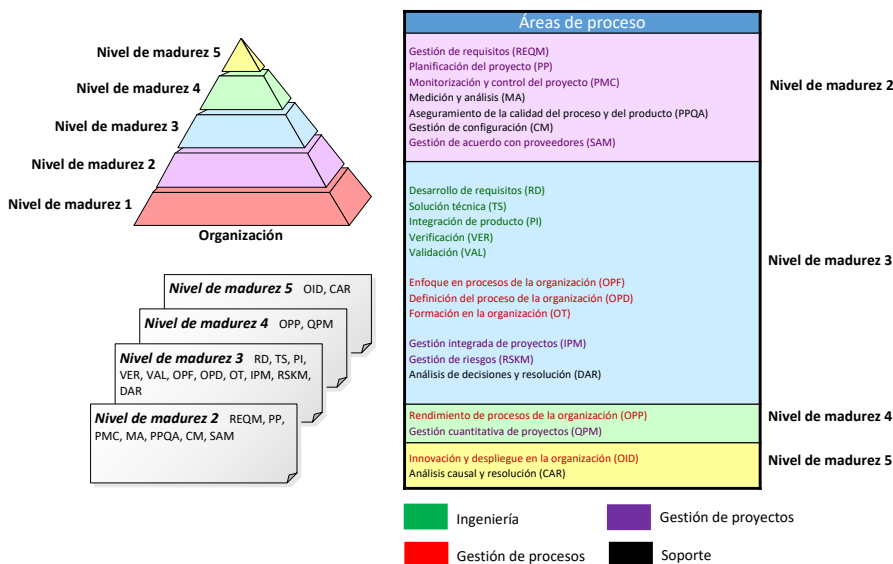


Figura 8.7. Niveles de madurez con áreas de proceso

Fuente: elaboración propia con base en sei (2010).

8.3.3.4. Representación continua

La representación continua permite a las organizaciones mejorar incrementalmente un área de proceso individual (o un grupo de áreas de proceso) seleccionada. Esta representación está asociada con el nivel de capacidad (logro en la mejora de procesos dentro de un área de proceso individual). El cumplimiento de las metas específicas de cada área de proceso especificará el nivel de capacidad obtenido, como se observa en la figura 8.8.

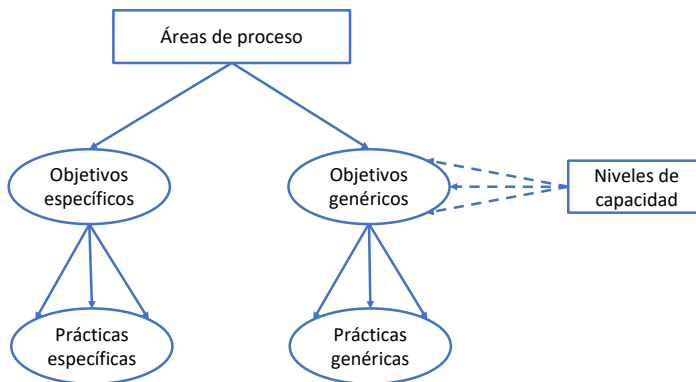


Figura 8.8. Estructura de la representación continua

Fuente: elaboración propia con base en SEI (2010).

Los cuatro niveles de capacidad para una determinada área de proceso se denominan por los números del 0 al 3 (SEI, 2010):

- ▶ *Nivel de capacidad 0, incompleto:* Al menos uno de los objetivos específicos del área de proceso no se satisface y no existen objetivos genéricos para ese nivel.
- ▶ *Nivel de capacidad 1, realizado:* Se realiza el trabajo necesario para producir los productos de trabajo. Se satisfacen los objetivos específicos del área de proceso.
- ▶ *Nivel de capacidad 2, gestionado:* El proceso se planifica y ejecuta de acuerdo con las políticas; utiliza personal cualificado para producir resultados controlados; involucra a las partes interesadas relevantes; se monitoriza, controla y revisa el proceso, y se evalúa su conformidad con respecto a la descripción del proceso.
- ▶ *Nivel de capacidad 3, definido:* El proceso se ajusta aplicando guías de adaptación al conjunto de procesos estándar de la organización.

La representación continua permite identificar dónde se concentrarán los esfuerzos de mejora de procesos de la organización, seleccionando aquellas áreas de proceso o conjuntos de áreas de proceso interrelacionados que sean más relevantes a la organización (figura 8.9).



Figura 8.9. Áreas de proceso en representación continua
Fuente: elaboración propia con base en SEI (2010).

8.3.3.5. Diferencias entre las dos representaciones

Tabla 8.4. Comparación entre representación continua y por etapas

Nivel	Representación continua Nivel de capacidad	Representación por etapas Nivel de madurez
0	Incompleto	
1	Realizado	Inicial
2	Gestionado	Gestionado
3	Definido	Definido
4		Gestionado cuantitativamente
5		En optimización

Fuente: elaboración propia con base en SEI (2010).

La tabla 8.4 realiza una comparación entre los cuatro niveles de capacidad con los cinco niveles de madurez del modelo **CMMI**. Los nombres de los niveles gestionado y definido son los mismos en ambas representaciones y corresponden a los niveles 2 y 3. El nivel 0 no existe en la representación por etapas. No hay niveles de capacidad 4 y 5 en la representación continua. Por último, el nivel 1 tiene nombres distintos en cada representación (realizado en representación continua e inicial en representación por etapas).

8.3.3.6. Beneficios de **CMMI**

El objetivo principal de **CMMI** es mejorar el rendimiento existente de los estándares, procesos y procedimientos de la organización. **CMMI** también es muy útil para las organizaciones que desean mejorar su capacidad para entregar de manera consistente y previsible los productos y servicios que desean sus clientes. Por tanto, también ayuda a las organizaciones a lograr sus objetivos de desempeño. Entre algunos de los beneficios de implementar el modelo **CMMI-DEV**, están (Chaudhary y Chopra, 2017):

- ▶ Entrega de productos dentro del presupuesto y en los plazos especificados.
- ▶ Mejora de la productividad en un periodo específico.
- ▶ Mantiene la satisfacción del cliente.
- ▶ Mejora del tiempo de comercialización de nuevos productos.
- ▶ Reduce la tasa de retiros de productos.
- ▶ Disminuye el costo de mantener productos heredados.

8.3.4. Modelo **TSP**

Gran parte de los procesos de desarrollo de software son realizados por equipos de trabajo conformados por desarrolladores individuales. El proceso de software en equipo (Team Software Proces [**TSP**]) es un modelo propuesto por Watts Humphrey en el **SEI**, diseñado para construir y mantener equipos de desarrollo motivados y comprometidos. Los requisitos para que los equipos estén comprometidos son (Humphrey, 2001):

- ▶ Los miembros del equipo tienen las habilidades y son capaces de realizar su trabajo.
- ▶ El equipo tiene un objetivo importante y agresivo que los miembros deben cumplir cooperativamente.

- ▶ Los miembros del equipo creen que el objetivo es alcanzable y cada uno tiene un rol definido para lograr el objetivo.
- ▶ Los miembros del equipo tienen un proceso común y un plan que los guía para realizar el trabajo y hacer seguimiento de su progreso.
- ▶ El líder del equipo apoya y protege al equipo y mantiene a los miembros informados de su progreso y de cuestiones de administración.

El proceso **TSP** se presenta en la figura 8.10. Se requiere preparación del equipo para que todos sus miembros cuenten con las habilidades de planificación, medición y seguimiento. El entrenamiento de los miembros se basa en los principios de conocer el proceso que es utilizado, cómo medir y gestionar la calidad y cumplir consistentemente con los compromisos.

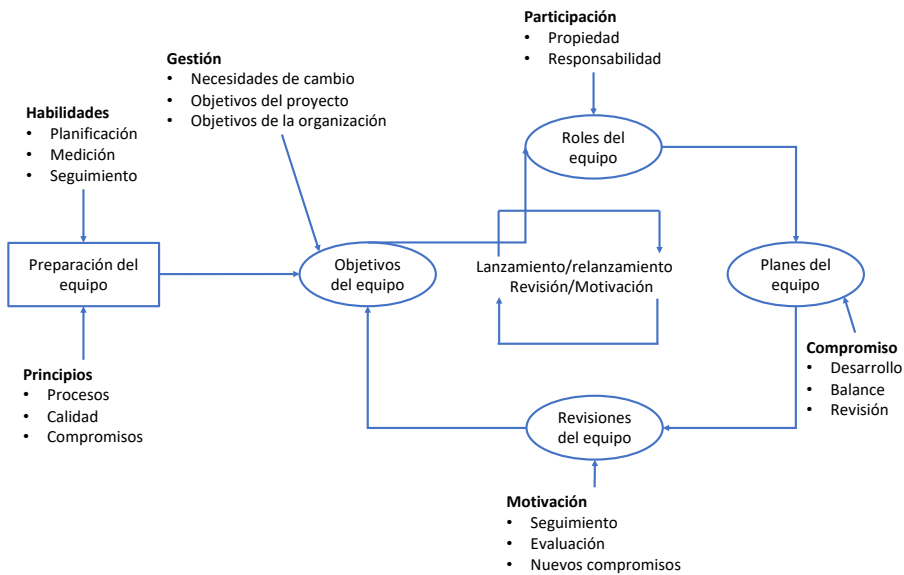


Figura 8.10. El proceso TSP

Fuente: elaboración propia con base en Humphrey (2001).

Durante el lanzamiento (*launch*) del proceso **TSP**, el equipo establece sus objetivos, asigna roles a los miembros del equipo y desarrolla un plan del equipo. A su vez, cada miembro desarrolla planes personales de su propio trabajo. Cuando se finaliza el lanzamiento, el equipo ejecuta su trabajo por medio de una estrategia de desarrollo cíclico y realiza revisiones

periódicas del estado del trabajo, se evalúa el desempeño de los miembros y se realizan reuniones de gestión para mantener el compromiso y la motivación. Después de cierto tiempo, se realiza un relanzamiento (*relaunch*) para actualizar sus planes ajustándolos a los cambios en los requisitos. El relanzamiento permite que los planes del equipo estén actualizados. Entre el lanzamiento/relanzamiento y el *post mortem*, el equipo gestiona su trabajo tarea por tarea, día a día y semana a semana por medio de reuniones semanales estructuradas (Humphrey y Over, 2010).

TSP hace énfasis en la actividad de lanzamiento, la cual puede tener una duración aproximada de 4 días, como se muestra en la figura 8.11. El objetivo del lanzamiento es que el equipo logre un compromiso y construya conjuntamente un plan detallado.

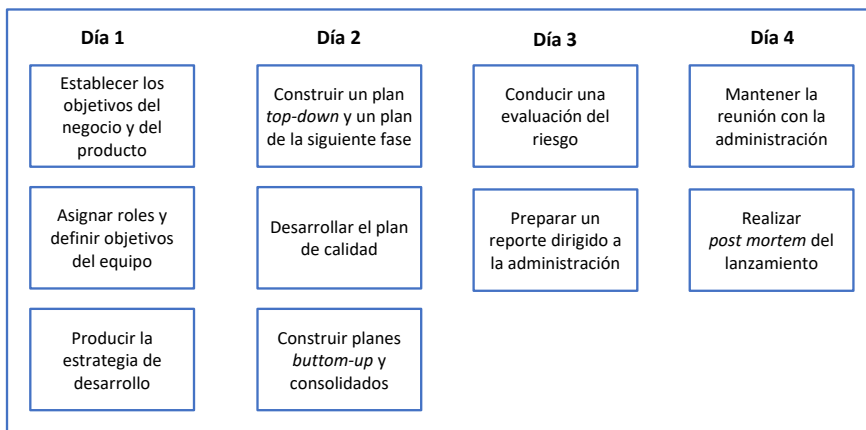


Figura 8.11. El proceso de lanzamiento TSP

Fuente: elaboración propia con base en Humphrey (2001).

Las actividades que conforman la actividad de lanzamiento de **TSP** son (Humphrey, 2001):

- ▶ *Establecer los objetivos del negocio y del producto*: Se describe la importancia del proyecto para la organización.
- ▶ *Asignar roles y definir objetivos del equipo*: Los roles construyen la pertenencia al equipo y proporcionan un grado de control sobre el proceso.
- ▶ *Producir la estrategia de desarrollo*: El equipo define su propio proceso de desarrollo incluyendo las tareas que realizará.

- *Construir un plan top-down y un plan de la siguiente fase:* El equipo construye un plan detallado (cronograma) para cada una de las tareas planificadas.
- *Desarrollar el plan de calidad:* El equipo establece un plan de calidad basado en métricas de proceso y producto.
- *Construir planes bottom-up y consolidados:* Se realizan planes detallados del trabajo a realizar por cada individuo y se consolidan en un plan general del proyecto.
- *Conducir una evaluación del riesgo:* Se consideran los riesgos priorizándolos por probabilidad e impacto y desarrollando acciones de mitigación para estos. Los riesgos más prioritarios se asignan a diferentes miembros para su seguimiento.
- *Preparar un reporte del lanzamiento dirigido a la administración:* El equipo realiza un informe de todas las actividades desarrolladas en el lanzamiento.
- *Mantener la reunión con la administración:* Se realiza la reunión con la administración para obtener su retroalimentación y aprobación de los compromisos adquiridos por el equipo.
- *Realizar post mortem del lanzamiento:* Se realiza una revisión del lanzamiento para identificar mejores prácticas, lecciones aprendidas y oportunidades de mejora.

8.3.5. Modelo PSP

El proceso personal de software (Personal Software Process [PSP]) es un modelo desarrollado por Watss Humphrey en el SEI para ampliar el concepto de mejora del proceso de software del nivel organizacional al individual (Humphrey, 2005). PSP es un proceso definido, medido y diseñado para ser utilizado de manera individual por desarrolladores de software. PSP se enfoca en los siguientes objetivos (Grütter y Ferber, 2002):

- Mejorar las prácticas y habilidades individuales de desarrollo.
- Identificar fácilmente áreas de mejora en el proceso.
- Permitir hacer estimaciones de tamaño, tiempo y defectos.

PSP fue concebido para ayudar a los desarrolladores a construir productos de alta calidad a través de la mejora de su proceso de desarrollo de software. PSP sugiere actividades de medición y análisis consistentes para que los desarrolladores puedan identificar deficiencias del proceso y hacer una estimación confiable de esfuerzo y calidad (Shin *et al.*, 2007).

PSP está conformado con un conjunto de prácticas que generan un proceso disciplinado y sólido para desarrollar software. Estas prácticas se aplican en varios niveles de procesos en los que cada nivel incluye un conjunto de actividades para el desarrollo de pequeños programas en forma individual (Borstler *et al.*, 2002). Los niveles de **PSP** se muestran en la figura 8.12. Cada nivel incorpora nuevas destrezas y técnicas para mejorar la calidad de software e incluye un conjunto de *scripts*, *checklists* y plantillas para guiar al desarrollador a través de los pasos requeridos. Los elementos del proceso incluidos en cada nivel son incrementales de tal manera que los elementos de un nivel superior incluyen los elementos de proceso de niveles inferiores. Se recomienda aplicar el proceso de manera secuencial iniciando con **PSP0**.

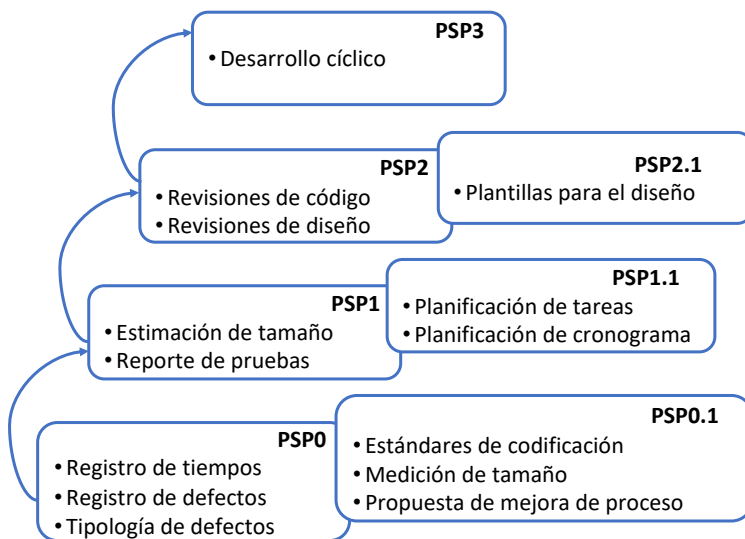


Figura 8.12. Niveles de PSP

Fuente: elaboración propia con base en Humphrey (2005).

Los niveles del modelo **PSP** son los siguientes:

- ▶ **PSP0**: Línea base del proceso a partir de la recolección de datos de tiempos y defectos.
- ▶ **PSP0.1**: Incorpora estándares de codificación, estándares de conteo basados en LOC para la medición del tamaño de los programas y un mecanismo de propuesta para la mejora del proceso.

- **PSP1:** Incluye prácticas de estimación de tamaño y tiempo basado en el método PROxy Based Estimating (**PROBE**) utilizando datos históricos de proyectos. Además, incluye la realización de reportes de prueba.
- **PSP1.1:** Involucra prácticas de planificación y seguimiento de proyectos utilizando la técnica del valor ganado.
- **PSP2:** Incluye aspectos de calidad del proceso de software a partir de nuevas actividades, como revisiones de diseño y de código basadas en el uso de *checklists*. Además, se incorpora un conjunto de métricas para evaluar la calidad del proceso y del producto desarrollado.
- **PSP2.1:** Introduce plantillas para especificar en forma detallada el diseño de programas y técnicas para realizar la verificación del diseño.
- **PSP3:** Amplía el proceso para desarrollar programas grandes en forma individual aplicando un enfoque iterativo e incremental.

PSP contiene cuatro elementos básicos que están incluidos en los niveles del proceso personal y forman un *framework* para organizar el trabajo del proyecto (Pomeroy-Huff *et al.*, 2008). Estos elementos se presentan en la figura 8.13.

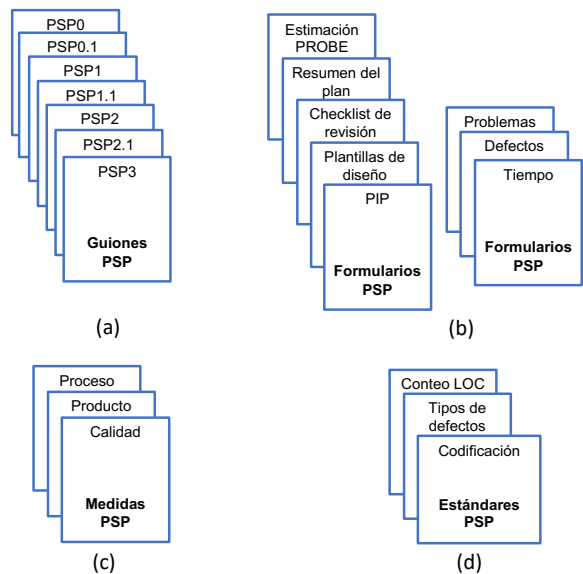


Figura 8.13. Elementos del proceso PSP

Fuente: elaboración propia con base en Humphrey (2005).

- ▶ *Scripts*: Son descripciones a nivel experto que guían la ejecución personal del proceso especificando objetivos del proceso, criterios de entrada, etapas a ejecutar, medidas del proceso, criterios de calidad y criterios de salida. Hay un *script* por cada nivel PSP.
- ▶ *Formularios y logs*: Proporcionan un medio para recolectar datos. Especifican los datos requeridos y dónde recolectarlos. Los *checklists* son formularios especializados para guiar las revisiones personales de diseño y código, entre los cuales está el resumen de plan de proyecto, la plantilla de estimación **PROBE** y los *checklists* de revisión. Entre los diferentes *logs* se encuentran el *log* de tiempo, el *log* de defectos y el *log* de problemas.
- ▶ *Medidas*: Cuantifican los procesos y productos permitiendo desarrollar perfiles de datos de proyectos previos que pueden ser utilizados para planificación, analizar un proceso, determinar cómo mejorarlo y monitorear su ejecución.
- ▶ *Estándares*: Proporcionan definiciones consistentes y precisas que guían el trabajo y la recolección y el uso de datos. Estándares de codificación, conteo y tipos de defectos permiten realizar mediciones que son aplicadas uniformemente a través de múltiples proyectos.

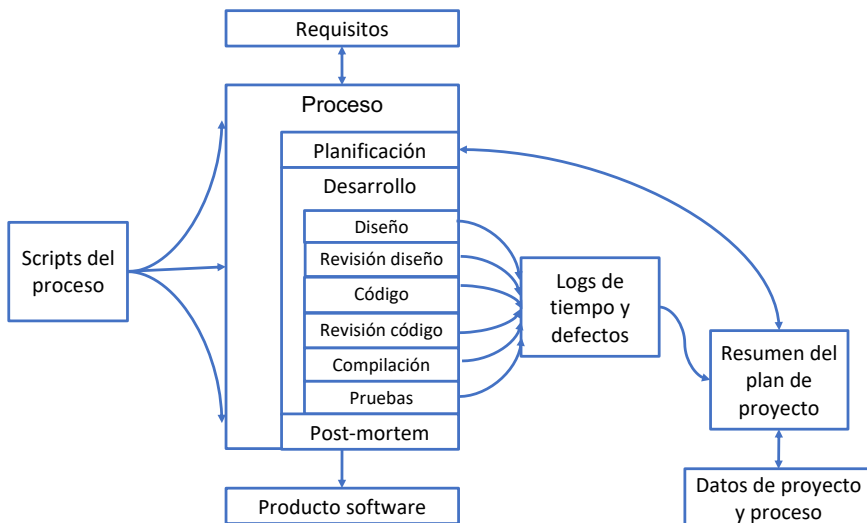


Figura 8.14. Proceso PSP

Fuente: elaboración propia con base en Humphrey (2005).

El proceso **PSP** se detalla en la figura 8.14. El proceso se ejecuta consultando los *scripts*. A medida que se ejecuta cada fase, se van recolectando datos a través de formularios y *logs*. Los datos del proyecto y del proceso se consolidan en un resumen de plan de proyecto. Este proceso general se aplica a todos los niveles **PSP**. El proceso está conformado por tres fases (Pomeroy-Huff *et al.*, 2009):

- *Planificación*: Se produce un plan de realización del programa a desarrollar.
- *Desarrollo*: Se construye el programa planificado. Esta fase se descompone en las siguientes actividades:
 - *Diseño*: Se produce un diseño del programa.
 - *Revisión del diseño*: Se revisa el diseño y se corrigen defectos detectados.
 - *Codificación*: Se codifica el programa en un determinado lenguaje de programación.
 - *Revisión de código*: Se revisa el código y se corrigen defectos detectados.
 - *Compilación*: Se compila el programa y se corrigen defectos detectados.
 - *Pruebas*: Se prueba el programa y se corrigen defectos detectados.
- *Post mortem*: Se compara el desempeño real con el desempeño planificado, se registran datos del proceso, se produce un resumen del plan de proyecto y se documentan ideas de mejora del proceso.

Para el desarrollo de cada una de estas fases, se siguen los *scripts* del proceso **PSP**. Este permitirá, a partir de un conjunto de requisitos, construir un producto de software aplicando un proceso que incluya las mejores prácticas de desarrollo de la industria y aplicadas de manera incremental.

8.3.6. Modelo Six Sigma

Six Sigma es un enfoque estructurado basado en datos para mejorar los procesos de fabricación y eliminar defectos (Pillai *et al.*, 2012). La base del modelo Six Sigma es el ciclo definir, medir, analizar, mejorar, controlar (Define, Measure, Analyze, Improve, Control [**DMAIC**]). Este ciclo está inspirado en el ciclo **PDCA** de Deming. **DMAIC** ofrece una metodología estructurada y disciplinada para resolver problemas de negocio y permite que una empresa logre tasas de incumplimiento extremadamente bajas (Harry

y Schroeder, 2000). **DMAIC** es útil para que los procesos existentes mejoren su calidad y previsibilidad.

Las principales intenciones de esta metodología son (Pournaghshband y Watson, 2017):

1. Reducir la variación del proceso.
2. Mejorar la calidad de los procesos de producción haciéndolos más consistentes en términos de los resultados finales alcanzados para satisfacer y preferiblemente superar las expectativas de los clientes.
3. Disminuir el número de defectos.
4. Mejorar la capacidad de los procesos que se utilizan para producir productos o servicios.

Six Sigma está orientado a los datos, es decir, se refiere a tener seis desviaciones estándar entre la media del rendimiento real del proceso y los límites de rendimiento esperados de este desde la perspectiva del cliente. Por tanto, se tiene una probabilidad del 99,9997% de que el rendimiento del proceso se ajuste a los límites deseados o menos de 3,4 fallas por un millón de casos (Bubevski, 2010). La calidad de los datos utilizados para el análisis es el factor crítico para el éxito de Six Sigma.

Los métodos y las técnicas Six Sigma se aplican en proyectos empresariales y de la tecnología de la información (TI) para el diseño y la mejora de productos y procesos. El comienzo de la metodología data de mediados de la década de 1980. Mikel Harry, junto con Bill Smith, la desarrollaron en Motorola para la solución de problemas siguiendo un enfoque disciplinado. La metodología alcanzó su mayor éxito a mediados de la década de 1990 cuando General Electric la adoptó como su principal estrategia de negocios.

Six Sigma se puede considerar la aplicación del método científico al proceso de mejora (figura 8.15). A continuación, se describen cada una de las fases de Six Sigma.

- ▶ *Fase de definición:* Las entregas principales de esta fase son una declaración del alcance, de los objetivos y de los participantes del proyecto, de tal manera que se identifica el problema a ser solucionado. También en esta fase el equipo desarrolla una lista de características críticas para la calidad.
- ▶ *Fase de medición:* Su objetivo es cuantificar el problema mediante la obtención desde el proceso actual de tanta información como sea posible. Se debe determinar con precisión cómo funciona el proceso. Las tareas principales son la creación de un mapa de proceso

detallado, la recopilación de datos de referencia y un resumen de los datos recopilados.

- *Fase de análisis:* Se establece el equipo para identificar las causas del problema en estudio. Estas deben ser validadas por los datos. Por tanto, se debe realizar un análisis estadístico profundo que distinga a Six Sigma de otras estrategias de resolución de problemas. Estas técnicas estadísticas incluyen análisis de varianza (Analysis of Variance [ANOVA]), análisis de correlación, diagramas de dispersión o análisis de chi-cuadrado, entre otras.

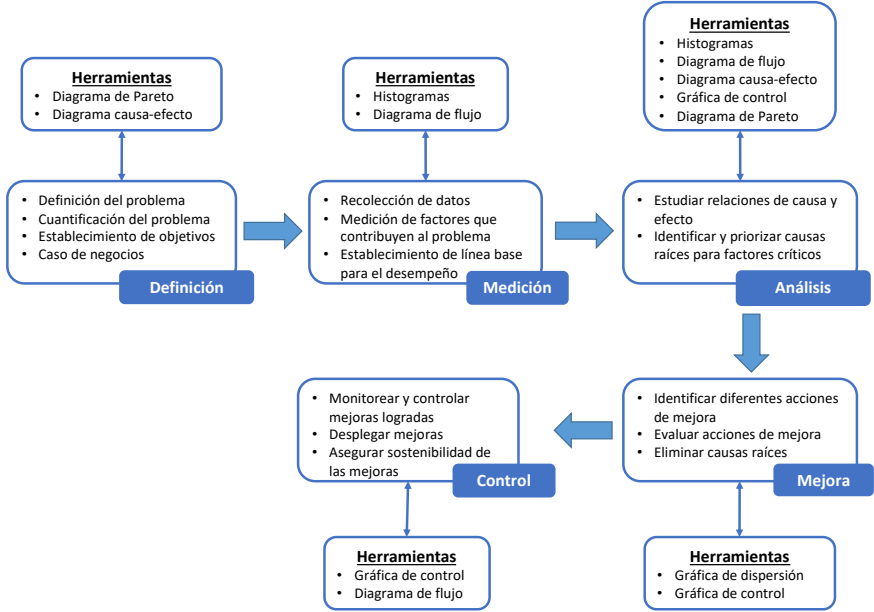


Figura 8.15. Ciclo Six Sigma
Fuente: elaboración propia.

- *Fase de mejora:* Se determina una solución al problema. Para generar un conjunto de soluciones potenciales, se puede utilizar una lluvia de ideas y una combinación de experiencia y análisis científico. Antes de la implementación, el equipo debe asegurarse de que la solución propuesta realmente funcione. Se pueden utilizar técnicas, como programas piloto, simulaciones por computador e implementación segmentada para garantizar el éxito. Por último, el equipo también debe crear un mapa de proceso del estado futuro.

- *Fase de control*: Su objetivo es garantizar la sostenibilidad de la mejora que fue realizada al proceso. Además, se desarrollan planes de monitoreo y control para detectar cualquier cambio en el proceso y tomar acciones correctivas de inmediato.

En la tabla 8.5, se presenta la conexión entre el ciclo Six Sigma con el método científico. Algunas fases del ciclo DMAIC se solapan con varias etapas del método científico.

Tabla 8.5. Relación entre el ciclo DMAIC y el método científico

Ciclo DMAIC	Método científico
Definir	Realizar una pregunta
Medir	Realizar alguna investigación de fondo
Analizar	Construir una hipótesis
Mejorar	Probar la hipótesis con un experimento
	Analizar los datos y trazar conclusiones
Controlar	Comunicar resultados

Fuente: elaboración propia con base en Cano *et al.* (2012).

La estructura de trabajo típica de Six Sigma viene dada por los siguientes roles: campeón, cinturón negro maestro, cinturón negro, propietario del proceso y cinturón verde. La terminología proviene de las artes marciales.

- *Campeón*: Persona responsable de un área organizacional. Dirige el programa de mejora dentro de su área y seleccionará los proyectos con las mejores oportunidades de mejora. Coordina su trabajo con otros campeones de la organización.
- *Propietario del proceso*: Persona a cargo de un proyecto concreto. Se reporta directamente al campeón. Tiene la tarea de controlar el proceso del que está a cargo.
- *Cinturón negro maestro*: Debe poseer un conocimiento profundo de técnicas estadísticas, técnicas de planificación y gestión. Jerárquicamente, informa directamente al consejo directivo de la organización sobre la estrategia general del programa de mejora y a los campeones sobre la selección de los proyectos a mejorar. Además, será responsable del programa de capacitación Six Sigma.

- *Cinturón negro*: Responsable del grupo de mejora creado por el campeón dentro de un proyecto determinado. Debe ser lo suficientemente autónomo para liderar un proyecto. También se asegurará de que el ciclo **DMAIC** se siga con disciplina.
- *Cinturón verde*: Tiene conocimiento estadístico elemental que ocasionalmente puede liderar un proyecto de mejora simple. Es el nivel ideal para propietarios de proyectos y gerentes intermedios.

La relación jerárquica de los roles estratégicos de Six Sigma se observa en la figura 8.16. Algunos beneficios que se obtienen al lograr implementar Six Sigma en procesos de software son acelerar el desarrollo y la integración de sistemas, mejorar la calidad de las pruebas, centrarse en los requisitos del cliente y garantizar la entrega de productos de alta calidad (Antony y Banuelas, 2002; Mills y Linger, 2002).

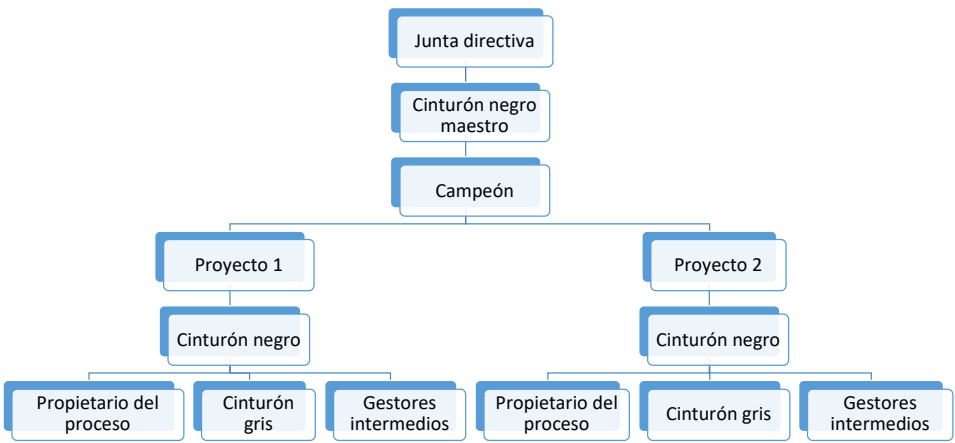


Figura 8.16. Roles estratégicos de Six Sigma.

Fuente: elaboración propia

8.3.7. Modelo BSC

El cuadro de mando integral (**CMI**), o Balance Score Card (**BSC**), es una de las herramientas más utilizadas en el proceso de planificación estratégica y ha sido empleada con éxito en proyectos de desarrollo de software. El modelo fue propuesto por Kaplan y Norton (1992) y tiene como objetivo convertir la estrategia de una empresa en acción y resultados a través de la alineación

de los objetivos de negocio desde las perspectivas financiera, clientes, procesos internos y aprendizaje y desarrollo. Las cuatro perspectivas del CMI se observan en la figura 8.17 (Kazi *et al.*, 2011).

- ▶ *Perspectiva de aprendizaje y crecimiento:* Incluye aprendizaje continuo de los empleados, capacitación, actitudes culturales corporativas relacionadas con la superación personal y corporativa, facilidad de comunicación entre los trabajadores, así como herramientas tecnológicas para la gestión del conocimiento.
- ▶ *Perspectiva de procesos internos:* Los indicadores basados en esta perspectiva permiten conocer cómo está funcionando el negocio y si los productos y servicios cumplen con los requisitos del cliente.
- ▶ *Perspectiva de cliente:* Desarrolla indicadores de satisfacción del cliente. Los clientes deben analizarse en términos de tipos de clientes y los tipos de procesos para los que se está proporcionando un producto o servicio.
- ▶ *Perspectiva financiera:* Incluye datos de financiamiento oportunos y precisos, evaluación de riesgos, datos de costo-beneficio, flujos de efectivo, crecimiento de ventas, participación de mercado, ingresos operativos y retorno sobre capital, gastos operativos y rotación de activos.

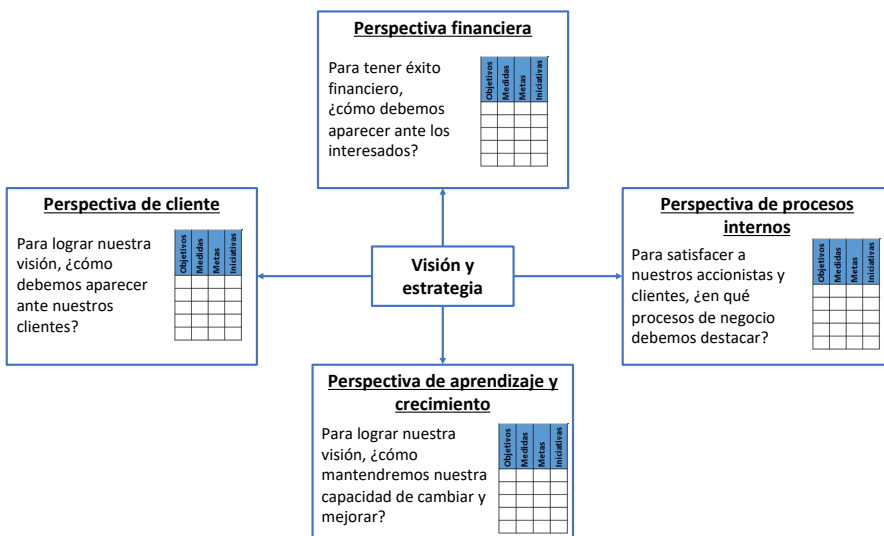


Figura 8.17. Perspectivas del CMI

Fuente: elaboración propia con base en Kazi *et al.* (2011).

El **CMI** se convierte en una herramienta de gestión que ayuda a la toma de decisiones al proporcionar información periódica sobre el nivel de cumplimiento de los objetivos previamente establecidos mediante indicadores. Los indicadores son índices que describen el comportamiento de diversas variables por cuantificación directa de una variable o por comparación entre variables (Kaplan y Norton, 2009).

8.3.8. ISO 33000

El estándar **ISO 33000** reemplazó el modelo **ISO/IEC 15504**, también conocido como determinación de la capacidad de mejora de procesos de software (Software Process Improvement and Capability Determination [**SPICE**]), el cual era un modelo para la mejora y evaluación de procesos de desarrollo y mantenimiento de productos de software y sistemas de información (**ISO/IEC 15504**, 2003).

Tabla 8.6. Familia de estándares ISO 33000

Requisitos generales		Conceptos y terminología (ISO/IEC 33001)	
	Procesos de evaluación	Framework de medición	Modelos de proceso
Principios	Requisitos para realizar la evaluación del proceso (ISO/IEC 33002)	Requisitos para <i>frameworks</i> de medición del proceso (ISO/IEC 33003)	Requisitos para referencia del proceso, evaluación del proceso y modelos de madurez (ISO/IEC 33004)
Estándares de elementos	Proceso de evaluación documentado (ISO/IEC 33030-33039)	<i>Framework</i> s para medir las características de calidad del proceso (ISO/IEC 33020-33029)	Modelos de referencia del proceso (ISO/IEC 33004) Modelos de evaluación del proceso (ISO 33040-33059) Modelos de madurez (ISO 33080-33090)
Guías de aplicación y suplementos	Guía para realizar evaluaciones (ISO 33010) Guía para definir un proceso de evaluación documentado Cuerpo del conocimiento de evaluación del proceso (ISO 33016)	Guía para la mejora del proceso (ISO 33012) Guía para la determinación de la capacidad del proceso (ISO 33013)	Guía para la construcción de modelos de referencia del proceso, modelos de evaluación del proceso y modelos de madurez organizacional para evaluaciones (ISO 33014)

Fuente: elaboración propia con base en Davis y Lewis (2018).

La familia **ISO 33000** permite evaluar características de calidad del proceso, como seguridad, eficiencia, efectividad, integridad y sostenibilidad. La familia de estándares se presenta en la tabla 8.6.

A continuación, se describen brevemente los estándares relacionados con la evaluación de procesos (**ISO 33000**, 2020):

- ▶ **ISO/IEC 33001:2015** *Tecnología de la información - Evaluación del proceso - Conceptos y terminología*: Proporciona un glosario de términos relacionados para la evaluación del proceso, además de una introducción general de los conceptos y estándares para la evaluación del proceso de la familia de normas **ISO/IEC 33000**.
- ▶ **ISO/IEC 33002:2015** *Tecnología de la información - Evaluación del proceso - Requisitos para ejecutar la evaluación del proceso*: Establece los requisitos para la evaluación garantizando la coherencia y la repetitividad de los valores obtenidos durante la evaluación del proceso.
- ▶ **ISO/IEC 33003:2015** *Tecnología de la información - Evaluación del proceso - Requisitos para frameworks de medición del proceso*: Proporciona los requisitos que se aplican a los *frameworks* de medición de procesos, los cuales permiten la evaluación de las características de calidad del proceso.
- ▶ **ISO/IEC 33004:2015** *Tecnología de la información - Evaluación del proceso - Requisitos para modelos de referencia del proceso, modelos de evaluación del proceso y modelos de madurez*: Establece los requisitos para la construcción y verificación de modelos de referencia de procesos, modelos de evaluación de procesos y modelos de madurez.
- ▶ **ISO/IEC TR 33014:2013** *Tecnología de la información - Evaluación del proceso - Guía para la mejora del proceso*: Proporciona una guía informativa sobre la evaluación de procesos como parte de un *framework* para realizar la mejora de procesos. Esta se aborda desde los niveles estratégico, táctico y operativo, y tiene tres perspectivas de mejora: proceso, organización y proyecto.
- ▶ **ISO/IEC 33020:2015** *Tecnología de la información - Evaluación del proceso - Framework de medición del proceso para la evaluación de la capacidad del proceso*: Define un *framework* de medición del proceso que respalda la evaluación de la capacidad del proceso de acuerdo con los requisitos establecidos en la norma **ISO/IEC 33003**.
- ▶ **ISO/IEC TS 33030:2017** *Tecnología de la información - Evaluación del proceso - Ejemplo de proceso de evaluación documentado*: Contiene un

ejemplo de un proceso de evaluación y sirve como una guía de las actividades requeridas para este proceso.

El modelo tiene dos dimensiones: dimensión del proceso y dimensión de la capacidad. La dimensión del proceso consiste en procesos y cada proceso se define en términos de su propósito y resultados. Un nivel de capacidad es un conjunto de características de calidad que proporcionan una mejora importante en la capacidad de realizar un proceso. La dimensión de capacidad define cinco niveles de capacidad. Los modelos de madurez organizacional agrupan los procesos del modelo de referencia en conjuntos predefinidos de procesos para definir un camino de mejora incremental organizacional. Cada nivel de madurez proporciona un conjunto de procesos que definen los diferentes comportamientos de la organización (figura 8.18).

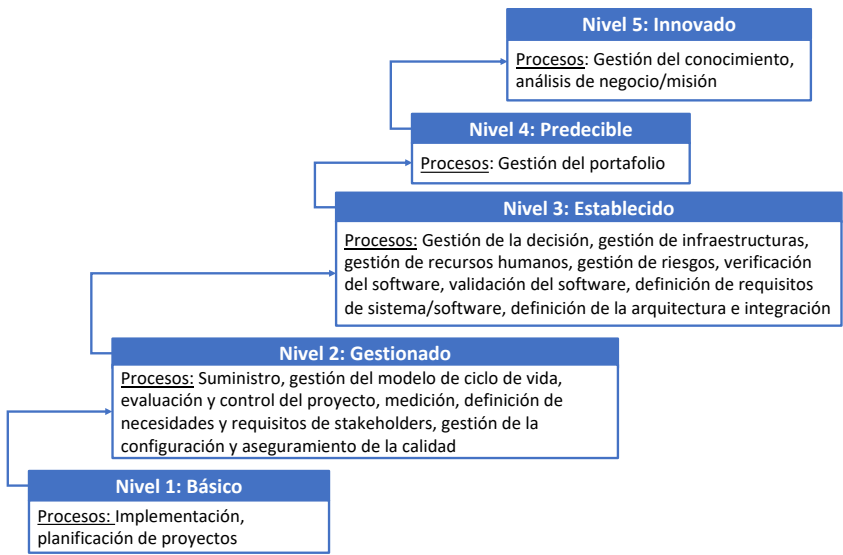


Figura 8.18. Niveles de capacidad de iso 33020

Fuente: elaboración propia.

- *Nivel 0, básico:* Dos procesos son identificados y existen los productos de trabajo de su ejecución.
- *Nivel 1, gestionado:* La organización demuestra la gestión de la ejecución de sus procesos y que los productos de trabajo asociados están debidamente documentados, establecidos, controlados y mantenidos.

- ▶ *Nivel 2, establecido*: La organización demuestra la efectiva definición, mejora, despliegue y aseguramiento de sus procesos.
- ▶ *Nivel 3, predecible*: La organización demuestra un análisis y control cuantitativo de los procesos fundamentales para la consecución de sus objetivos de negocio. Las necesidades de gestión cuantitativa son identificadas, los datos de medición son recogidos y analizados para identificar y corregir las causas de variación asignables.
- ▶ *Nivel 4, innovado*: La organización demuestra la innovación de procesos y su correspondiente implementación para conseguir sus objetivos de negocio.

8.3.9 Modelo TickITplus

El proyecto TickIT inició en 1991 y es un programa de certificación para empresas en las industrias de desarrollo de software respaldado principalmente por las industrias del Reino Unido y Suecia. Desde sus inicios, TickIT ha proporcionado guías para la interpretación de **ISO 9001**. El nuevo esquema TickITplus fue lanzado a principios de 2011 por el Comité Directivo Conjunto de la Industria TickIT (**JTISC**, por sus siglas en inglés). El objetivo principal de este esquema es capitalizar las fortalezas de TickIT mientras se reconocen los cambios en el mundo actual del desarrollo de software. Algunos de los objetivos importantes del modelo son (British Standards Institution [**BSI**], 2011):

- ▶ Adoptar un enfoque completo basado en procesos para la gestión de sistemas de negocio.
- ▶ Introducir conceptos de evaluación de capacidades.
- ▶ Acomodar múltiples requisitos estándar, por ejemplo, **ISO 9001**, **ISO/IEC 20000-1** (gestión de servicios de **TI**) e **ISO/IEC 27001** (gestión de seguridad de la información de **TI**).
- ▶ Fortalecer el compromiso con las mejoras.
- ▶ Permitir que las evaluaciones colaborativas se realicen de manera más formal.

TickIT define un conjunto central de procesos bien definidos que proporcionan cubrimiento completo a un rango de actividades organizacionales. Define cinco niveles de madurez de una organización, los cuales son, en orden ascendente, fundación, bronce, plata, oro y platino. Estos niveles son progresivos y determinan si una organización ha completado ciertos atributos de proceso por medio de evaluaciones de capacidad.

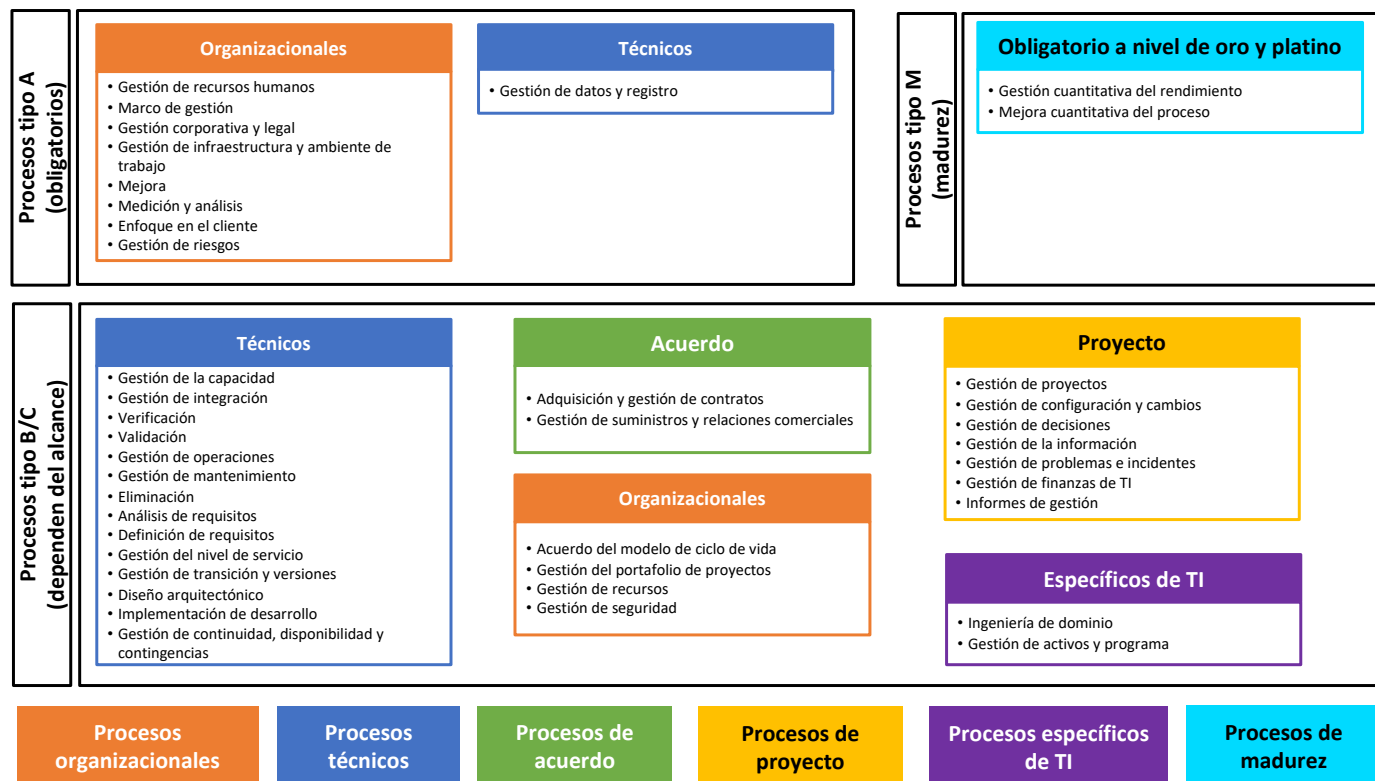


Figura 8.19. Áreas de proceso TickITplus
Fuente: elaboración propia con base en BSI (2011).

El modelo consta de cuarenta procesos que cubren actividades de negocio, ingeniería, funcionales y soporte. Los procesos están contenidos en una base de datos denominada Base Process Library (**BPL**) y se agrupan en una de las seis categorías definidas (**BSI**, 2011), como se observa en la figura 8.19.

Los procesos obligatorios de tipo A son necesarios para cumplir con la norma **ISO 9001** y para todas las evaluaciones certificadas de TickITplus. Los procesos de tipo **B/C** se denominan procesos dependientes del alcance. Si se requieren o no depende de los perfiles de alcance seleccionados, los requisitos y estándares de referencia y el alcance de la certificación. Los procesos de tipo M solo se requieren en los niveles oro y platino.

► 8.4. Metodologías de evaluación del proceso de mejora

Las evaluaciones de los modelos de mejora permiten a una organización comprender la madurez actual del proceso de software, incluidas las fortalezas y debilidades en sus procesos. Se realiza una evaluación inicial al comienzo para permitir que la organización comprenda la madurez actual de su proceso y planifique y priorice las mejoras para el primer ciclo. Luego, se implementan estas y generalmente se realiza una evaluación al final del ciclo para confirmar que se ha avanzado en la iniciativa de mejora (O'Regan, 2017).

Una evaluación es un examen independiente de las prácticas de ingeniería y gestión de software en la organización y se realiza utilizando una metodología de evaluación. La evaluación identificará fortalezas y debilidades en los procesos y cualquier brecha que exista con respecto al modelo de madurez (O'Regan, 2017). Las fases de una evaluación de un proceso de software se presentan en la tabla 8.7.

Tabla 8.7. Fases de una evaluación de proceso de software

Fase	Descripción
Planificación y preparación	Implica identificar los objetivos del patrocinador y los requisitos para la evaluación.
Realización de la evaluación	El equipo de evaluación entrevista a los participantes y examina los datos para juzgar en qué medida se implementa el modelo.
Informar los resultados	Los hallazgos (incluida una presentación y un informe de evaluación) se informan al patrocinador.

Fuente: elaboración propia con base en O'Regan (2017).

Los resultados de la evaluación son valiosos y permitirán al gerente del proyecto planificar y programar el próximo ciclo de mejora. Las evaluaciones permiten que una organización (O'Regan, 2017) comprenda la madurez actual del proceso (incluidas las fortalezas y debilidades), relacione sus fortalezas y debilidades con el modelo de mejora, priorice sus mejoras para el ciclo de mejora y compararse con otras organizaciones.

8.4.1. Modelo SCAMPI

La metodología Standard **CMMI** Appraisal Method for Process Improvement (**SCAMPI**) es la metodología de evaluación utilizada con el **CMMI**. La evaluación identificará fortalezas y debilidades en los procesos y cualquier brecha con respecto a las prácticas **CMMI**. El alcance de la evaluación incluye áreas de proceso, proyectos y unidades organizacionales que se examinarán. Puede estar limitado a las áreas de proceso de un nivel específico (O'Regan, 2017).

La evaluación identificará cualquier brecha existente con respecto a la implementación de las prácticas **CMMI** para cada área de proceso. El equipo de evaluación llevará a cabo entrevistas, revisará la documentación del proyecto y examinará cómo se implementan las prácticas. Los resultados de la evaluación se utilizan para planificar y priorizar el próximo ciclo de mejora. **SCAMPI** permite a una empresa desarrolladora de software (**SCAMPI** Upgrade Team, 2011):

- Obtener una idea de la capacidad de una organización al identificar fortalezas y debilidades de sus procesos actuales en relación con los modelos de referencia de evaluación.
- Priorizar los planes de mejora.
- Centrarse en las mejoras que son más beneficiosas para la organización dado su nivel actual de madurez organizacional o capacidad del proceso.
- Derivar calificaciones del nivel de capacidad, así como una calificación de nivel de madurez.
- Identificar riesgos relativos a las determinaciones de capacidad/madurez.

Existen tres tipos de evaluación **SCAMPI**: clases A, B y C. Estas varían en formalidad, costo, esfuerzo y plazos involucrados, calificación de los procesos e informe de resultados. En la tabla 8.8, se presentan diferencias de requisitos para los tres tipos de clases de **SCAMPI**.

Tabla 8.8. Diferentes tipos de clases SCAMPI

Requisitos	Clase A	Clase B	Clase C
Tipo de evidencia objetiva	Documentos y entrevistas	Documentos y entrevistas	Documentos y entrevistas
Clasificaciones generadas	Clasificación requerida de objetivos	No permitidas	No permitidas
Cubrimiento de unidad organizacional	Requerido	No requerido	No requerido
Tamaño mínimo de equipo	4	2	1
Requisitos del líder de equipo de evaluación	Evaluador principal	Personal entrenado y experimentado	Personal entrenado y experimentado

Fuente: elaboración propia con base en O'Regan (2017).

Todas las clases de **SCAMPI** deben ser supervisadas por agentes autorizados del **SEI**, incluso las clases B y C, para garantizar interpretaciones correctas y autorizadas.

Para realizar una evaluación **SCAMPI**, se deben recolectar indicadores de implementación de las prácticas, los cuales son evidencia de la implementación de una práctica. La verificación consiste en revisar que se encuentra “evidencia objetiva” para determinar si una práctica está implementada. La evidencia objetiva consiste en los siguientes elementos (**SCAMPI** Upgrade Team, 2011):

- ▶ *Artefactos directos*: Son salidas tangibles que resultan de la aplicación directa de las prácticas tanto específicas como genéricas. Por ejemplo, productos de trabajo, documentos, etc.
- ▶ *Artefactos indirectos*: Artefactos que son consecuencia de la aplicación de una práctica (específica o genérica), pero que no son el propósito para el cual se realiza la práctica. Por ejemplo, revisión de resultados, reportes de estado, métricas, etc.
- ▶ *Afirmaciones*: Expresiones orales o escritas expresadas por los responsables de ejecutar las prácticas que confirman la implementación de una práctica específica o genérica. Por ejemplo, entrevistas, videoconferencias, cuestionarios, etc.

Los principales participantes de una evaluación **SCAMPI** son (**SCAMPI** Upgrade Team, 2011):

- ▶ *Patrocinador*: Encargado de liderar el proyecto de mejora. Generalmente, es un directivo de la organización.

- *Jefe del equipo de evaluación*: Responsable de realizar la evaluación. Debe ser un *lead appraiser* y pertenecer a una empresa *partner* del SEI para realizar la evaluación.
- *Coordinador de la organización*: Encargado de apoyar la realización de la auditoría proporcionando información necesaria al evaluador.
- *Miembros del equipo de evaluación*: Personal de la organización a auditar que forma parte del equipo de evaluación contando con la experiencia y el entrenamiento suficiente.
- *Participantes seleccionados*: Encargados de proporcionar la información necesaria. Generalmente, son jefes de los proyectos evaluados.

En la figura 8.20, se presenta un ejemplo de una evaluación **SCAMPI** Clase A **CMMI** nivel 5 (cubre las 22 áreas de proceso). Cada columna representa un área de proceso **CMMI** y cada fila una práctica específica o genérica. Los colores se emplean para indicar en qué medida se han implementado las prácticas específicas o genéricas. El alcance de la implementación puede ser completamente satisfecho, en gran parte satisfecho, parcialmente satisfecho, no satisfecho y no evaluado. Las fases de una evaluación **SCAMPI** se detallan en la tabla 8.9.

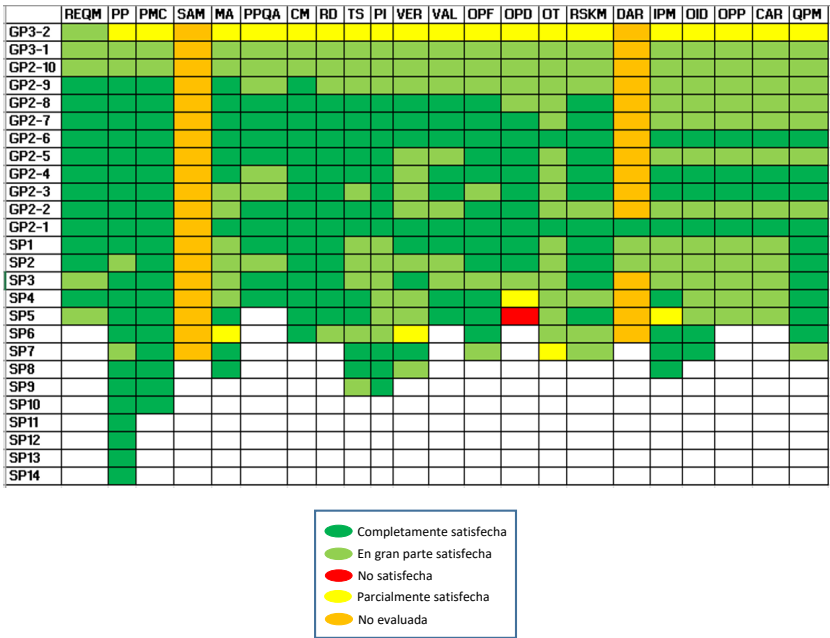


Figura 8.20. Evaluación de las prácticas de las áreas de proceso CMMI

Fuente: elaboración propia con base en O'Regan (2017).

Tabla 8.9. Fases de la evaluación SCAMPI

Fase	Proceso	Descripción
Planificación y requisitos para la evaluación	Analizar requisitos	El líder de la evaluación determinará las metas y los objetivos de la evaluación en discusiones con el evaluador.
	Desarrollar plan de evaluación	El plan de evaluación es preparado por el líder de evaluación y describe el alcance de la evaluación y cómo se llevará a cabo.
	Seleccionar y preparar equipo	El líder determinará el tamaño y la composición del equipo, y seleccionará a los miembros calificados del equipo.
	Obtener y analizar evidencia inicial	La organización proporcionará información inicial al líder de la evaluación antes de la evaluación para mostrar cómo ha implementado el modelo CMMI.
	Prepararse para realizar una evaluación	El líder de la evaluación realizará una revisión de preparación de la evaluación para juzgar si la organización está lista para la evaluación.
Realización de la evaluación	Preparar participantes	El coordinador orientará a los participantes de la evaluación para asegurarse de que entiendan su propósito y sus roles.
	Examinar la evidencia objetiva	El equipo recopila información sobre las prácticas genéricas y específicas de CMMI.
	Documentar evidencia objetiva	Se refiere a documentar la implementación de las prácticas y las fortalezas y debilidades identificadas.
	Verificar evidencia objetiva	Se refiere a verificar la implementación de las prácticas de la organización para cada instancia de proyecto.
	Validar hallazgos preliminares	Validar la comprensión del equipo de evaluación de los procesos implementados ante los miembros de la organización.
	Generar resultados de evaluación	El equipo de evaluación evaluará la satisfacción de objetivos para determinar el nivel de madurez general.
Informe de los resultados	Entregar los resultados de la evaluación	Se presentan las fortalezas y debilidades de los procesos y las calificaciones de las áreas del proceso al <i>partner</i> y a los participantes.
	Archivar resultados de la evaluación	Los datos y registros de la evaluación se conservarán y la información confidencial se eliminará de forma segura.

Fuente: elaboración propia con base en O'Regan (2017).

► 8.5. Mejora de procesos ágiles

Las principales críticas a las iniciativas tradicionales de **SPI** son la necesidad de una gran burocracia y la restricción de la innovación. La extensa documentación y el monitoreo cuantitativo del proceso de desarrollo del producto desempeñan un papel central en enfoques **SPI** (Salo, 2006). La mejora continua para encontrar el proceso de software más adecuado puede no funcionar bien en entornos susceptibles a cambios. En muchos casos, cada organización de software individual debe desarrollar sus propias prácticas de mejora continua y establecer una infraestructura de mejora apropiada (Birk y Rombach, 2001). Un enfoque ágil para **SPI** sería flexible y receptivo de acuerdo con las necesidades locales, alentaría la innovación, crearía proyectos alrededor de aquellos que estuvieran motivados, alentaría a equipos competentes autoorganizados y promovería el desarrollo sostenible de los procesos (Allison, 2005).

Algunos enfoques son métodos, mientras otros son más modelos. Sin embargo, todos los enfoques ágiles tienen un mecanismo inherente para mejorar sus resultados (Poth *et al.*, 2017).

Es notable que muchos enfoques ágiles tienen como concepto central para **SPI** el desarrollo iterativo e incremental de productos con retrospectivas más o menos formales. Sin embargo, el enfoque **SPI** en las organizaciones ágiles debe centrarse en equipos funcionales, que difieren del enfoque **SPI** tradicional en el que se mejoran los procedimientos o procesos organizacionales establecidos. El enfoque de la mejora no está en los procesos de la empresa, sino en el comportamiento individual dentro de los equipos individuales.

La madurez en los procesos ágiles se toma principalmente en principios como (Fontana *et al.*, 2018):

- *Adopción de prácticas*: La madurez se gana cuando el equipo adopta nuevas prácticas.
- *Mejora continua*: Se toma el concepto de madurez definido por el modelo **CMMI-DEV**.
- *Agilidad sostenida*: Los equipos maduros tienen procesos limpios y se mantienen ágiles.
- *Rendimiento del proyecto*: La madurez como una forma de obtener resultados en los proyectos.
- *Equipos altamente productivos*: Se posiciona a los equipos como un papel central para la madurez en el desarrollo ágil de software.

Las prácticas ágiles que se consideran maduras en los modelos de madurez actuales para el desarrollo ágil de software se agrupan en diferentes temas. Esto significa que la madurez en el desarrollo de software ágil se describe principalmente por prácticas relacionadas con el software en sí. Los temas básicos (prácticas y resultados) identificados para promover la mejora de procesos son (Fontana *et al.*, 2018).

- ▶ *Software*: Temas relacionados con el software que se está desarrollando.
 - *Prácticas*: Pruebas concurrentes, compilaciones automatizadas, pruebas automatizadas, codificación eficiente, herramientas integradas, compilaciones diarias, implementación automática, desarrollo basado en pruebas, programación por pares, integración continua y revisión de código.
 - *Resultados*: Pruebas ágiles, codificación ágil, software de alto nivel, código fuente de alto nivel y técnicas ágiles de ingeniería.
- ▶ *Personas*: Temas relacionados con las personas involucradas en el desarrollo del software.
 - *Prácticas*: El equipo conduce a la mejora continua, abraza el cambio, interactúa cara a cara, tiene responsabilidad compartida y trabajo colaborativo.
 - *Resultados*: Tener personas experimentadas, un equipo asertivo, productividad mejorada, cultura de aprendizaje, cultura de mejora y liderazgo ágil.
- ▶ *Procesos*: Temas relacionados con la actuación en los procesos de trabajo.
 - *Prácticas*: Eliminación de residuos (defectos), mejora continua, recolección de medidas y estimaciones de manera ágil.
 - *Resultados*: Procesos ligeros, comprender su situación y mantener un ritmo sostenible.
- ▶ *Cliente*: Temas relacionados con el vínculo con el cliente.
 - *Práctica*: Cliente en el sitio.
 - *Resultados*: Satisfacción del cliente y confianza.
- ▶ *Organización*: Temas relacionados con la estructura organizacional y el soporte.
 - *Prácticas*: Configuración física ágil y prioridad ágil en la organización.
 - *Resultados*: Ninguno.

- *Requisitos*: Temas relacionados con la definición de los requisitos del software.
 - *Prácticas*: Uso de metáforas.
 - *Resultados*: Descubrimiento de requisitos.

Se han identificado cuatro formas de realizar la evaluación en modelos de madurez ágil (Fontana *et al.*, 2018):

- *Evaluación de indicadores*: La mayoría de los modelos de madurez ágil presentan este tipo de evaluación: evaluación de indicadores como no logrado (NL), parcialmente logrado (PL), logrado (L) o totalmente logrado (TL).
- *Basado en CMMI-DEV*: Los modelos proponen áreas de proceso que deben evaluarse en función de sus evidencias.
- *Adopción de prácticas*: Los modelos proponen que la evaluación debe evaluar las prácticas adoptadas por el equipo.
- *Evaluación del logro de resultados/objetivos*: Los modelos presentan elementos tales como metas o resultados que deben estar presentes para evaluar la madurez.

En el **SPI** tradicional, considerando un enfoque de arriba hacia abajo o de abajo hacia arriba, el objetivo es elaborar un proceso repetible que podría mejorarse con las lecciones aprendidas. El proceso **SPI** ágil debe estar listo para los cambios y sus mejoras significan mejorar la capacidad del proceso para realizar los cambios (Santana *et al.*, 2015).

► 8.6. Herramientas de mejora del proceso

Los trabajos pioneros en el área de herramientas y técnicas para la calidad fueron realizados por Kaoru Ishikawa, profesor de ingeniería en la Universidad de Tokio y padre de los círculos de calidad.

Las herramientas básicas de calidad de Ishikawa son hoja de verificación, diagrama de Pareto, histograma, diagrama de dispersión, diagrama de ejecución, diagrama de causa y efecto y gráficos de control (estos últimos estudiados en el capítulo siete). Según su usabilidad, estas herramientas también se clasifican en herramientas de análisis de causa (diagrama de Pareto, diagrama de dispersión y diagrama de causa y efecto) y herramientas de análisis y recopilación de datos (hoja de verificación, histograma, diagrama de dispersión y gráfico de control) (American Society for Quality [ASQ], 2020).

8.6.1. Hoja de verificación

Las hojas de verificación son útiles para la recopilación de datos (durante la fase medir del ciclo **DMAIC**). El resumen orientado al tiempo es particularmente valioso en la búsqueda de tendencias u otros patrones significativos (Montgomery, 2012).

Las hojas de verificación se utilizan cuando (Tague, 2005):

- ▶ Los datos pueden ser observados y recopilados repetidamente.
- ▶ Al recopilar datos sobre la frecuencia o patrones de eventos, problemas, defectos, ubicación de defectos, causas de defectos o problemas similares.
- ▶ Al recopilar datos de un proceso de producción.

Al diseñar una hoja de verificación, es importante especificar claramente el tipo de datos que se recopilará, las partes, la fecha, el analista y cualquier otra información útil para diagnosticar la causa de un bajo rendimiento. En la figura 8.21, se presenta un ejemplo de hoja de verificación que registra los diferentes tipos de defectos detectados en un producto de software en una semana de operación.

Location: _____
Data Collection Dates: _____

Tipo de defecto	Días							Total
	Domingo	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	
Defecto 1	3	1	0	2	0	0	0	6
Defecto 2	0	0	4	4	1	1	1	11
Defecto 3	4	0	0	0	1	0	0	5
Defecto 4	2	0	0	0	0	0	0	2
Defecto 5	0	0	0	0	0	0	1	1
Defecto 6	0	1	0	1	0	0	0	2
Defecto 7	1	0	0	0	5	5	6	17
Defecto 8	1	0	0	0	0	1	1	3
Defecto 9	0	0	0	0	0	0	1	1
Defecto 10	0	5	0	1	2	3	4	15
TOTAL	11	7	4	8	9	10	14	63

Figura 8.21. Ejemplo de hoja de verificación

Fuente: elaboración propia.

8.6.2. Histogramas

Un histograma es una distribución que muestra con qué frecuencia se produce un valor diferente en un conjunto de datos. Son útiles como herramienta de recopilación y análisis de datos. Se utilizan cuando (Tague, 2005):

- Los datos son numéricos.
- Se desea identificar la forma de la distribución de datos, especialmente para determinar si la salida de un proceso se distribuye normalmente o sigue otro tipo de distribución.
- Se desea analizar si un proceso puede cumplir con los requisitos del cliente.
- Se desea ver si se ha producido un cambio de proceso de un periodo a otro.
- Se desea determinar si las salidas de dos o más procesos son diferentes.
- Se desea comunicar la distribución de datos de manera rápida y fácil a otros.

En la figura 8.22, se presenta un ejemplo de histograma aplicado a los datos de la hoja de verificación presentada en la figura 8.21. Viernes, sábado y domingo fueron los días de la semana con mayor número de defectos detectados.

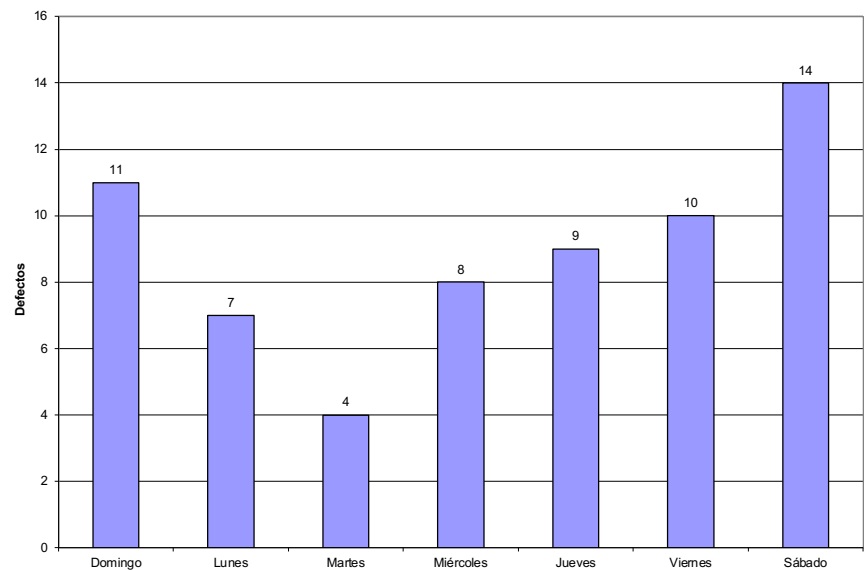


Figura 8.22. Ejemplo de histograma
Fuente: elaboración propia.

8.6.3. Diagrama de Pareto

Un diagrama de Pareto es una distribución de frecuencia de datos de atributos organizados por categoría (Montgomery, 2012). Se utilizan en las fases de medición y análisis de **DMAIC**. Las longitudes de las barras representan la frecuencia o el costo (tiempo o dinero) y se organizan con las barras más largas a la izquierda y las más cortas a la derecha. De esta manera, el cuadro representa visualmente qué situaciones son más significativas.

En la figura 8.23, se presenta un ejemplo de diagrama de Pareto, con los datos de defectos de la hoja de verificación de la figura 8.21. Al trazar la frecuencia total de ocurrencia de cada tipo de defecto contra los diversos tipos de defectos, se concluye que los defectos 7 y 10 son los más frecuentes en el producto de software. Se debe considerar que el diagrama de Pareto no identifica automáticamente los más importantes, sino solo los más frecuentes.

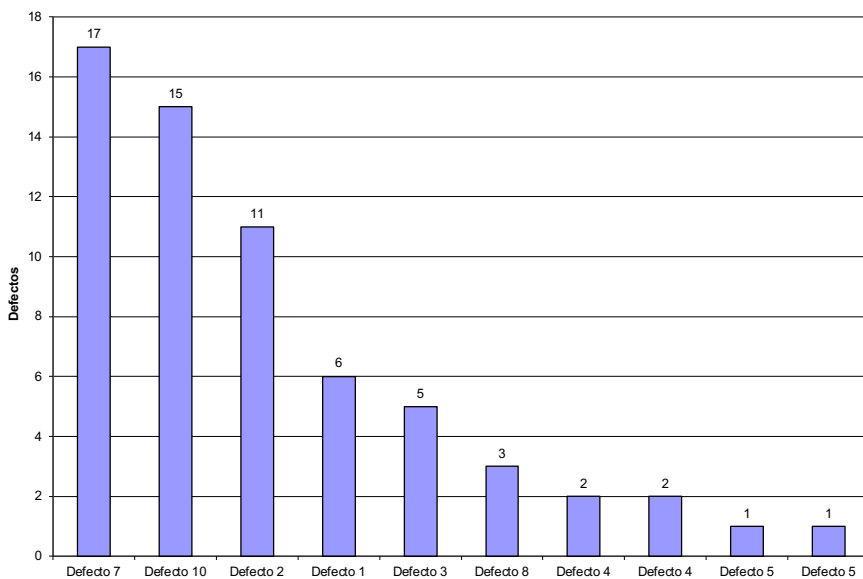


Figura 8.23. Ejemplo de diagrama de Pareto

Fuente: elaboración propia.

Los diagramas de Pareto se utilizan cuando (Tague, 2005):

- ▶ Al analizar datos sobre la frecuencia de problemas o causas en un proceso.

- Cuando hay muchos problemas o causas y se desea centrarse en los más importantes.
- Al analizar causas amplias mirando sus componentes específicos.

8.6.4. Diagrama de dispersión

El diagrama de dispersión presenta pares de datos numéricos, con una variable en cada eje, para buscar una relación entre ellos. El diagrama de dispersión identifica una posible relación entre dos variables. La forma del diagrama a menudo indica qué tipo de relación puede existir entre las dos variables. Si las variables están correlacionadas, los puntos caerán a lo largo de una recta o curva. Cuanto mejor sea la correlación, más cercanos estarán los puntos a la recta o curva. Son útiles en el modelado de regresión, una técnica utilizada en la fase de análisis de **DMAIC** (Montgomery, 2012). En la figura 8.24, se presenta un ejemplo de diagrama de dispersión que relaciona los elementos de una base de datos con los tamaños dados en **LOC** junto con el tiempo de desarrollo real en horas.

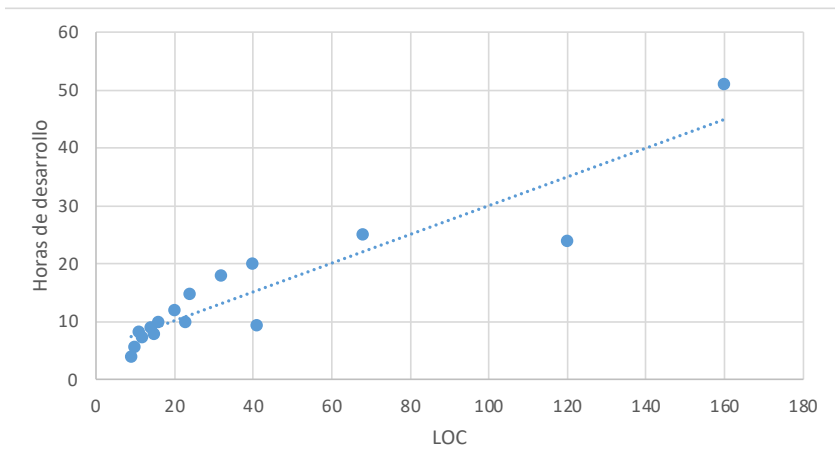


Figura 8.24. Ejemplo de diagrama de dispersión

Fuente: elaboración propia.

Un diagrama de dispersión se puede utilizar para las siguientes situaciones (Tague, 2005):

- Cuando haya parejas de datos numéricos.

- ▶ Cuando una variable dependiente puede tener múltiples valores para cada valor de su variable independiente.
- ▶ Al intentar determinar si dos variables están relacionadas; por ejemplo, al tratar de identificar las posibles causas de los problemas y al determinar si dos efectos que parecen estar relacionados, ambos ocurren por la misma causa.

8.6.5. Diagrama de causa y efecto

También denominados diagrama de Ishikawa o diagrama de espina de pescado, ya que el diseño del diagrama se parece mucho al esqueleto de un pez. Identifica muchas causas posibles para un efecto o problema. Se puede usar para estructurar una sesión de lluvia de ideas. Inmediatamente, clasifica las ideas en categorías útiles. Este diagrama se utiliza cuando se requiere identificar posibles causas de un problema o cuando el pensamiento de un equipo tiende a caer en la rutina. El diagrama es muy útil para analizar y mejorar los pasos de **DMAIC**. Una vez que se ha identificado y aislado un defecto, error o problema para su posterior estudio, se deben analizar las posibles causas de este efecto indeseable (Montgomery, 2012).

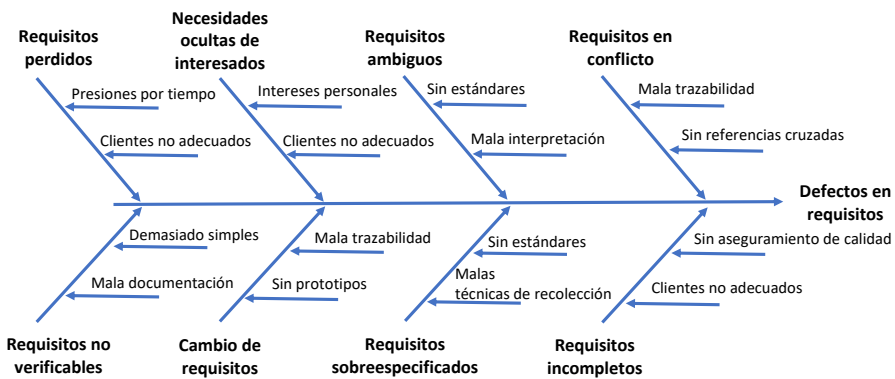


Figura 8.25. Ejemplo de histograma

Fuente: elaboración propia con base en Hauser (2008).

Los diagramas de espina de pescado generalmente se trabajan de derecha a izquierda, con cada “espina” grande del pez ramificándose para incluir huesos más pequeños que contienen más detalles. Hay cuatro pasos para utilizar esta herramienta (Liliana, 2016):

1. Identificar el problema.
2. Resolver los principales factores involucrados.
3. Identificar posibles causas.
4. Analizar el diagrama.

En la figura 8.25, se presenta un ejemplo de diagrama de causa y efecto para analizar los defectos detectados en requisitos de un producto de software.

8.6.6. Diagrama de flujo

Un diagrama de flujo muestra las conexiones entre las diferentes etapas de un proceso o partes de un sistema. Consiste en un conjunto de símbolos y líneas de conexión que muestra la progresión gradual a través de un procedimiento, un proceso o un sistema. Los diagramas de flujo ilustran los pasos de un proceso en orden secuencial y son herramientas comunes del análisis de procesos. Los elementos que pueden incluirse en un diagrama de flujo son una secuencia de acciones, materiales o servicios que entran o salen del proceso, decisiones que deben tomarse, personas que se involucran, tiempo involucrado en cada paso o mediciones del proceso (Bekaroo y Warren, 2016). En la figura 8.26, se presenta un ejemplo de diagrama de flujo para verificar la capacidad de un control de proceso estadístico.

Los diagramas de flujo se utilizan en situaciones como (Tague, 2005):

- Desarrollar la comprensión sobre cómo se realiza un proceso.
- Estudiar un proceso de mejora.
- Comunicar a los demás cómo se realiza un proceso.
- Para documentar un proceso.

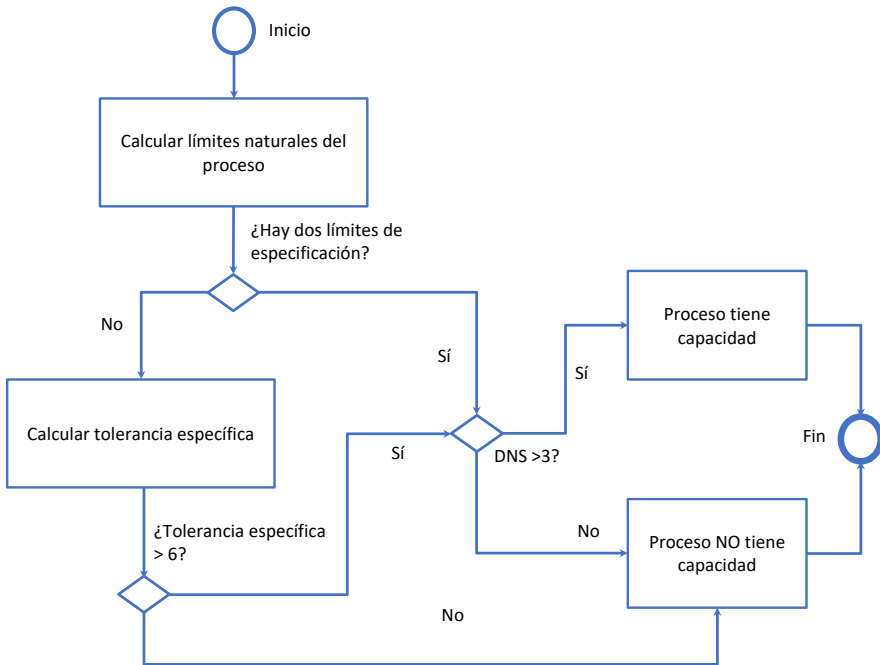


Figura 8.26. Ejemplo de diagrama de flujo

Fuente: elaboración propia con base en Cangussu *et al.* (2003).

► Desafíos

La implementación de programas **SPI** son todo un reto para las organizaciones de software, ya requieren personal dedicado, conocimiento, enfoques dogmáticos y compromiso en todos los niveles (Wiegiers, 1999).

En primer lugar, se requiere tiempo para reflexionar periódicamente sobre qué errores e inconsistencias se presentan en el proceso actual de desarrollo. Se deben integrar las actividades **SPI** con el proceso actual de una forma rutinaria dedicando parte del tiempo de cada proyecto. A su vez, hay que destinar un porcentaje específico del presupuesto y esfuerzo de desarrollo a actividades **SPI**.

Un segundo obstáculo para la mejora generalizada de procesos es que muchos profesionales de software no están familiarizados con las mejores

prácticas de la industria. El desarrollador de software promedio no dedica mucho tiempo a leer literatura especializada para conocer los enfoques de desarrollo de software más conocidos. Los desarrolladores se centran en comprar literatura técnica sobre lenguajes de programación, pero no consultan nada sobre procesos o temas de calidad (Iqbal *et al.*, 2020).

Otro desafío para la mejora efectiva de procesos es tener una mentalidad de lista de verificación, una implementación rígida y dogmática de las políticas **SPI** o de marcos de mejora de procesos específicos. La mentalidad de la lista de verificación se enfoca en el objetivo espurio de obtener algún tipo de certificación del proceso, no en el objetivo real de mejorar los resultados de los proyectos. El resultado final de la mejora de procesos debe ser que los miembros del equipo están trabajando de una manera nueva que obtiene mejores resultados. El cambio de proceso es realmente un cambio de cultura organizacional en que toda la organización se involucra en cambiar su cultura, no es solo la implementación de nuevas prácticas técnicas (Beecham *et al.*, 2003).

Por último, las organizaciones afirman tener las mejores intenciones, pero puede ocurrir que el personal en todos los niveles carezca de un verdadero compromiso con las iniciativas **SPI**. Realizan una evaluación del proceso, pero no cumplen con los cambios reales. Se dedican recursos insuficientes, pero no se definen planes de mejora, ni hojas de ruta ni se implementan nuevos procesos (Wiegers, 1999).

► Ejemplos

Para presentar un ejemplo de implementación de **CMMI**, se describirá un breve ejemplo de la implementación de prácticas del área de proceso de gestión de la configuración (nivel 1 **CMMI**) de una empresa desarrolladora de software (Gupta y Rao, 2011).

Se describirán algunos aspectos desarrollados para lograr el primer objetivo del área de la gestión de configuración: establecer una línea base. La línea de base es un objeto que normalmente representa una configuración estable de un componente. Este primer objetivo está conformado por las siguientes prácticas:

1. Identificar elementos de configuración.

La identificación de la configuración es la selección, creación y especificación de elementos como:

- ▶ Productos que se entregan al cliente.
- ▶ Productos de trabajo internos designados.
- ▶ Productos adquiridos.
- ▶ Herramientas y otros activos del entorno de trabajo del proyecto.

El enfoque de implementación de esta práctica permitió que, cuando un miembro del equipo del proyecto crea un elemento de la documentación del proyecto, se requiere su distribución a otros miembros de este. Por tanto, debe enviarlo al administrador de configuración para que puedan decidir si constituye o no un elemento de configuración. Si es un elemento de configuración, se puede cargar a través de una herramienta de gestión de configuración en su base de datos. Un elemento de configuración puede tener varias versiones. Inicialmente, el administrador de configuración cargará la primera versión de la línea de base aprobada. El elemento de configuración puede modificarse a medida que se prueba y se corrigen los errores, o cuando se solicitan cambios.

1.1. Subpráctica 1.1.1: Seleccionar los elementos de configuración y los productos de trabajo que los componen con base en criterios documentados.

El enfoque de implementación de esta subpráctica está basado en mantener los siguientes artefactos en subcarpetas pertenecientes a la carpeta principal del proyecto:

- ▶ *Entregables:* En esta carpeta, se colocan aquellos documentos que se envían al cliente después de la ejecución de la prueba.
 - Plan de prueba y conjuntos de prueba.
 - Resumen de pruebas.
 - Matriz de trazabilidad de requerimientos.
 - Lecciones aprendidas.
- ▶ *Recibibles:* En esta carpeta, se colocan aquellos documentos que se reciben del cliente antes de la ejecución de la prueba.
 - Requisitos del cliente, especificación de requerimientos software, diseño de alto nivel.
 - Documentos de cambio de alcance.

1.2. Subpráctica 1.1.2: Asignar identificadores únicos a los elementos de configuración.

El enfoque de implementación de esta subpráctica se basó en que los elementos de gestión de configuración deben identificarse de manera única para que puedan distinguirse de los demás elementos.

La herramienta de gestión de configuración tiene la funcionalidad de recuperar los documentos con base en sus etiquetas, lo que ayuda a identificar la línea base. Se utiliza una convención de nomenclatura: número de proyecto_nombre del equipo_nombre del documento.

1.3. Subpráctica 1.1.3: Especificar las características importantes de cada elemento de configuración.

El enfoque de implementación de esta subpráctica se basó en que la herramienta de gestión de la configuración especifica los atributos importantes de un elemento de configuración, como quién desprotegió o devolvió un documento y la fecha que indica cuándo el usuario desprotege y devuelve el registro. Las versiones nuevas y antiguas de un documento están disponibles para el usuario.

1.4. Subpráctica 1.1.4: Especificar cuándo se coloca cada elemento de configuración bajo la gestión de configuración.

El enfoque de implementación de esta subpráctica consistió en que la herramienta de gestión de la configuración etiqueta los documentos de tal manera que puedan representar la etapa en la que se creó el documento. Se puede utilizar una convención en la que el identificador consta de tres partes separadas por guiones en el formato xxx-yy-zz (la primera parte es el número de proyecto, la segunda identifica la etapa de creación del elemento y la tercera es un orden ascendente de elementos generados durante la etapa).

1.5. Subpráctica 1.1.5: Identificar el propietario responsable de cada elemento de configuración.

El enfoque de implementación de esta subpráctica se realiza a través de la herramienta de gestión de la configuración en la que se puede representar al propietario que registra el documento o que ha realizado los cambios en este.

► Resumen

SPI desempeña un papel clave para ayudar a las empresas a mejorar su capacidad de ingeniería de software y alcanzar sus objetivos estratégicos, permite a las organizaciones implementar las mejores prácticas y lograr mejores resultados. Las empresas examinan críticamente sus procesos para

determinar en qué medida son adecuados para su propósito identificando cómo se puede mejorar el proceso y cómo se pueden eliminar los defectos.

El ciclo de mejora del proceso inicia con una planificación de las mejoras, continúa con la implementación de estas, luego se realizan programas piloto que permitirán redefinir las mejoras y después desplegarlas en la organización, y el ciclo se repite nuevamente.

Existen numerosos modelos de mejora de procesos de software. Algunos de los más conocidos y utilizados en la industria son el modelo **QIP** que cuenta con los ciclos de aprendizaje corporativo y de proyectos y basados en una fábrica de experiencia que reutiliza el conocimiento; el modelo **IDEAL** como enfoque genérico para la creación de propuestas **SPI**; el modelo **CMMI** para valorar la capacidad/madurez de las áreas de procesos de una organización desarrolladora de software; el modelo **TSP** para mejorar el proceso de desarrollo de software basado en el trabajo en equipo; el modelo **PSP** para mejorar el proceso individual de desarrollo de software; el modelo Six Sigma que implementa un ciclo de mejora basado en el método científico mediante la recolección de datos estadísticos; el modelo **CMI** para la planificación estratégica utilizando las perspectivas financiera, clientes, procesos internos, y aprendizaje y desarrollo, y el modelo TickI-Tplus para cumplir múltiples requisitos estándar de modelos, como **ISO 9001**, **ISO/IEC 20000-1** e **ISO/IEC 27001**.

También existen varias metodologías para la evaluación de los procesos de mejora en las que el más conocido y utilizado es **SCAMPI** para evaluar las organizaciones que han implementado las prácticas **CMMI** y determinar el nivel de madurez de la capacidad de la organización.

Además, se han desarrollado propuestas que adaptan las iniciativas **SPI** a entornos ágiles. La madurez de los procesos considera la adopción de prácticas, la mejora continua, la agilidad sostenida y el rendimiento del proyecto. Su evaluación incorpora aspectos relacionados con la evaluación de indicadores, adopción de prácticas, evaluación del logro de resultados/objetivos, que toma como punto de partida **CMMI-DEV**.

Por último, existen algunas herramientas que se pueden aplicar para ayudar a realizar la mejora del proceso. Generalmente, estas herramientas están basadas en los conceptos de calidad del proceso y su control estadístico. Las herramientas más ampliamente utilizadas son hojas de verificación, diagramas de Pareto, histogramas, diagramas de dispersión, diagramas de ejecución, diagramas de causa y efecto, y gráficos de control.

► Ejercicios

1. ¿Cuáles son los beneficios de la mejora de procesos de software?
2. Describir y comparar dos modelos de mejora de procesos de software.
3. Especificar dos ventajas de utilizar **CMMI** como modelo de mejora de procesos de software.
4. ¿Qué tipo de empresas son idóneas para aplicar el modelo **CMMI**?
5. ¿Cuándo se debe aplicar la representación continua o por etapas de **CMMI**?
6. Describir los niveles de madurez de **CMMI** y las áreas de proceso en cada nivel.
7. Una empresa tiene un problema al hacer seguimiento de las diferentes versiones de su código fuente. Desarrollar un diagrama de causa y efecto para ese problema.
8. ¿Cómo se utiliza Six Sigma para la mejora del proceso de software?
9. ¿Cuáles son los beneficios de utilizar **BSC** para la mejora del proceso de software?
10. Describir varias herramientas para la mejora del proceso como el diagrama de Pareto, histogramas y diagramas de dispersión, y cómo se pueden utilizar para resolver diferentes problemas identificados durante la ejecución de un proyecto de software.
11. Discutir el propósito de las metodologías para la evaluación de la mejora del proceso de software.
12. Comparar las tres clases de evaluación **SCAMPI**.

► Referencias

- Allison, I. (2005). Towards an agile approach to Software Process Improvement: Addressing the changing needs of software products. *Communications of IMA*, 5(1), 67-76. <https://doi.org/10.58729/1941-6687.1256>
- American Society for Quality. (2020). *Quality tools*. <https://asq.org/quality-resources/quality-tools>
- Antony, J. y Banuelas, R. (2002). Key ingredients for the effective implementation of Six Sigma program. *Measuring Business Excellence*, 6(4), 20-27. <https://doi.org/10.1108/13683040210451679>

- Basili, V., Caldiera, G. y Rombach, D. (1994). Experience factory. En *Encyclopedia of software engineering* (vol. 1, pp. 476-496). John Wiley & Sons.
- Beecham, S., Hall, T. y Rainer, A. (2003). Software process improvement problems in twelve software companies: An empirical analysis. *Empirical Software Engineering*, 8, 7-42. <https://doi.org/10.1023/A:1021764731148>
- Bekaroo, G. y Warren, P. (2016). Self-tuning flowcharts: A priority-based approach to optimize diagnostic flowcharts. En *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)* (pp. 279-285). IEEE. <https://doi.org/10.1109/EmergiTech.2016.7737352>
- Birk, A. y Rombach, D. (2001). A practical approach to continuous improvement in software engineering. En M. Wieczorek y D. Meyerhoff (eds.), *Software quality: State of the art in management, testing, and tools* (pp. 34-45). https://doi.org/10.1007/978-3-642-56529-8_3
- Borstler, J., Carrington, D., Hislop, G. W., Lisack, S., Olson, K. y Williams, L. (2002). Teaching PSP: Challenges and lessons learned. *IEEE Software*, 19(5), 42-48. <https://doi.org/10.1109/MS.2002.1032853>
- British Standards Institution. (2011). *Kick start guide TickITplus*. https://www.tickitplus.org/en/standards-and-guidance/guidance.html?file=files/content/tickitplus/TickITplus_-_Kick_Start_Guide_1.pdf&cid=33397
- Bubevski, V. (2010). An application of Six Sigma and simulation in software testing risk assessment. En *2010 Third International Conference on Software Testing, Verification and Validation* (pp. 295-302). IEEE. <https://doi.org/10.1109/ICST.2010.23>
- Cangussu, J. W., DeCarlo, R. A. y Mathur, A. P. (2003). Monitoring the software test process using statistical process control: A logarithmic approach. En *Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 158-167). IEEE. <https://doi.org/10.1145/940071.940093>
- Cano, E. L., Moguerza, J. M. y Redchuk, A. (2012). *Six Sigma with R: Statistical engineering for process improvement*. Springer.
- Chaudhary, M. y Chopra, A. (2017). *CMMI for development: Implementation guide*. Apress. <https://doi.org/10.1007/978-1-4842-2529-5>
- Davis, P. T. y Lewis, B. D. (2018). *Project management capability assessment: Performing ISO 33000-Based capability assessments of project management*. CRC Press.
- Ferreira, M. G. y Wazlawick, R. S. (2011). Complementing the SEI-IDEAL model with deployers' real experiences: The need to address human factors in SPI Initiatives. En *CibSE* (pp. 39-52). https://www.academia.edu/download/32809080/cibse_paper03.pdf
- Fontana, R. M., Albuquerque, R., Luz, R., Moises, A. C., Malucelli, A. y Reinehr, S. (2018). Maturity models for agile software development: What are they? En X. Larrucea, I. Santamaria, R. O'Connor y R. Messnarz (eds.), *Systems, Software and Services Process Improvement. EuroSPI 2018. Communications in Computer and Information Science* (pp. 3-14). Springer. https://doi.org/10.1007/978-3-319-97925-0_1
- Grütter, G. y Ferber, S. (2002). The Personal Software Process in practice: Experience in two cases over five years. En J. Kontio y R. Conradi (eds.), *Software Quality - ECSQ*

2002. **ECSQ** 2002. *Lecture Notes in Computer Science* (pp. 165-174). Springer. https://doi.org/10.1007/3-540-47984-8_20
- Gupta, P. y Rao, D. S. (2011). Best practices to achieve CMMI level 2 configuration management process area through **vss** tool. *International Journal of Computer Technology and Applications*, 2(3), 542-558.
- Harry, M. y Schroeder, R. (2000). *Six Sigma: The breakthrough management strategy revolutionizing the world's top corporations*. Doubleday.
- Hauser, S. (2018). *Analysis of requirement problems regarding their causes and effects for projects with the objective to model qualitative pris-empirical study*. <https://ceur-ws.org/Vol-2075/DS-paper3.pdf>
- Humphrey, W. S. (2001). *Winning with software: An executive strategy*. Pearson Education.
- Humphrey, W. S. (2005). **PSP** (sm): *A self-improvement process for software engineers*. Addison-Wesley.
- Humphrey, W. S. y Over, J. W. (2010). *Leadership, teamwork, and trust: Building a competitive software capability*. Addison-Wesley.
- Iqbal, J., Nasir, M. H. N., Khan, M., Awan, I. y Farid, S. (2020). Software process improvement implementation issues in small and medium enterprises that develop healthcare applications. *Journal of Medical Imaging and Health Informatics*, 10(10), 2393-2403. <https://doi.org/10.1166/jmihi.2020.3187>
- ISO 33000. (2020). **ISO 33000**. <https://www.iso33000.es/>
- ISO/IEC 15504. (2003). International Organization for Standardization and the International Electrotechnical Commission (**ISO/IEC**). **ISO/IEC 15504-2 - Information technology - Process assessment - Part 2: Performing an assessment**.
- Kandt, R. K. (2003). *Ten steps to successful Software Process Improvement*. <https://dataverse.jpl.nasa.gov/api/access/datafile/6189?gbrecs=true>
- Kaplan, R. S. y Norton, D.P. (1992). The balanced scorecard: Measures that drive performance. *Harvard Business Review*, 70(1), 71-79. <https://hbr.org/1992/01/the-balanced-scorecard-measures-that-drive-performance-2>
- Kaplan, R. S. y Norton, D.P. (2009). *El cuadro de mando integral*. Gestión 2000.
- Kazi, L., Radosav, D., Nikolic, M. y Chotaliya, N. (2011). Balanced scorecard framework in software project monitoring. *Journal of Engineering Management and Competitiveness (JEMC)*, 1(1-2), 51-56. <http://www.tfzr.uns.ac.rs/JEMC/files/V1N1-22011-10.pdf>
- Kuhrmann, M., Konopka, C., Nellesmann, P., Diebold, P. y Münch, J. (2015). Software process improvement: Where is the evidence? Initial findings from a systematic mapping study. En *Proceedings of the 2015 International Conference on Software and System Process* (pp. 107-116). <https://doi.org/10.1145/2785592.2785600>
- Kuilsboer, J. P. y Ashrafi, N. (2000). Software process and product improvement: An empirical assessment. *Information and Software Technology*, 42(1), 27-34. [https://doi.org/10.1016/S0950-5849\(99\)00054-3](https://doi.org/10.1016/S0950-5849(99)00054-3)
- Lee, J. C., Hsu, W. C. y Chen, C. Y. (2018). Impact of absorptive capability on Software Process Improvement and firm performance. *Information Technology and Management*, 19, 21-35. <https://doi.org/10.1007/s10799-016-0272-6>

- Liliana, L. (2016). A new model of Ishikawa diagram for quality assessment. En *Iop Conference Series: Materials Science and Engineering*, 161(1), 012099. <https://doi.org/10.1088/1757-899X/161/1/012099>
- McFeeley, B. (1996). *IDEAL: A user's guide for Software Process Improvement*. Software Engineering Institute, Carnegie Mellon University. https://insights.sei.cmu.edu/documents/1622/1996_002_001_16433.pdf
- Mejía, J., Íñiguez, E y Muñoz, M. (2017). Data Analysis for Software Process Improvement: A systematic literature review. En Á. Rocha, A. Correia, H. Adeli, L. Reis y S. Costanzo (eds.), *Recent Advances in Information Systems and Technologies. WorldCIST 2017. Advances in Intelligent Systems and Computing* (pp. 48-59). Springer. https://doi.org/10.1007/978-3-319-56535-4_5
- Mills, H. D. y Linger, R. C. (2002). Cleanroom software engineering: Developing software under statistical quality control. En *Encyclopedia of Software Engineering*. John Wiley & Sons. <https://doi.org/10.1002/0471028959.sof040>
- Montgomery, D. C. (2012). *Statistical quality control*. Wiley Global Education.
- Niazi, M., Mishra, A. y Gill, A. Q. (2018). What do software practitioners really think about Software Process Improvement project success? An exploratory study. *Arabian Journal for Science and Engineering*, 43, 7719-7735. <https://doi.org/10.1007/s13369-018-3140-3>
- O'Regan, G. (2017). *Concise guide to software engineering*. Springer.
- Pernstål, J., Feldt, R., Gorschek, T. y Florén, D. (2019). **FLEX-RCA**: A lean-based method for root cause analysis in Software Process Improvement. *Software Quality Journal*, 27, 389-428. <https://doi.org/10.1007/s11219-018-9408-8>
- Piattini Velthuis, M. G. y Garzás Parra, J. (2007). *Fábricas de software: Experiencias, tecnologías y organización*. **RA-MA**.
- Pillai, A. K. R., Pundir, A. K. y Ganapathy, L. (2012). Implementing integrated Lean Six Sigma for software development: A flexibility framework for managing the continuity. Change dichotomy. *Global Journal of Flexible Systems Management*, 13, 107-116. <https://doi.org/10.1007/s40171-012-0009-2>
- Pomeroy-Huff, M., Mullaney, J., Cannon, R. y Seburn, M. (2008). *The Personal Software Process-SM (PSP-SM) Body of Knowledge, Version 1.0. No. CMU/SEI-2005-SR-003*. Software Engineering Institute, Carnegie Mellon University. <https://apps.dtic.mil/sti/tr/pdf/ADA636411.pdf>
- Poth, A., Sasabe, S. y Mas, A. (2017). Lean and agile Software Process Improvement: An overview and outlook. En J. Stolf, S. Stolf, R. O'Connor y R. Messnarz (eds.), *Systems, Software and Services Process Improvement. EuroSPI 2017. Communications in Computer and Information Science* (pp. 471-485). Springer. https://doi.org/10.1007/978-3-319-64218-5_38
- Pournaghshband, H. y Watson, J. (2017). Should Six Sigma be incorporated into software development & project management? En *2017 International Conference on Computational Science and Computational Intelligence (CSCI)* (pp. 1021-1026). **IEEE**. <https://doi.org/10.1109/CSCI.2017.176>
- Pressman, R. S. (2005). *Software engineering: A practitioner's approach*. Palgrave Macmillan.

- Qumer, A., Henderson-Sellers, B. y McBride, T. (2007). Agile adoption and improvement model. En *Proceedings European and Mediterranean Conference on Information Systems (EMCIS)*. The Information Institute, Brunel University. <https://opus.lib.uts.edu.au/bitstream/10453/6833/1/2006014581.pdf>
- Salo, O. (2006). *Enabling Software Process Improvement in agile software development teams and organisations* [tesis de doctorado, **VTT** Technical Research Centre of Finland]. <https://publications.vtt.fi/pdf/publications/2006/P618.pdf>
- Santana, C., Queiroz, F., Vasconcelos, A. y Gusmão, C. (2015). Software process improvement in agile software development a systematic literature review. En *2015 41st Euromicro Conference on Software Engineering and Advanced Applications* (pp. 325-332). **IEEE**. <https://doi.org/10.1109/SEAA.2015.82>
- SCAMPI** Upgrade Team. (2011). **SCAMPI** - Standard **CMMI** Appraisal Method for Process Improvement (**SCAMPI**) A, Version 1.3: Method Definition Document. Technical Report **CMU/SEI-2011-HB-001**. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. https://insights.sei.cmu.edu/documents/1618/2011_002_001_15311.pdf
- Shin, H., Choi, H. J. y Baik, J. (2007). Jasmine: A PSP supporting tool. En Q. Wang, D. Pfahl y D. M. Raffo (eds.), *Software Process Dynamics and Agility*. **ICSP 2007. Lecture Notes in Computer Science** (pp. 73-83). Springer. https://doi.org/10.1007/978-3-540-72426-1_7
- Software Engineering Institute. (2010). **CMMI** para Desarrollo, Versión 1.3. Software Engineering Institute, Carnegie Mellon University. https://insights.sei.cmu.edu/documents/87/2010_019_001_28782.pdf
- Tague, N. R. (2005). *The quality toolbox*. **ASQ** Quality Press.
- Wieggers, K. E. (1999). Process Improvement that works. *Software Development*, 7(10), 24-30.
- Zahran, S. (1998). *Software process improvement: Practical guidelines for business success*. Addison-Wesley.



► Parte III. Temas avanzados del proceso de software

Esta tercera parte tiene como objetivo fundamental que el lector comprenda algunos temas avanzados que permitirán obtener una visión más completa del proceso de software y estudiar algunas particularidades de este referente a la definición de una infraestructura adecuada para su implementación, su adaptación a pymes y aspectos relacionados con el reúso del proceso.

► **Capítulo 9. Infraestructura del proceso de software**

Para implementar con éxito un enfoque orientado al proceso en una organización desarrolladora de software, se requiere una infraestructura en recursos humanos, tecnológicos y de formación. Para ello, este capítulo presenta conceptos relacionados con el despliegue del proceso, es decir, con su implantación y uso efectivo en la organización, de tal manera que los usuarios del proceso conozcan y utilicen el proceso continuamente. El capítulo también explica los diferentes grupos de recursos humanos y tecnológicos que se deben implementar para llevar con éxito las estrategias de mejora de procesos de software.

► **Capítulo 10. Procesos de software para pymes**

Los enfoques orientados al proceso requieren numerosos recursos para su implementación y evaluación continua. La consecución de estos recursos puede ser onerosa para numerosas organizaciones de tamaño pequeño y mediano. Debido a esto, se han creado y adaptado modelos espe-

cíficos orientados a las particularidades de estas empresas. Este capítulo pretende que el lector comprenda las características de las empresas desarrolladoras de software y cómo pueden utilizar diversas propuestas de estándares, infraestructuras, modelos y técnicas de procesos de software en pymes desarrolladoras de software.

► Capítulo 11. Reúso del proceso de software

Una vez las organizaciones desarrolladoras de software tengan definidos e implementados sus procesos de software, deben tener la capacidad de reutilizarlos en diferentes proyectos. El reúso del proceso de software se convierte en un elemento clave para implementar con éxito el proceso en proyectos futuros. El capítulo explica el concepto de reúso de software y cómo puede ser extendido al dominio del reúso de procesos de software. Además, presenta diferentes tecnologías orientadas al reúso del proceso basadas en líneas de procesos, arquitecturas de procesos y patrones de proceso. También describe cómo adaptar un proceso a las características particulares de un proyecto de software y finaliza con una descripción de técnicas de gestión del conocimiento aplicadas al reúso de procesos de software.

Infraestructura del proceso de software

► Objetivos

Los objetivos de este capítulo están orientados a entender la infraestructura en recursos humanos, tecnológicos, financieros y de formación necesarios para implementar exitosamente un proceso de software. Al finalizar, el lector estará en capacidad de:

- Comprender el concepto de despliegue del proceso de software para implementar nuevos o mejores procesos en las organizaciones.
- Entender la infraestructura organizacional, de gestión y tecnológica para implementar el proceso de software.
- Comprender el proceso de formación para capacitar al personal en los nuevos o mejores procesos de software.
- Entender diferentes métricas que permitan evaluar el retorno de la inversión en la mejora de procesos de software.

En los últimos treinta y cinco años, en la literatura se encuentra un amplio espectro de investigación sobre procesos de software que aún no finaliza sus actividades de producción y publicación. Una de las líneas de investigación más importantes ha sido tratar los procesos como un tipo de software, es decir, “los procesos de software también son software”, propuesta por Osterweil (2011). La idea, aunque inicialmente algo controvertida, ha sido reconocida como una de las más influyentes en la ingeniería de software.

Para que el proceso de software sea correctamente implementado en las organizaciones, se requiere una infraestructura global que abarque aspectos de gestión, personal, tecnológico y financiero. Humphrey (2005) estableció objetivos globales para la investigación de procesos y en particular identificó cuatro áreas críticas: diseño de procesos, utilización de procesos, análisis de procesos y soporte del proceso. Estas áreas críticas deben ser soportadas por la infraestructura tecnológica del proceso para que el definido por la organización se institucionalice formalmente y pueda ser ejecutado plenamente como se diseñó. A su vez, las infraestructuras tecnológicas permitirán su uso y control efectivo en las organizaciones.

La infraestructura necesaria para soportar adecuadamente el proceso debe ser suministrada antes de que comiencen los proyectos piloto para proporcionar circunstancias realistas. Esto incluye proveer una cantidad suficiente de herramientas, sitios de trabajo, recursos de *hardware*, así como recursos educativos, como aulas y personal para servicios de asistencia técnica, entre otros.

► 9.1. Despliegue del proceso de software

Una vez la mejora del proceso ha formulado un nuevo proceso de software o un proceso modificado, se requiere que el proceso sea desplegado en la organización. Sin embargo, la implementación de estos modelos en las organizaciones presenta dificultades que incluyen (Messnarz *et al.*, 2008):

- Los esfuerzos de mejora no están alineados con los objetivos de negocio.
- Falta de liderazgo y compromiso visible con los esfuerzos de mejora.
- El proceso no responde a las necesidades del negocio.
- Los esfuerzos para implementar aspectos técnicos ignoran las estrategias basadas en los aspectos sociales.

De acuerdo con Bayona *et al.* (2013), los elementos de implementación del proceso de software que son factores de éxito determinantes para su correcto despliegue se presentan en la figura 9.1 y se describen a continuación:

- *Organización*: En el área de tecnología e información (TI) se implementarán los nuevos procesos (incluso la estructura organizacional y funcional). Cada organización tiene una visión, misión, objetivos de negocio, valores y cultura organizacional con los cuales el proceso de software debe estar alineado.

- *Biblioteca de activos de proceso (Process Asset Library [PAL])*: Es un repositorio para descripciones de procesos y activos de proceso. Este elemento se describe en detalle en el apartado 9.4.4 de este capítulo.
- *Procesos*: Los procesos que se pretenden implementar y desplegar.
- *Personas*: Recursos humanos con el conocimiento, las habilidades y actitudes requeridas para realizar el proceso. Se debe considerar su capacidad y competencia, el trabajo en equipo, la alineación con la visión organizacional, la formación y comunicación.
- *Metodología de implementación del proceso*: Establece pautas, procedimientos y reglas para implementar los procesos. Incluye actividades de formación y motivación a las personas en los nuevos procesos.
- *Proceso de implementación*: Seleccionar una estrategia de implementación del proceso con elementos como la comunicación, capacitación y evaluación.
- *Modelo de referencia*: Modelo de proceso de software desarrollado y adecuado a las características de la organización que sirve como base para la definición de los procesos.
- *Gestión del cambio*: Se presentan cambios que implementan el nuevo proceso en la organización y estos deben gestionarse para minimizar la resistencia del personal.

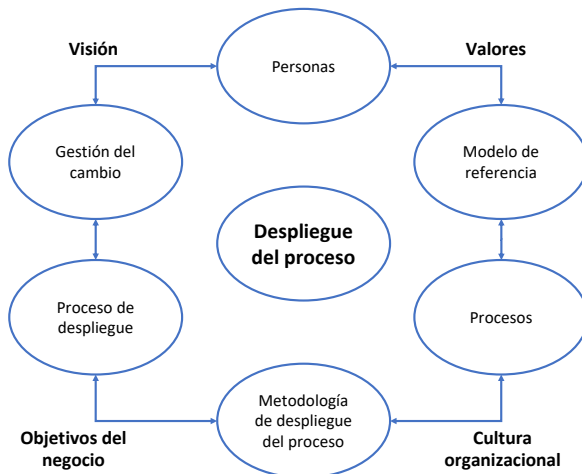


Figura 9.1. Elementos del despliegue del proceso

Fuente: elaboración propia con base en Bayona *et al.* (2013).

Se ha observado que la resistencia al cambio se manifiesta en el comportamiento individual de numerosas maneras, ya sea consciente o inconscientemente (Maurer, 2010). La tabla 9.1 presenta diversas formas de resistencia junto con su manifestación.

Tabla 9.1. Formación de resistencia al cambio

Forma de resistencia	Manifestación
Confusión	Dificultad para darse cuenta de que el cambio va a suceder.
Crítica inmediata	Rechazar el cambio antes de escuchar los detalles.
Negación	Negarse a aceptar que las cosas han cambiado.
Cumplimiento malicioso	Aparente aceptación solo para demostrar una falta de cumplimiento más adelante.
Sabotaje	Tomar medidas para inhibir o eliminar el cambio.
Acuerdo fácil	Aprobar el cambio con poca resistencia, pero sin darse cuenta de lo acordado.
Desviación	Cambiar de tema y esperar que desaparezca el cambio.
Silencio	Ausencia completa de aportes, puede ser la resistencia más difícil de manejar.

Fuente: elaboración propia con base en Nikula *et al.* (2010).

Los cambios en una organización se han descrito gráficamente utilizando la curva clásica del cambio, siguiendo el comportamiento mostrado en la figura 9.2. Una caída en el rendimiento durante el periodo de transición caracteriza la curva de cambio clásico. En un programa de cambio típico, el desempeño en el estado de equilibrio final puede diferir poco del estado de equilibrio inicial, pero un programa de cambio efectivo debería resultar en un desempeño mucho mejor que antes a medida que la organización se instala en su nueva forma de hacer las cosas.

Las capacidades organizacionales se definen por los recursos, luego por procesos y valores y por la cultura. Mientras las capacidades residan en recursos, cambiar las capacidades organizacionales es relativamente fácil; pero, cuando residen en procesos y valores, y especialmente en la cultura, un cambio puede ser muy difícil de implementar (Christensen y Overdorf, 2002).

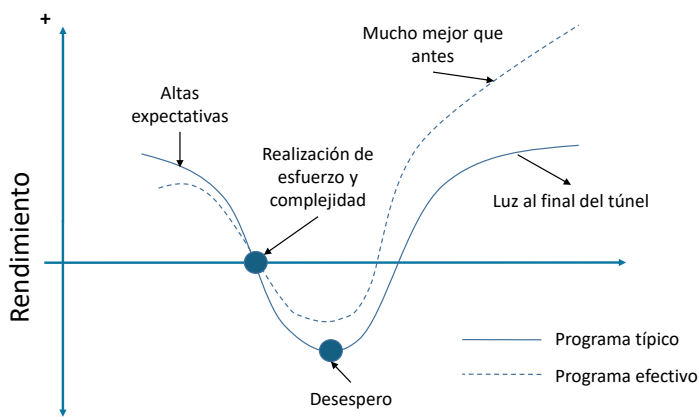


Figura 9.2. Curva clásica del cambio

Fuente: elaboración propia con base en Schneider y Goldwasser (1998).

Henderson y Clark (1990) proponen un marco para clasificar cuatro tipos de cambio tecnológico o innovación en función de los componentes del sistema y sus enlaces (figura 9.3). Cuando un cambio refuerza los conceptos centrales de los componentes, pero deja intactos los enlaces entre ellos, el cambio se considera incremental. Un cambio que anula los conceptos centrales y modifica el enlace entre los componentes se considera un cambio radical. Un cambio que se limita a los componentes, pero que anula los conceptos centrales se considera un cambio modular. Por último, si un cambio se centra en el vínculo entre los conceptos centrales y los componentes, se considera un cambio arquitectónico.

		Conceptos centrales	
		Reforzados	Anulados
Enlaces entre conceptos centrales y componentes	Sin enlaces	Cambio incremental	Cambio modular
	Con enlaces	Cambio arquitectónico	Cambio radical

Figura 9.3. Tipos de cambio según conceptos centrales y enlaces entre ellos

Fuente: elaboración propia con base en Henderson y Clark (1990).

► 9.2. Conceptos sobre infraestructura del proceso de software

La infraestructura del proceso de software se refiere a una aplicación sistemática de procesos de software y modelos de ciclo de vida de software en una organización (Bourque y Fairley, 2014). La infraestructura del proceso permite que este se institucionalice en la organización. La institucionalización del proceso especifica que una infraestructura de proceso efectiva debe cubrir aspectos importantes como (Bourque y Fairley, 2014; Zahran, 1998):

- Definiciones del proceso.
- Capacitación y apoyo al proceso.
- Monitoreo y aplicación del proceso.
- Políticas para interpretar y aplicar los procesos.
- Descripciones de los procedimientos que se utilizarán para implementar los procesos.
- Financiamiento y herramientas.
- Personal que establece y mantiene la infraestructura de proceso de software.

Los procesos de software se implementan a través del desarrollo de proyectos específicos de desarrollo de software. La infraestructura del proceso de software varía según el tamaño y la complejidad de la organización y de los proyectos de la organización. Las organizaciones y los proyectos pequeños y simples tienen a su vez necesidades de infraestructura pequeñas y simples. Las organizaciones y proyectos grandes y complejos tienen igualmente infraestructuras de proceso de software más grandes y complejas (Bourque y Fairley, 2014).

Algunas de las principales tendencias actuales que afectan la infraestructura tecnológica de los procesos de software tanto para el desarrollo como para las operaciones son (Kneuper, 2018):

- Digitalización y sistemas ciberfísicos.
- Computación en la nube.
- Aumentar la orientación al servicio a través de la Service-Oriented Architecture (SOA) y microservicios.
- Una cooperación más estrecha entre desarrollo y operaciones, por ejemplo, despliegue continuo y Development Operations (DevOps).

- Creciente relevancia de la seguridad.
- Globalización con diferentes culturas dispersas geográficamente trabajando juntas, a veces en el mismo proyecto.
- Creciente relevancia de los productos estándar en que la configuración de componentes y servicios estándar se vuelve más importante que el desarrollo genuino.
- Creciente importancia del desarrollo de pequeñas aplicaciones móviles.

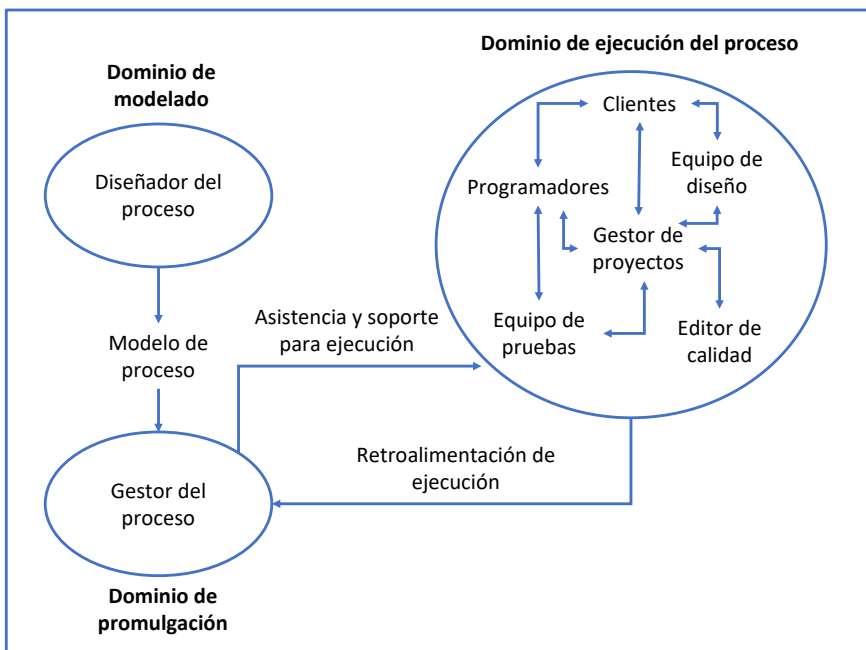


Figura 9.4. Dominios del proceso de software

Fuente: elaboración propia con base en Dowson (1993).

El marco conceptual y terminológico de la infraestructura se muestra en la figura 9.4. Este presenta tres dominios del proceso de software (Dowson, 1993):

- *Dominio de modelado del proceso:* Abarca la definición y el mantenimiento de modelos de proceso de software en un lenguaje de modelado de procesos.

- *Dominio de promulgación de procesos*: Los dominios de modelo y ejecución están unidos entre sí a través del dominio de promulgación de procesos. Este dominio es el encargado de las actividades de producción, control y soporte del proceso.
- *Dominio de ejecución del proceso*: Cubre el conjunto de tareas y actividades reales que realizan los agentes de proceso humanos o no humanos durante el proceso de software.

Zahran (1998) proporciona una definición para la infraestructura de procesos de software que consiste en una infraestructura organizacional y de gestión junto con una infraestructura tecnológica.

► 9.3. Infraestructura organizacional y de gestión

La infraestructura organizacional y de gestión establece varias unidades organizacionales para supervisar la implementación y la mejora de los procesos de software (Bourque y Fairley, 2014). La implementación de mejoras requiere que haya una estructura de mejora apropiada para apoyar estas actividades.

El aspecto personal de los procesos de software incluye dos puntos de vista (Kneuper, 2018):

- Una definición muy restrictiva de los procesos reducirá la motivación y la creatividad de los desarrolladores, quienes sienten una fuerte necesidad de libertad en su trabajo. DeMarco y Lister (2013) introdujeron el término *peopleware* para enfatizar la importancia de las personas en el desarrollo de software. Los procesos ágiles ponen más énfasis en las personas que realizan el desarrollo y les dan mucha más libertad sobre cómo realizar su trabajo.
- La cooperación en los equipos es muy difícil sin reglas acordadas y definidas sobre quién debe hacer o entregar qué y cuándo hacerlo. El espectro va desde un proceso estándar completamente definido en la organización hasta un conjunto de reglas internas acordadas por un equipo *scrum* (definición de “hecho”) en el otro extremo.

Los roles involucrados generalmente en la infraestructura organizacional y de gestión serán el grupo directivo de gestión, el Software Engineering Process Group (**SEPG**) y los equipos dedicados de mejora, como se presentan en la figura 9.5.

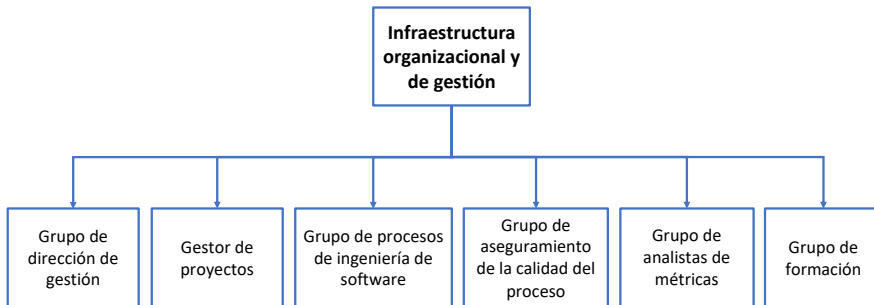


Figura 9.5. Infraestructura organizacional y de gestión

Fuente: elaboración propia.

9.3.1. Grupo de dirección de gestión

Un grupo directivo para supervisar las mejoras y proporcionar dirección, recursos y fondos para la mejora.

9.3.2. Gestor de proyectos

El gestor de proyectos tiene como responsabilidades: planificación y monitoreo del programa de mejora, gestión del compromiso de los *stakeholders*, desarrollo del informe de estado a la alta gerencia, participación en reuniones del **SEPG**, ayudar a lograr los planes y realizar la gestión de riesgos.

9.3.3. Software Engineering Process Group (SEPG)

Este grupo es responsable tanto de los activos de proceso como de las actividades de mejora de procesos de software de la organización. El **SEPG** ayuda a institucionalizar una cultura de mejora de procesos y un enfoque estructurado para la definición, implementación y adaptación de procesos.

El **SEPG** coordina las actividades de mejora de procesos, realiza un seguimiento del progreso, recopila las mejores prácticas y mide la mejora. El **SEPG** debe establecer equipos de mejora específicos para llevar a cabo acciones de mejora específicas, como la definición de nuevos procesos y de material de capacitación para implementarlos.

Algunas de las responsabilidades desempeñadas por el **SEPG** son (Chaudhary y Chopra, 2017):

- Asegura que los procesos relevantes estén definidos y alineados con los objetivos del negocio para apoyar la entrega de productos o servicios.
- Asegura que los activos del proceso se revisen y mantengan regularmente en función de las mejoras identificadas.
- Revisa y aprueba la adaptación del proceso.
- Asegura que los procesos se evalúen periódicamente para identificar sus fortalezas y debilidades.

9.3.4. Grupo de Process Quality Assurance (PQA)

El grupo de Process Quality Assurance (**PQA**) está formado por personas que tienen un buen conocimiento sobre aseguramiento de la calidad del proceso y del producto. Las responsabilidades típicas de este grupo incluyen, entre otras (Chaudhary y Chopra, 2017):

- Revisiones/auditorías de proyectos.
- Realizar entrenamientos del proceso.
- Revisiones de calidad de productos y procesos.
- Verificaciones de cumplimiento de procesos.
- Cotejo y análisis de métricas del proyecto.
- Proporcionar actualizaciones de mejora de procesos relevantes para el **SEPG**.

9.3.5. Grupo de analistas de métricas

Es un subgrupo del **SEPG**. Los miembros de este grupo recopilan y analizan datos de métricas del proyecto. Su análisis servirá para la toma de decisiones. Entre sus responsabilidades típicas se encuentran (Chaudhary y Chopra, 2017):

- Recopilación de datos de métricas de proyectos.
- Realizar análisis de métricas.
- Utilizar el análisis para identificar y establecer nuevas métricas de referencia.

9.3.6. Grupo de formación

El papel principal de este grupo es (Chaudhary y Chopra, 2017):

- ▶ Identificar las necesidades de formación y capacitación en la organización.
- ▶ Planificar y realizar capacitaciones.
- ▶ Evaluar la efectividad de la capacitación.
- ▶ Mejorar el proceso de entrega de la capacitación.

▶ 9.4. Infraestructura tecnológica

El proceso de software requiere, además de la infraestructura organizacional y de gestión, elementos tecnológicos que brinden soporte al proceso. Esta infraestructura está conformada por diferentes ambientes y herramientas que han surgido para apoyar la definición, el modelado, la medición, el control y la mejora del proceso de software.

La infraestructura tecnológica permite integrar los procesos de producción y de gestión en los proyectos software. Algunas de estas tecnologías representativas de soporte al proceso son Process-Centered Software Engineering Environment (**PSEE**), manuales de procesos, guías electrónicas de procesos, bibliotecas de activos de proceso y wikis (figura 9.6).

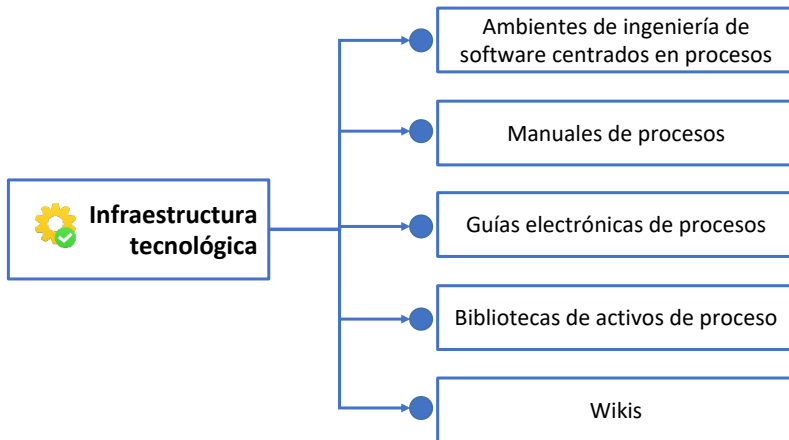


Figura 9.6. Infraestructura tecnológica del proceso de software

Fuente: elaboración propia.

9.4.1. Ambientes de ingeniería de software centrados en procesos

Poner en práctica un modelo de proceso se llama ejecución (*enactment*) de procesos. Las prácticas de ingeniería de software se pueden aplicar, guiar y automatizar a través de la ejecución de procesos. Un **PSEE** es un entorno que proporciona diversos servicios orientados a desarrolladores de software mediante la ejecución de modelos de procesos. Por ejemplo, la asistencia interactiva durante el desarrollo de software, la automatización de tareas rutinarias, la invocación y el control de herramientas de desarrollo de software y la aplicación de reglas y prácticas obligatorias (Matinnejad y Ramsin, 2012). Estos ambientes ofrecen la opción de supervisar la ejecución del proceso: orden de ejecución de las actividades del proceso y producción y consumo de artefactos (Chaghrouchni *et al.*, 2016). No hay que confundirlos con los sistemas de *workflow* o sistemas cooperativos, aunque pueden basarse en ellos.

Los **PSEE** generalmente están dedicados a apoyar proyectos de desarrollo de software que se llevan a cabo a través de un conjunto de pasos que definen sus procesos. Los **PSEE** desempeñan un papel importante en la comprensión del proceso, la capacitación, al contratar nuevos ingenieros e, incluso, mejorar las estrategias de organización, son el núcleo de cualquier proceso de desarrollo de software (Smatti *et al.*, 2015).

Entre algunos de los requisitos de los **PSEE** se encuentran (Matinnejad y Ramsin, 2012):

- *Apoyo a la promulgación*: Permiten la existencia de una forma natural y simple para implementar el paradigma de definición/instanciación/promulgación de procesos. Se requiere un entorno de ejecución basado en lenguajes de modelado de procesos y se logra un nivel de automatización de las actividades del proceso.
- *Distribución del equipo de software*: Admiten la comunicación y la coordinación no solo entre los usuarios, sino también entre los usuarios y los elementos de proceso automatizados.
- *Gestión de la coherencia*: Gestionan, además de los procesos de software, los productos de software que están sujetos a cambios frecuentes. La preservación de la coherencia se puede realizar gestionando las relaciones de dependencia que permiten rastrear la cadena de cambios en los productos de software o mediante la definición de predicados que examinen las condiciones de coheren-

cia y desencadenen una excepción en caso de cualquier violación de consistencia.

- ▶ *Flexibilidad del proceso*: Es la capacidad de modificar dinámicamente las actividades del proceso de software durante su ejecución. De esta manera, es posible modificar y adaptar el orden de las acciones del proceso.
- ▶ *Evolución del proceso*: El proceso real está sujeto a errores, descuidos y desviaciones del proceso estándar. Los **PSEE** ayudan a mejorar el proceso de software, reflejándose en una mejora de los productos de software.
- ▶ *Requisitos modernos*: Los **PSEE** deben incluir técnicas modernas de desarrollo, como procesos ágiles, desarrollo de software *open-source*, seguridad y desarrollo de aplicaciones móviles.

Existen muchos **PSEE** en la industria, como **DOSDE** (De Oliveira *et al.*, 2004), **CASDE** (Jiang *et al.*, 2006), Transforms (Maciel *et al.*, 2009), **SPACE** (Weber *et al.*, 2009), **ADAMS** (De Lucia *et al.*, 2004), entre otros.

9.4.2. Manuales del proceso

Los manuales del proceso definen el trabajo que se debe hacer, los resultados esperados, las responsabilidades de los roles, las condiciones que controlan la secuencia del trabajo y realizar seguimiento del progreso, etc. Generalmente se organizan de acuerdo con la descomposición del trabajo, con capítulos para subprocesos importantes, secciones para actividades principales en cada subproceso y subsecciones para tareas específicas (Münch *et al.*, 2012).

El contenido y la organización de un manual de procesos reflejan posibles preguntas relacionadas con las tareas realizadas por los usuarios del proceso. Algunas preguntas típicas son (Münch *et al.*, 2012):

- ▶ ¿Qué se debe hacer para realizar una actividad?
- ▶ ¿Cuáles son los requisitos para realizar una tarea?
- ▶ ¿Qué se debe producir durante la ejecución de una tarea?
- ▶ ¿Por qué se debe realizar una tarea de una manera específica?
- ▶ ¿Cuáles son las tareas responsables de un determinado rol?
- ▶ ¿Cómo se determina si se ha realizado con éxito una tarea?
- ▶ ¿Dónde se puede encontrar una plantilla en particular para realizar una determinada tarea?

El contenido de los manuales de procesos comprende generalmente (Münch *et al.*, 2012):

- Una descripción de las actividades que un usuario del proceso debe ejecutar.
- Una descripción de los productos de trabajo producidos por cada actividad.
- Una descripción de los roles asociados con las actividades.
- Información opcional:
 - Descripción de los recursos o las herramientas que se utilizarán durante las actividades.
 - Plantillas de documentos para los productos de trabajo.
 - Ejemplos o instancias de productos de trabajo.
 - Instrucciones para tareas específicas.
 - Consejos y trucos.

9.4.3. Guías electrónicas de procesos

Las guías electrónicas de procesos son representaciones electrónicas de una descripción del proceso de software. Estas guías sirven para documentar y publicar información del proceso de una forma fácilmente accesible (Scott *et al.*, 2002). En comparación con los manuales de proceso, pueden capturar realmente la estructura en red de un proceso. Una guía electrónica de proceso generalmente se genera mediante un entorno de modelado de procesos, es decir, utiliza el modelo de proceso como entrada y genera una representación basada en la web, es decir, un conjunto de páginas web interconectadas. Las páginas web entrelazadas permiten seguir fácilmente todas las rutas posibles a través del proceso (Münch *et al.*, 2012).

Una guía electrónica del proceso puede verse como un documento de referencia estructurado, orientado al flujo de trabajo para un proceso particular que existe para ayudar a los usuarios del proceso a llevar a cabo el proceso previsto (Kellner *et al.*, 1998). Las guías electrónicas de proceso contienen definiciones de proceso y servicios, como navegación, búsqueda y almacenamiento de la información del estado del proceso. Además de otros servicios adicionales como (Münch *et al.*, 2012):

- Proporcionar plantillas para un producto de trabajo específico que se pueden descargar desde una intranet.

- ▶ Ejemplos de “buenos” productos de trabajo para mostrar a los usuarios del proceso que sirvan de guía y ayuden a crear el producto de trabajo respectivo.
- ▶ Áreas de discusión que brindan comentarios directos sobre la descripción del proceso.
- ▶ Presentar consejos útiles para realizar una determinada tarea.
- ▶ Respalda la ejecución del proceso al proporcionar diferentes puntos de vista sobre el proceso.
- ▶ Proporcionar diferentes vistas de acuerdo con el papel ayudando a las personas a concentrarse en su rol específico y no saturándolas con detalles innecesarios.
- ▶ Proporcionar vistas de proceso específicas bajo demanda, al generarlas a partir del modelo de proceso subyacente.

Las guías electrónicas de proceso incluyen los siguientes elementos básicos (Moe y Dybå, 2006):

- ▶ *Actividades*: Descripciones sobre cómo se hacen las actividades, incluida una descripción general de las actividades y los detalles de cada actividad individual.
- ▶ *Artefactos*: Descripciones de productos creados o modificados por una actividad, ya como resultado final, ya como intermedio de la actividad.
- ▶ *Agentes*: Descripciones de entidades que pueden realizar actividades. Un agente puede ser un individuo o un grupo.
- ▶ *Roles*: Detalles sobre los roles y agentes involucrados en la realización de las actividades.
- ▶ *Recursos*: Detalles sobre las herramientas y técnicas utilizadas para soportar o automatizar la ejecución de una actividad.

Meso y Jain (2006) distinguen entre guías de proceso desarrolladas para propósitos específicos, “enfoques fuertes de resolución de problemas” y guías de proceso para propósitos generales, “enfoques débiles de resolución de problemas”.

9.4.4. Bibliotecas de activos de procesos

Las bibliotecas de activos de proceso (Process Asset Library [PAL]) son colecciones de activos de proceso utilizados por una organización o proyec-

to durante la definición, implementación, gestión y ejecución del proceso (Software Engineering Institute [SEI], 2010).

Los activos de proceso son cualquier elemento que la organización considere útil para realizar con éxito el proceso de software (Bermón Angarita, 2010; García *et al.*, 2011). El término activo se utiliza para resaltar que estos artefactos permiten cumplir los objetivos de negocio de la organización y que son “inversiones” con las cuales se espera alcanzar valores agregados actuales y futuros (SEI, 2010). Además, los activos proveen el fundamento para institucionalizar el proceso en la organización.

En la tabla 9.2, se presenta un listado de posibles activos de proceso desarrollados y mantenidos por una organización.

Tabla 9.2. Lista de activos de proceso

Activos	Descripción
Ciclos de vida	Periodos consistentes en fases que comienzan cuando un producto es concebido y finalizan cuando el producto ya no está disponible para su uso.
Procesos	Definiciones de los principales componentes de un proceso para la realización de proyectos junto con mediciones recolectadas durante su ejecución.
Guías y criterios de adaptación	Guías que permiten a proyectos, grupos o funciones organizacionales adaptar apropiadamente el proceso estándar para su utilización.
Plantillas	Esqueletos que proporcionan la estructura de los documentos que pueden ser utilizados para crear productos de trabajo en las descripciones de procesos.
Guías	Información útil y detallada sobre cómo realizar una actividad.
Estándares	Reglas aplicables a un tipo particular de producto de trabajo.
Listas de verificación	Ayudas para valorar la completitud o exactitud de una actividad o producto de trabajo.
Lecciones aprendidas	Buenos ejemplos tomados de proyectos pasados que pueden ser activos útiles para la realización de proyectos futuros.
Material de entrenamiento	Materiales de referencia para entrenamiento sobre procesos, herramientas o métodos que pueden ser activos valiosos.
Mediciones	Medidas sobre la ejecución y utilización de los procesos para entender el proceso.
Otros	Cualquier documentación relacionada con el proceso de software.

Fuente: elaboración propia con base en Layman (2005).

Las **PAL** son un requisito obligatorio para alcanzar el nivel 3 de Capability Maturity Model Integration (**CMMI**) debido a que facilitan la estandarización y mejora del proceso. El área de proceso de definición de procesos de la organización (Organizational Process Definition [OPD]) define una práctica general: establecer los activos de proceso de la organización, la cual tiene su quinta práctica específica: establecer la biblioteca de activos de proceso de la organización. Las subprácticas y los productos de trabajo asociados a esta quinta práctica específica se muestran en la figura 9.7.

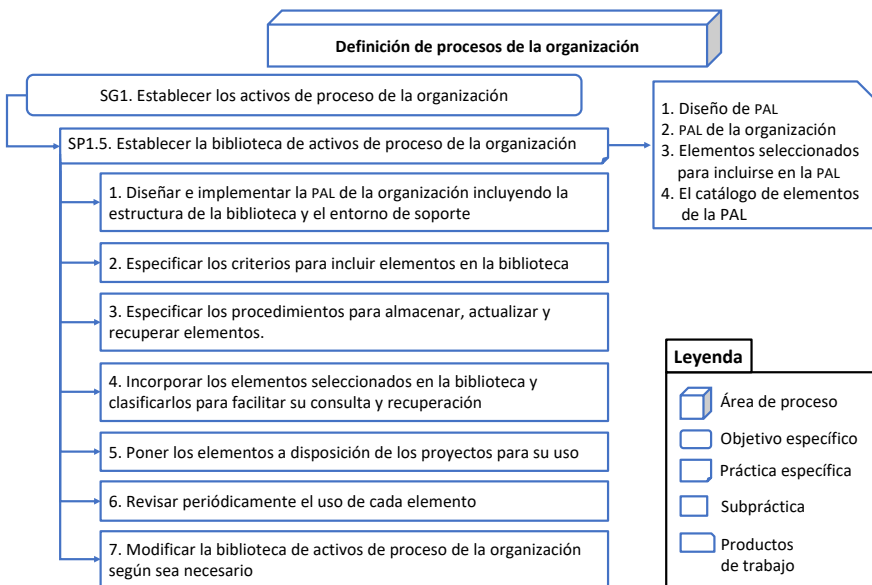


Figura 9.7. Definición de procesos de la organización en CMMI nivel 3

Fuente: elaboración propia.

Algunos atributos de una **PAL** son (Amescua *et al.*, 2010; García y Turner, 2007):

- ▶ Fácil navegación y búsqueda.
- ▶ Múltiples vistas de información basada en diversos criterios definidos por el usuario.
- ▶ Seguimiento del estado de documentos y la gestión de versiones de los elementos almacenados.
- ▶ Soporte a la gestión de peticiones de cambios desde múltiples fuentes.

- Accesos libres y restringidos a la biblioteca.
- Promover la utilización de activos de proceso, no su creación o mantenimiento.

9.4.5. Wikis

Wikipedia (s. f.) define una *wiki* como “una comunidad virtual, cuyas páginas son editadas directamente desde el navegador, donde los mismos usuarios crean, modifican, corrigen o eliminan contenidos que, habitualmente, comparten”. Son tecnologías para la comunicación, documentación y difusión del conocimiento, convirtiéndose en espacios para el desarrollo de contenidos en línea que permiten a los usuarios crear, editar y distribuir información en la web. Su gran ventaja es permitir que un sitio web sea completamente editable ofreciendo una forma rápida de colaboración durante su creación y actualización (Hasan y Pfaff, 2006).

Las paginas wiki pueden ser visualizadas de dos modos (Leuf y Cunningham, 2001):

- *Lectura*: Los usuarios pueden ver los contenidos como una página web normal que contiene imágenes, enlaces, texto, etc.
- *Edición*: Se activa la adición y modificación de contenidos por parte de los usuarios utilizando la sintaxis propia de la wiki.

Entre las funcionalidades básicas de las wikis se encuentran las presentadas en la tabla 9.3.

Las wikis son comunes en el desarrollo ágil haciendo énfasis en recopilar pautas acordadas dentro del esfuerzo de desarrollo, así como en descripciones de apoyo adicionales sobre cómo realizar ciertos pasos. Todos los miembros del equipo pueden editar las descripciones a medida que utilizan la wiki. Además, van más allá de documentar el proceso que se utilizará, actúan como un depósito para documentación del proyecto, y así apoyar la colaboración dentro del esfuerzo de desarrollo (Dengler *et al.*, 2009). Si se utilizan para documentar los procesos de software, se utilizan al menos dos roles diferentes: los lectores y autores de la documentación del proceso.

Una de las principales ventajas de los wikis para apoyar el despliegue del proceso en las organizaciones es que son muy fáciles de usar tanto para los autores como para los lectores de los documentos. Además, ayudan a garantizar que los usuarios del proceso siempre trabajen con la última versión actualizada de la documentación.

Tabla 9.3. Funcionalidades de las wikis

Funcionalidad	Descripción
Repositorio central	Las páginas de las wikis deben estar almacenadas en un repositorio central, en una base de datos o en archivos de texto.
Edición	La edición y organización de contenidos de cada página es flexible y pueden ser cambiados cuando sea necesario.
Enlaces	Cada página puede ser enlazada a otras páginas formando una estructura de red.
Historia	Permite almacenar versiones o modificaciones previas de cualquier página, y así realizar un seguimiento del proceso de edición.
Cambios recientes	Consultar una visión general de un cierto número de cambios recientes a las páginas o todos los cambios en un cierto periodo.
Búsqueda	Ofrecen funciones de búsqueda de texto completo o por título de página.
Categorías	Permiten mantener las páginas organizadas y ordenadas.
Seguridad	Características para vigilar páginas o detener a usuarios no autorizados que editen páginas particulares.
Discusiones	Incorporan discusiones sobre las páginas.
Plantillas	Proporcionan mecanismos para insertar textos estándares.
Extensiones	Funcionalidades adicionales pueden añadirse a las wikis por medio de <i>plugins</i> o extensiones.

Fuente: elaboración propia con base en Ebersbach *et al.* (2008).

► 9.5. Formación en procesos de software

La formación es una tarea importante para implementar y utilizar procesos de software. Puede suceder en diferentes niveles, ya sea en el nivel de los procesos estándares o en el nivel inferior de las organizaciones que desean implementar un determinado proceso y, por tanto, necesitan capacitar a sus miembros en el uso de ese proceso (Kneuper, 2018).

Un programa de capacitación organizacional involucra (SEI, 2010):

- Establecer las necesidades de formación y capacitación de la organización.

- Proporcionar actividades de capacitación para satisfacer esas necesidades.
- Establecer y mantener la capacidad de capacitación.
- Establecer y mantener registros de las capacitaciones realizadas.
- Evaluar la efectividad de la capacitación.

La formación se puede proporcionar de varias maneras: aula, programas informales de mentores, autoestudio guiado, entrenamiento basado en la web y sesiones de entrenamiento virtual.

Las metas y prácticas específicas para el proceso de formación según el nivel de 3 de **CMMI** se muestran en la tabla 9.4.

Tabla 9.4. Requisitos CMMI para la formación

Objetivo específico	Práctica específica	Descripción
Objetivo específico 1	Establecer una capacidad de formación organizacional	
	Práctica específica 1.1	Establecer necesidades estratégicas de formación
	Práctica específica 1.2	Determinar cuáles son las necesidades de formación
	Práctica específica 1.3	Establecer un plan táctico de formación organizacional
	Práctica específica 1.4	Establecer capacidad de formación
Objetivo específico 2	Proporcionar la formación necesaria	
	Práctica específica 2.1	Realizar la formación
	Práctica específica 2.2	Establecer registros de formación
	Práctica específica 2.3	Evaluar la efectividad de la formación

Fuente: elaboración propia con base en SEI (2010).

En la figura 9.8, se detalla un diagrama de flujos sobre el proceso de formación. El diagrama incluye actividades para identificar las necesidades de formación, planificar la formación para satisfacer esas necesidades, realizar la formación y construir materiales de formación de la organización.

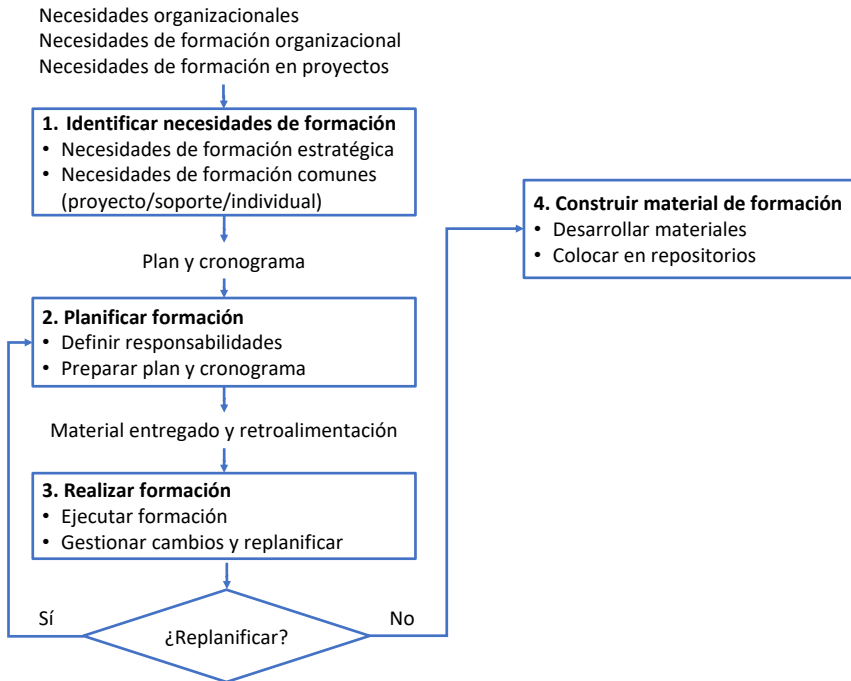


Figura 9.8. Diagrama de actividades del proceso de formación

Fuente: elaboración propia con base en Kneuper (2018).

► 9.6. Retorno de la inversión en SPI

Una percepción errónea común es pensar que establecer una infraestructura de proceso de software e implementar procesos de software repetibles agregará tiempo y costo al desarrollo de software. Hay un costo asociado con la introducción o mejora de un proceso de software. Sin embargo, la experiencia ha demostrado que la implementación de mejoras de los procesos de software tiende a resultar en un menor costo a través de una mayor eficiencia, evitando el reprocesamiento y un software más confiable y asequible (Bourque y Fairley, 2014).

Dybå (2005) identifica seis factores críticos de éxito en iniciativas **SPI** que influyen en el nivel de éxito percibido por la organización y su desempeño organizacional:

- *Orientación empresarial:* Alcanzar los objetivos de negocio es la meta de cualquier programa de cambio y el papel de la mejora del proceso se ha reconocido durante mucho tiempo como factor esencial para lograr la excelencia empresarial y la ventaja competitiva.
- *Liderazgo involucrado:* Un cambio importante requiere liderazgo. Por tanto, se necesita compromiso y participación de los líderes organizacionales en la implementación de acciones de mejora.
- *Participación de los empleados:* Su participación y la forma en que se trata a las personas se ha señalado como un factor crucial en la gestión y el desarrollo organizacional.
- *Interés por la medición:* La medición es una parte esencial de la comprensión, el control, el monitoreo, la predicción y evaluación de proyectos de desarrollo y mantenimiento de software, así como de cualquier iniciativa **SPI**.
- *Estrategia de aprendizaje:* Se logra mediante la implementación de dos tipos de estrategias.
 - *Explotación:* Adopción y uso del conocimiento y experiencia existentes. Implica mejorar las capacidades al refinar, estandarizar y elaborar ideas, paradigmas, tecnologías, estrategias y conocimientos establecidos.
 - *Exploración:* Implica aprender a través del descubrimiento y experimentar con ideas, paradigmas, tecnologías, estrategias y conocimientos con la esperanza de encontrar nuevas alternativas y oportunidades que sean superiores a la práctica actual.

Desde una perspectiva práctica, estos factores sugieren que, en lugar de tratar de centrarse en los procedimientos técnicos, las organizaciones de software deberían centrar sus esfuerzos de **SPI** en crear una cultura organizacional dentro de la cual estos procedimientos puedan prosperar.

El retorno de la inversión (Return on Investment [**ROI**]) es un enfoque ampliamente utilizado para medir el valor de un proceso o tecnología nuevos o mejorados. El **ROI** también se puede utilizar para medir el valor económico de los enfoques populares para la mejora de procesos de software (Rico, 2005).

El **ROI** es una relación aritmética entre beneficios y costos, expresada como un porcentaje, es decir, los beneficios menos los costos son el numerador y los costos son el denominador, multiplicando el resultado por

100 %. Los costos y beneficios se expresan con mayor frecuencia en términos económicos o monetarios. Un gran **ROI** indica dinero bien invertido o gastado (Rico, 2005).

Las métricas de **ROI** están diseñadas para medir el valor económico de un proceso de software nuevo o mejorado. Rico (2005) propone seis métricas básicas relacionadas con el **ROI**. Estas métricas son costos, beneficios, relación costo-beneficio, **ROI**, valor presente neto (**VPN**) y punto de equilibrio. En la tabla 9.5, se presenta este listado de métricas que pueden utilizarse para calcular y evaluar el **ROI** de proyectos **SPI**.

Tabla 9.5. Métricas **ROI** para **SPI**

Métrica	Definición	Fórmula
Costos	Cantidad total de dinero gastado en un proceso de software nuevo o mejorado	$\sum_{i=1}^n Costo_i$
Beneficios	Cantidad total de dinero ganado en un proceso de software nuevo o mejorado	$\sum_{i=1}^n Beneficio$
Tasa B/C	Tasa de relación entre beneficios y costos	$\frac{Beneficios}{Costos}$
ROI	Tasa ajustada de relación entre beneficios y costos	$\frac{Beneficios - Costos}{Costos} \times 100$
VPN	Flujo de caja	$\sum_{i=1}^{años} \frac{Beneficios}{(1 + TasaDescuento)^{años}} - Costos$
Punto de equilibrio	Punto en que los beneficios cumplen o exceden a los costos	$\frac{Costos}{Costos antiguos - Costos nuevos - 1}$

Fuente: elaboración propia con base en Rico (2005).

Calcular y analizar el **ROI** para proyectos **SPI** es relevante en las organizaciones desarrolladoras de software para (Van Solingen, 2004):

- Convencer a los gerentes para que inviertan dinero y esfuerzo en mejorar y convencerlos de que las iniciativas **SPI** pueden ayudar a resolver problemas estructurales.

- Estimar cuánto esfuerzo es necesario invertir para resolver un determinado problema.
- Decidir qué mejora de proceso implementar primero realizando priorizaciones debido a limitaciones de tiempo y recursos.
- Analizar los presupuestos de **SPI** en los que las organizaciones deben mostrar un **ROI** suficiente.
- Sobrevivir ya que cualquier inversión en una organización debe valorarse contra su retorno.

► Desafíos

La infraestructura organizacional y tecnológica para implementar de forma exitosa un proceso de software es una tarea compleja que tiene varios desafíos. En primer, utilizar **TI** como una **PAL** es una buena idea. Sin embargo, implementar una **PAL** exitosa debe considerar aspectos esenciales, como (Rogers, 2004):

- Si la **PAL** incluye actividades de aprendizaje, estas deben ser útiles para todos los usuarios del proceso de software y no solo para los futuros usuarios.
- Debe quedar claro cómo una **PAL** ayuda al equipo actual a realizar mejor su trabajo.
- Una **PAL** debe ser un repositorio dinámico con información del proceso, el cual debe estar continuamente actualizado y ser realmente consultado y utilizado durante la ejecución de proyectos.
- Una **PAL** tiene el potencial de ofrecer un valor muy alto a la organización. Por tanto, requiere inversiones necesarias en tiempo y esfuerzo por parte de los participantes.

Por otro lado, el **SEPG** sirve como un catalizador para los programas **SPI** que emprenda una organización de software. Sin embargo, uno de los mayores desafíos para el **SEPG** es mantener la motivación y el entusiasmo por la mejora de procesos en todos los niveles de la organización, además de cambiar la cultura organizacional para garantizar el uso efectivo de nuevos procesos (Kaltio, 2001).

Con respecto al uso de **PSEE**, la proliferación de herramientas y diferentes niveles de rigidez hacen difícil cumplir y garantizar la ejecución exitosa de los procesos de software. Así que los **PSEE** deben estar disponibles,

ser flexibles y no intrusivos (flexibles y simples) garantizando la transparencia de la información y permitiendo el acceso distribuido a herramientas y objetos. Además, ni el soporte aislado para actividades individuales (y su integración *ad hoc*) ni el soporte sofisticado para un solo tipo de proceso de software son aceptables para la multitud de procesos de software actuales (Gruhn, 2002). Muchos PSEE tienen limitaciones, como el soporte restringido para un proceso de software específico, un enfoque restringido a las tareas de administración y la falta de funciones para el soporte a la ejecución del proceso (Maciel *et al.*, 2013).

► Ejemplos

Un ejemplo de librería de activos de procesos de software es la **PAL** del Departamento de Asuntos de los Veteranos (United States Department of Veterans Affairs [VA]) (figura 9.9).

The screenshot shows the VA Process Asset Library (PAL) website. The header includes the VA logo and navigation links: "VA Benefits and Health Care", "About VA", "Find a VA Location", and "My VA". A search bar and "Sign in" button are also present. The main content area is titled "Welcome to the Office of Information and Technology (OIT) Process Asset Library Website". Below this, there is a section for "Process Asset Library" with a list of templates and guides, including "Artifact Review Agenda and Minutes Template", "Digital Signature Guide", "Documentation Style Guide", "Lessons Learned Report Template", "Master Test Plan Template", "Open Source Submission Checklist", "PIV Credential Identity Verification Matrix", "Position Task Risk Designation Template", "Product Documentation Artifacts Approval Signatures Template", "Product Documentation Review Checklist", "Project Management Plan (PMP) Template", "Record of Notification Template", "Requirements Specification Document Template", "Review Findings Summary Template", "Service Level Agreement Template", "Site Certification Checklist", "Test Evaluation Template", "TMS Self Registration OIT Contractors Jobaid Guide", and "VISTA Patch Template with Instructions". On the right, there is a "RESOURCES" section with links to "VA Internet", "Process Asset Library (PAL) Artifacts", and "Process Maps". Below that is a "CONNECT WITH US" section with a "Subscribe to Receive Email Updates" form and contact information for Benefits (1-800-827-1000), Health Care (1-877-222-VETS (8387)), and VA Insurance (General: 1-800-...

Figura 9.9. PAL de la VA

Fuente: Veteran Affairs, 2022.

Esta **PAL** contiene artefactos del proceso de software, generalmente documentos de Microsoft Word que incluyen plantillas, procesos detallados, guías y listas de verificación, como plantilla de agenda y actas de revisión de artefactos, guía de firma digital, guía de estilo de documentación, plantilla de informe de lecciones aprendidas, plantilla del plan maestro de pruebas, lista de verificación de envío de código abierto, plantilla de firmas de aprobación de artefactos de documentación de productos, lista de verificación de revisión de documentación del producto, plantilla del plan de gestión de proyectos, plantilla del documento de especificación de requisitos, plantilla de resumen de hallazgos de revisión, plantilla de acuerdo de nivel de servicio, lista de verificación de certificación del sitio, plantilla de evaluación de pruebas, entre otras.

De esta manera, esta **PAL** se convierte en un repositorio de información que puede ser consultado por los usuarios del proceso de software para realizar de manera oportuna y guiada algunas actividades de desarrollo.

► Resumen

Los procesos de software nuevo o mejorados deben ser implementados en las organizaciones. Su implementación y uso real en las organizaciones se denomina despliegue del proceso. Para realizar un despliegue efectivo, se debe contar con una infraestructura adecuada basada en recursos humanos y financieros, tecnologías y formación.

Los nuevos o mejorados procesos pueden presentar resistencia al cambio por parte de los usuarios que los utilizarán. Las organizaciones en cabeza de sus líderes deben convencer y demostrar con hechos y ejemplos que los nuevos o mejorados procesos sí apoyan realmente sus actividades y permiten realizar un mejor trabajo.

El marco conceptual de la infraestructura del proceso está basado en tres dominios: modelo de proceso, promulgación de procesos y ejecución del proceso.

La infraestructura organizacional involucra los roles del grupo directivo de gestión, el **SEPG**, el grupo de aseguramiento de la calidad del proceso, el grupo de analistas de métricas y el grupo de formación.

La infraestructura tecnológica brinda soporte para la adecuada ejecución del proceso e incluye herramientas, como **PSEE**, manuales de procesos, guías electrónicas de procesos, bibliotecas de activos de proceso y wikis.

La formación en los nuevos o mejorados procesos de software es un aspecto importante del despliegue. Se requiere, por tanto, establecer una capacidad de formación organizacional y proporcionar la formación necesaria a los diferentes usuarios del proceso.

Por último, los nuevos o mejorados proceso de software se deben evaluar desde el punto de vista financiero a partir del retorno de la inversión. Se pueden utilizar diferentes métricas tomadas de la evaluación financiera de proyectos.

► Ejercicios

1. ¿Qué es la infraestructura para el proceso de software?
2. ¿Qué es el despliegue del proceso de software y cuál es su importancia?
3. Describir en qué consiste una **PAL** de una organización de software.
4. ¿Cuáles son las responsabilidades del **SEPG**?
5. Describir qué características debe tener un **PSEE**.
6. Realizar un cuadro comparativo del uso de guías electrónicas de procesos, manuales de proceso, wikis y **PAL** como infraestructura tecnológica del proceso de software.
7. ¿Qué artefactos candidatos seleccionaría para almacenar en una **PAL** relacionados con las siguientes áreas de proceso: diseño arquitectónico, diseño detallado, codificación, planificación y gestión de la configuración?
8. Investigar en internet diferentes infraestructuras tecnológicas orientadas al proceso de software.

► Referencias

- Amescua, A., Bermón Angarita, L., García, J. y Sánchez-Segura, M. I. (2010). Knowledge repository to improve agile development processes learning. *JET Software*, 4(6), 434-444. <https://doi.org/10.1049/iet-sen.2010.0067>
- Bayona, S., Calvo Manzano, J., Cuevas, G. y San Feliu, T. (2013). Identify and classify the critical success factors for a successful process deployment. En R. Pooley, J. Coady, C. Schneider, H. Linger, C. Barry y M. Lang (eds.), *Information systems development: Reflections, challenges and new directions* (pp. 11-22). Springer. https://doi.org/10.1007/978-1-4614-4951-5_2

- Bermón Angarita, L. (2010). *Librería de activos para la gestión del conocimiento sobre procesos de software: PAL-Wiki* [tesis de doctorado, Universidad Carlos III de Madrid]. <https://e-archivo.uc3m.es/handle/10016/10231#preview>
- Bourque, P. y Fairley, R. E. (eds.) (2014). *Guide to the Software Engineering Body of Knowledge, Version 3.0*. IEEE Computer Society. <https://cs.fit.edu/~kgallagher/Schtick/Serious/SWEBOKv3.pdf>
- Chaghrouchni, T., Kabbaj, M. I. y Bakkoury, Z. (2016). Optimized approach for dynamic adaptation of process models. En A. El Oualkadi, F. Choubani y A. El Moussati (eds.), *Proceedings of the Mediterranean Conference on Information & Communication Technologies 2015. Lecture Notes in Electrical Engineering* (pp. 101-109). Springer. https://doi.org/10.1007/978-3-319-30298-0_11
- Chaudhary, M. y Chopra, A. (2017). *CMMI for development: Implementation guide*. Apress. [http://ndl.ethernet.edu.et/bitstream/123456789/27112/1/Mukund %20Chaudhary.pdf](http://ndl.ethernet.edu.et/bitstream/123456789/27112/1/Mukund%20Chaudhary.pdf)
- Christensen, C. M. y Overdorf, M. (2000). Meeting the challenge of disruptive change. *Harvard Business Review*, 78(2), 66-77. http://innovbfa.viabloga.com/files/HBR___Christensen___meeting_the_challenge_of_disruptive_change___2009.pdf
- De Lucia, A., Fasano, F., Francese, R. y Tortora, G. (2004). ADAMS: An artefact-based process support system. En F. Maurer y G. Ruhe (eds.), *Proceedings of the Seventh International Conference on Software Engineering and Knowledge Engineering (APEIE)* (pp. 31-36). IEEE.
- De Oliveira, K. M., Zlot, F., Rocha, A. R., Travassos, G. H., Galotta, C. y de Menezes, C. S. (2004). Domain-oriented software development environment. *Journal of Systems and Software*, 72(2), 145-161. [https://doi.org/10.1016/S0164-1212\(03\)00233-4](https://doi.org/10.1016/S0164-1212(03)00233-4)
- DeMarco, T. y Lister, T. (2013). *Peopleware: Productive projects and teams*. Addison-Wesley.
- Dengler, F., Lamparter, S., Hefke, M. y Abecker, A., (2009). Collaborative process development using Semantic MediaWiki. En K. Hinkelmann y H. Wache (eds.), *WM2009: 5th Conference on Professional Knowledge Management* (pp. 97-107). Gesellschaft für Informatik e.V. <https://new-dl.gi.de/bitstream/handle/20.500.12116/23326/gi-proc-145-008.pdf?sequence=1&isAllowed=y>
- Dowson, M. (1993). Consistency maintenance in process sensitive environments. En *Proceedings of Workshop on Process Sensitive Environments Architectures*. Rocky Mountain Institute of Software Engineering.
- Dybå, T. (2005). An empirical investigation of the key factors for success in Software Process Improvement. *IEEE transactions on Software Engineering*, 31(5), 410-424. <https://doi.org/10.1109/TSE.2005.53>
- Ebersbach, A., Glaser, M., Heigl, R. y Warta, A. (2008). *Wiki: Web collaboration* (2.^a ed.). Springer.
- García, J., Amescua, A., Sánchez, M. I. y Bermón Angarita, L. (2011). Design guidelines for software processes knowledge repository development. *Information and Software Technology*, 53(8), 834-850. <https://doi.org/10.1016/j.infsof.2011.03.002>
- García, S. y Turner, R. (2007). *CMMI survival guide: Just enough process improvement*. Addison-Wesley.

- Gruhn, V. (2002). Process-centered software engineering environments: A brief history and future challenges. *Annals of Software Engineering*, 14, 363-382. <https://doi.org/10.1023/A:1020522111961>
- Hasan, H. y Pfaff, C. C. (2006). The Wiki: An environment to revolutionise employees' interaction with corporate knowledge. En *Proceedings of the 18th Australia conference on Computer-Human Interaction: Design: Activities, Artefacts and Environments* (pp. 377-380). <https://doi.org/10.1145/1228175.1228250>
- Henderson, R. M. y Clark, K. B. (1990). Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly*, 35, 9-30. <https://doi.org/10.2307/2393549>
- Humphrey, W. S. (2005). The software process: Global goals. En M. Li, B. Boehm y L. J. Osterweil (eds.), *Unifying the Software Process Spectrum. SPW 2005. Lecture Notes in Computer Science* (pp. 35-42). Springer. https://doi.org/10.1007/11608035_4
- Jiang, T., Ying, J., Wu, M. y Fang, M. (2006). An architecture of process-centered context-aware software development environment. En *2006 10th International Conference on Computer Supported Cooperative Work in Design* (pp. 1-5). **IEEE**. <https://doi.org/10.1109/CSCWD.2006.253193>
- Kaltio, T. (2001). *Software process asset management and deployment in a multi-site organization* [tesis de doctorado, Helsinki University of Technology]. <https://aaltodoc.aalto.fi/server/api/core/bitstreams/627a1f2d-ca62-4915-a2af-5c69ba06d629/content>
- Kellner, M. I., Becker-Kornstaedt, U., Riddle, W. E., Tomal, J. y Verlage, M. (1998). *Process guides: Effective guidance for process participants*. **ISPA Press**. <https://publica-rest.fraunhofer.de/server/api/core/bitstreams/0cae4e01-20d2-490c-b565-3eb6dd58539a/content>
- Kneuper, R. (2018). *Software processes and life cycle models: An introduction to modelling, using and managing agile, plan-driven and hybrid processes*. Springer.
- Layman, B. (2005). Implementing an organizational Software Process Improvement program. *IEEE Software Engineering*, 2, 279-288.
- Leuf, B. y Cunningham, W. (2001). *The wiki way: Quick collaboration on the web*. Addison-Wesley.
- Maciel, R. S. P., da Silva, B. C., Magalhães, A. P. F y Rosa, N. S. (2009). An integrated approach for model driven process modeling and enactment. En *2009 XXIII Brazilian Symposium on Software Engineering* (pp. 104-114). **IEEE**. <https://doi.org/10.1109/SBES.2009.18>
- Maciel, R. S. P., Gomes, R. A., Magalhães, A. P., Silva, B. C. y Queiroz, J. P. B. (2013). Supporting model-driven development using a process-centered software engineering environment. *Automated Software Engineering*, 20, 427-461. <https://doi.org/10.1007/s10515-013-0124-0>
- Matinnejad, R. y Ramsin, R. (2012). An analytical review of process-centered software engineering environments. En *2012 IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems* (pp. 64-73). **IEEE**. <https://doi.org/10.1109/ECBS.2012.11>
- Maurer, R. (2010). *Beyond the wall of resistance: Why 70% of all changes still fail-and what you can do about it*. Bard Press.

- Meso, P. y Jain, R. (2006). Agile software development: Adaptive systems principles and best practices. *Information Systems Management*, 23(3), 19-30. <https://doi.org/10.1201/1078.10580530/46108.23.3.20060601/93704.3>
- Messnarz, R., Ekert, D., Reiner, M. y O'Suilleabhain, G. (2008). Human resources based improvement strategies: The learning factor. *Software Process: Improvement and Practice*, 13(4), 355-362. <https://doi.org/10.1002/spip.397>
- Moe, N. B. y Dybå, T. (2006). The use of an electronic process guide in a medium-sized software development company. *Software Process: Improvement and Practice*, 11(1), 21-34. <https://doi.org/10.1002/spip.250>
- Münch, J., Armbrust, O., Kowalczyk, M. y Sotó, M. (2012). *Software process definition and management*. Springer. <https://doi.org/10.1007/978-3-642-24291-5>
- Nikula, U., Jurvanen, C., Gotel, O. y Gause, D. C. (2010). Empirical validation of the Classic Change Curve on a software technology change project. *Information and Software Technology*, 52(6), 680-696. <https://doi.org/10.1016/j.infsof.2010.02.004>
- Osterweil, L. (2011). Software processes are software too. En P. Tarr y A. Wolf (eds.), *Engineering of Software*. Springer. https://doi.org/10.1007/978-3-642-19823-6_17
- Rico, D. F. (2005). Practical metrics and models for Return on Investment. *TickIT International*, 7(2), 10-16. <https://davidfrico.com/rico05p.pdf>
- Rogers, E. W. (2004). *Introducing the pause and learn (PAL) process: Adapting the Army after action review process to the NASA project world at the Goddard Space Flight Center*. NASA Goddard Space Flight Center Knowledge Management Office.
- Schneider, D. M. y Goldwasser, C. (1998). Be a model leader of change. *Management Review*, 87(3), 41-45.
- Scott, L., Carvalho, L., Jeffery, R., D'Ambra, J. y Becker-Kornstaedt, U. (2002). Understanding the use of an electronic process guide. *Information and Software Technology*, 44(10), 601-616. [https://doi.org/10.1016/S0950-5849\(02\)00080-0](https://doi.org/10.1016/S0950-5849(02)00080-0)
- Smatti, M., Oussalah, M. y Ahmed Nacer, M. (2016). Supporting deviations on software processes: A literature overview. En P. Lorenz, J. Cardoso, L. Maciaszek y M. van Sinderen (eds.), *Software Technologies. ICSoft 2015. Communications in Computer and Information Science* (pp. 191-209). Springer. https://doi.org/10.1007/978-3-319-30142-6_11
- Software Engineering Institute. (2010). *CMMI® para Desarrollo, Versión 1.3*. https://insights.sei.cmu.edu/documents/87/2010_019_001_28782.pdf
- Van Solingen, R. (2004). Measuring the ROI of Software Process Improvement. *IEEE Software*, 21(3), 32-38. <https://doi.org/10.1109/MS.2004.1293070>
- Veterans Affairs. (2022). *Process Asset Library*. <https://www.va.gov/process/artifacts.asp>
- Weber, S., Emrich, A., Broschart, J., Ras, E. y Ünalán, Ö. (2009). Supporting software development teams with a semantic process- and artifact-oriented collaboration environment. En *Software Engineering 2009 - Workshopband* (pp. 243-254). Gesellschaft für Informatik e.V. <https://dl.gi.de/server/api/core/bitstreams/ac0c66ff-4de1-4aad-bfee-da2ffb68ec0f/content>
- Wikipedia. (s. f.). Wiki. <http://en.wikipedia.org/wiki/Wiki>
- Zahran, S. (1998). *Software process improvement: Practical guidelines for business success*. Addison-Wesley.

Procesos de software para pymes

► Objetivos

Los objetivos de este capítulo están orientados a conocer y comprender diferentes estándares, infraestructuras, modelos y técnicas de mejora del proceso de software en pymes desarrolladoras de software. Al finalizar este capítulo, el lector estará en capacidad de:

- Conocer las características de las pymes desarrolladoras de software.
- Entender cómo planificar y adaptar propuestas de mejora del proceso de software a las características propias de las pymes desarrolladoras de software.
- Comprender diferentes propuestas de mejora del proceso de software orientadas a pymes desarrolladoras de software elaboradas por la industria y la academia (estándares, infraestructuras, modelos y técnicas).

Algunas iniciativas de investigación confirman el problema que los marcos Software Process Improvement (**SPI**) son demasiado grandes, costosos y se ejecutan durante largos periodos (hasta obtener el retorno de la inversión), lo cual incentiva el desarrollo de marcos **SPI** de tipo ligero para pymes desarrolladoras de software (Pettersson *et al.*, 2008).

Los modelos de mejora de procesos de software formales como Capability Maturity Model Integration (**CMMI**) han tenido bastantes críticas en el momento de aplicarlos en pymes. Por ejemplo, se ha afirmado en el ámbito de estas empresas que el modelo **CMMI** genera una documentación excesiva, interfiere con la creatividad, tiene costos muy altos y es demasiado grande y complejo (Turgeon, 2006). La introducción del concepto de **SPI** en pequeñas compañías puede no ser posible, porque en estas empresas la madurez de la organización puede ser baja. Por tanto, es fundamental comprender las necesidades comerciales y prácticas de las pymes para aumentar la relevancia y los beneficios de las prácticas **CMMI** aplicadas a ellas (Chen y Staples, 2007). A pesar de la importancia de modelos como **CMMI**, en general, se ha observado que la implementación exitosa de estos no es posible en el contexto de las pymes (ISO/IEC 29110-1, 2010).

Con recursos limitados, las pequeñas empresas generalmente necesitan asistencia externa en la planificación e implementación de mejoras de procesos para mantenerse actualizadas en la investigación de vanguardia en ingeniería de software. Además, los estándares internacionales de ciclos de vida, como ISO/IEC 12207 e ISO/IEC 15288, no abordan explícitamente las necesidades de las organizaciones pequeñas (Oktaba *et al.*, 2007).

► 10.1. Pymes desarrolladoras de software

El término *pyme* se refiere a una categoría de empresa que no es esencialmente una gran organización. Su definición formal varía de acuerdo con la literatura, país e indicadores utilizados, además de la cantidad de empleados, total de activos, nivel de ventas e inversiones.

Para la Unión Europea (**UE**), los principales factores que determinan si una empresa es una pyme son plantilla de personal, volumen de negocios y balance total. En la tabla 10.1, se presentan las categorías de mipymes de acuerdo con la cantidad de empleados, facturación y balance total.

Además, no todas las empresas desarrolladoras de software son iguales, varían según factores como el tamaño, el sector del mercado, el tiempo en el negocio, el estilo de gestión, la gama de productos y la ubicación geográfica (O'Connor y Laporte, 2014). El contexto situacional de estas empresas incluye ocho factores de clasificación: personal, requisitos, aplicación, tecnología, organización, operación, gestión y negocios (Clarke y O'Connor, 2010).

Tabla 10.1. Categorización de pymes y mipymes

Categoría	Cantidad de empleados	Facturación	Balance total
Mediana	<250	<= 50 millones de euros	<= 43 millones de euros
Pequeña	<50	<= 10 millones de euros	<= 10 millones de euros
Micro	<10	<= 2 millones de euros	<= 2 millones de euros

Fuente: elaboración propia con base en European Commission (2020).

Las pymes son fundamentales para la economía de la mayoría de los países. A nivel mundial, constituyen el segmento principal de la industria de desarrollo de software (Pino *et al.*, 2008). Las pymes desarrolladoras de software desempeñan un papel importante en la industria del software, al caracterizarse por ser innovadoras, utilizar nuevas tecnologías, crear oportunidades de trabajo y mantenerse como empresas firmemente establecidas (Baskerville y Pries-Heje, 1999; Chin, 2000; Vahaniitty y Rautiainen, 2005; Wheelen *et al.*, 2017).

La estructura organizacional de estas empresas a menudo es informal, los procesos definidos implícitamente evolucionan en función del trabajo diario y las acciones pueden no planificarse de antemano (Järvi *et al.*). El trabajo se basa en la confianza, por tanto, a menudo no se realiza una documentación formal y existe una falta de conocimiento sobre la cultura del proceso. Además, a menudo hay falta de recursos, habilidades, experiencia y personal calificado y motivado para realizar actividades **SPI**. Puede ser muy difícil para estas empresas establecer una cultura de proceso eficiente y competitiva, así como gestionar el crecimiento de forma simultánea (Savolainen *et al.*, 2007).

En la tabla 10.2, se presenta un listado de las características principales de las pymes desarrolladoras de software.

Además, este tipo de empresas presenta alta dependencia de los individuos, pequeño número de empleados con diferentes actividades a cargo, gran impacto del factor humano, pequeño número de proyectos, alta comunicación con los clientes y dificultad de invertir en actividades **SPI** (Horvat *et al.*, 2000).

Tabla 10.2. Características de las pymes desarrolladoras de software

Característica	Descripción
Ciclo de vida del software simplificado	Se definen solo algunas fases principales en las que el desarrollo y las pruebas son las más importantes. A menudo, las pruebas se acortan para cumplir con los plazos o enfrentar la falta de recursos.
Los procesos presentan niveles de calidad muy desiguales	Se pueden combinar muy buenos procesos con niveles de muy bajo nivel. Las prácticas de alto nivel de calidad son a menudo contractuales de relación cliente-proveedor y se consideran una carga en lugar de un activo.
Procedimientos de control muy poco formalizados	En vista de la falta de control, la motivación para respetar los nuevos procedimientos puede desaparecer rápidamente. La falta de control puede considerarse la causa básica detrás de la calidad desigual del proceso.
Prácticas de gestión de proyecto desiguales	Algunas empresas tienen una muy buena gestión de proyectos, mientras otras no. Las prácticas pueden ser muy desiguales según el proyecto, el cliente, el gerente del proyecto o el equipo de desarrollo.
Recursos limitados para capacitación y recursos humanos	Limitaciones debido a las graves restricciones presupuestarias. Las actividades de capacitación se realizan para responder a deficiencias técnicas puntuales.
Nivel de servicio a los clientes muy desigual	Los clientes internos generalmente reciben un servicio mucho peor que los externos, para quienes se llevan a cabo prácticas más rigurosas.
Estructuras pequeñas orientadas a proyectos	Los procesos rara vez son impulsados por una estrategia a largo plazo. Por tanto, las prácticas de aprendizaje y gestión del conocimiento son escasas.
Sin enfoques metodológicos	Las empresas tienen que seguir las pautas metodológicas de sus clientes (más grandes). La flexibilidad se convierte en una preocupación vital.
Comunicación muy informal	Es una ventaja en comparación con estructuras más grandes en las que la comunicación burocrática puede ser lenta e ineficiente. Sin embargo, es necesario un mínimo de formalización para garantizar la calidad del proceso.

Fuente: elaboración propia.

► 10.2. SPI en pymes de software

Las pymes pueden tener una baja prioridad del proceso de desarrollo de software, ya que se centran en la calidad del producto y el tiempo de entrega en lugar de la calidad del proceso (Basri y O'Connor, 2010).

Debido a los recursos limitados y el tamaño de las organizaciones, una extensa evaluación de las prácticas de software siguiendo enfoques complejos no se considera necesaria o apropiada en ese contexto (Sánchez-Gordón *et al.*, 2016). Además, algunos estudios (por ejemplo, Coleman y O'Connor, 2008; Clarke y O'Connor, 2012a) sugieren que las pymes no adoptan un enfoque proactivo y altamente prioritario para la mejora de procesos de software. Debido a la gran variedad de configuraciones de desarrollo de software, la implementación de un conjunto de prácticas para el desarrollo de software puede ser muy diferente de una configuración a otra (Jeners *et al.*, 2013).

Estas empresas no utilizan un modelo de proceso de software como un libro de texto, prefieren seleccionar algunos elementos de un modelo determinado o realizar un desarrollo que mejor se adapte a sus necesidades específicas. Las prácticas de trabajo se eligen de manera oportuna y son adaptadas y configuradas para proporcionar valor de acuerdo con las restricciones impuestas por su contexto.

Los siguientes elementos pueden influir en el éxito de la adopción de **SPI** en las pymes (Pino *et al.*, 2008):

- ▶ Contratación de asesoramiento de expertos sobre **SPI**.
- ▶ Apoyo financiero para financiar la mejora del proceso.
- ▶ Cooperación entre organizaciones interesadas en **SPI** para que puedan compartir los recursos empleados.
- ▶ Estrategias para obtener el apoyo de los equipos de la organización.
- ▶ Realización de análisis de brechas.
- ▶ Establecimiento e institucionalización de un plan de comunicación considerando a todos los *stakeholders*.
- ▶ Compromiso de la alta dirección y un fuerte compromiso de todos los *stakeholders*.

Los estudios de la literatura e industria se han concentrado en los requisitos específicos de las pymes y han dado como resultado el desarrollo de una serie de estándares, infraestructuras, modelos, métodos y técnicas para la mejora de procesos centrados en diferentes aspectos y con objetivos diferentes. En la figura 10.1, se presentan las diferentes relaciones entre esos elementos.

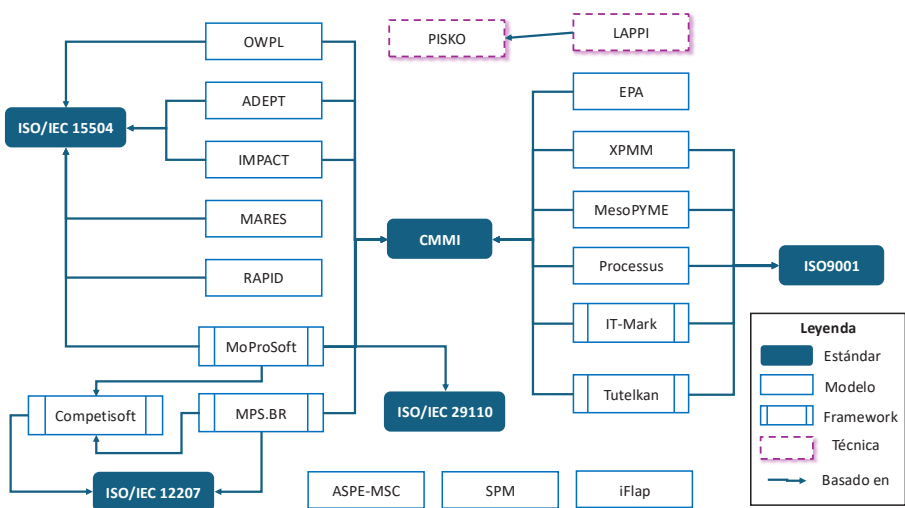


Figura 10.1. Relaciones entre estándares, infraestructura, modelos y técnicas de SPI en pymes

Fuente: elaboración propia con base en Sánchez-Gordón *et al.* (2016).

► 10.3. Estándares internacionales para SPI en pymes

El estándar más completo, desarrollado y promovido para pymes es **ISO/IEC 29110** “Perfiles del ciclo de vida para pequeñas organizaciones”, en que una muy pequeña entidad (Very Small Entity [VSE]) es “una empresa, organización, departamento o proyecto que tiene hasta 25 personas”. El estándar consta de cinco partes (**ISO/IEC 29110-1**, 2010), como se muestra en la figura 10.2, y está basado en procesos y resultados de otros estándares, como **ISO/IEC 12207** y productos de **ISO/IEC 15289**.

El estándar puede ser utilizado para diferentes propósitos y por diferentes tipos de organizaciones:

- **Para la implementación y mejora de procesos:** En proyectos nuevos o existentes con beneficios para facilitar la comunicación entre partes, mejorar la satisfacción del cliente y la productividad y la calidad.

- ▶ *Para la evaluación de procesos:* Los resultados de evaluación de procesos se pueden utilizar para la mejora de los procesos, la evaluación de riesgos de los procesos existentes o la evaluación comparativa.
- ▶ *Para la certificación:* El uso de la evaluación de los procesos sirve para tener credibilidad y respeto internacional y tener una ventaja competitiva, al mismo tiempo que mantiene a los clientes actuales y atrae a otros nuevos.

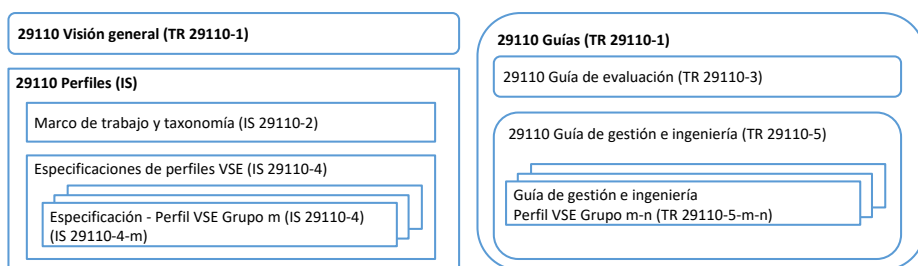


Figura 10.2. Conjunto de documentos iso/iec 29110

Fuente: elaboración propia con base en ISO/IEC 29110-1 (2010).

El enfoque utilizado para desarrollar el modelo **ISO/IEC 29110** consiste de tres pasos (O'Connor y Laporte 2014):

1. Seleccionar el subconjunto de procesos **ISO/IEC 12207** aplicable a muy pequeñas entidades de menos de 25 empleados.
2. Adaptar el subconjunto de procesos para satisfacer las necesidades de muy pequeñas entidades.
3. Desarrollar pautas para muy pequeñas entidades.

La parte 1, “Visión general (**ISO/IEC 29110-1**)”, explica los principales conceptos, términos y estructura del estándar. La parte 2, “Marco de trabajo y taxonomía (**ISO/IEC 29110-2**)”, presenta los principios y el mecanismo para construir perfiles de entidades muy pequeñas. La parte 3, “Guía de evaluación (**ISO/IEC 29110-3**)”, define las pautas de evaluación del proceso y los requisitos de cumplimiento necesarios para cumplir con el propósito de los perfiles **VSE** definidos. La parte 4, “Especificaciones de los perfiles **VSE** (**ISO/IEC 29110-4**)”, proporciona el mapeo de los estándares de origen y es útil para los desarrolladores y evaluadores de métodos. La parte 5, “Guía de gestión e ingeniería (**ISO/IEC 29110-5**)”, está destinada a las **VSE**.

Los requisitos básicos de un proceso de desarrollo de software deben ajustarse a las necesidades de los proyectos. La característica más importante de las entidades en el estándar es el tamaño. Sin embargo, existen otros aspectos y características que pueden afectar la selección del perfil, como modelos de negocios (bajo contrato, interno o comercial), factores situacionales (por ejemplo, la criticidad, el entorno de incertidumbre, etc.) y niveles de riesgo. La tabla 10.3 ilustra un grupo de perfiles que contiene tres perfiles (etiquetados A, B y C) que se asignan a nueve combinaciones de modelos de negocio y factores situacionales. Un requisito durante el desarrollo del estándar fue que los procesos deberían evaluarse utilizando el enfoque **ISO/IEC 15504**.

Tabla 10.3. Asignación de características de pymes a grupos de perfiles

Factores situacionales de los perfiles			
Modelos de negocio	<i>Crítico</i>	<i>Bajo incertidumbre</i>	<i>Cambio ambiental</i>
Bajo contrato	Perfil A	Perfil A	Perfil A
Interno	Perfil C	Perfil B	Perfil A
Comercial	Perfil B	Perfil A	Perfil A

Fuente: elaboración propia con base en O'Connor (2014).

Un concepto clave del estándar radica en el desarrollo de varios grupos de perfiles genéricos de entidades. Los grupos de perfiles son una colección de estos que están relacionados ya sea por composición de procesos (es decir, actividades o tareas), nivel de capacidad o ambos. El grupo de perfil genérico es aplicable a una gran mayoría de entidades que no desarrollan software crítico y tienen factores situacionales típicos. Dentro de ese grupo de perfil genérico, se propusieron cuatro perfiles. Al utilizarlos, las empresas muy pequeñas tienen la oportunidad de mejorar sus procesos de manera clara y gradual.

- *Básico*: Destinado a un proyecto único sin riesgos especiales o factores situacionales. Permite establecer buenas prácticas en el proceso de gestión de proyectos y en el proceso de ingeniería de software.

- ▶ *Inicial/entrada*: Como el perfil básico es difícil de implementar para algunas empresas, se propuso este perfil. Se aplica a emprendimientos o pequeñas organizaciones con proyectos de seis personas/mes. La empresa no tiene definido ningún proceso de software.
- ▶ *Intermedio*: Destinado a pequeñas empresas que están desarrollando más de un proyecto a la vez y son conscientes de asignar recursos, monitorear proyectos para lograr los objetivos de negocio y la satisfacción del cliente, así como definir, implementar y mejorar los procesos estándar de la organización para lograr resultados similares en todos los proyectos.
- ▶ *Avanzado*: Admite entidades con prácticas de gestión empresarial. Las empresas cuentan con procesos de software definidos, sus artefactos están basados en estándares y las buenas prácticas forman parte de la cultura organizacional.

El perfil básico detalla dos procesos interrelacionados con sus actividades y tareas, roles involucrados y productos del trabajo. Los procesos del perfil básico se pueden observar en la figura 10.3. Los dos procesos que componen este perfil son:

- ▶ *Proceso de gestión de proyectos*: Para establecer y llevar a cabo de manera sistemática las tareas del proyecto de implementación de software, lo que permitirá cumplir con los objetivos del proyecto en calidad, tiempo y costo esperados.
- ▶ *Proceso de implementación de software*: Produce un software específico implementado como un producto o servicio realizando las actividades de análisis, diseño, construcción, integración y prueba, de acuerdo con los requisitos especificados.

Para ayudar con el despliegue y la implementación del estándar, se han desarrollado una serie de paquetes y guías de implementación para definir pautas y explicar en detalle los procesos definidos en los perfiles.

Un conjunto de paquetes de implementación es un conjunto de artefactos desarrollados para facilitar la implementación de un conjunto de prácticas de un perfil seleccionado. Los elementos de un paquete típico son descripción de procesos, actividades, tareas, roles y productos, plantillas, listas de verificación, ejemplos, referencias y mapeo a estándares y modelos, y una lista de herramientas. La tabla de contenido del paquete de implementación de gestión de proyectos se muestra en la figura 10.4. Además, se han desarrollado una serie de guías de implementación para ayudar a implementar un proceso específico respaldado por una herramienta.

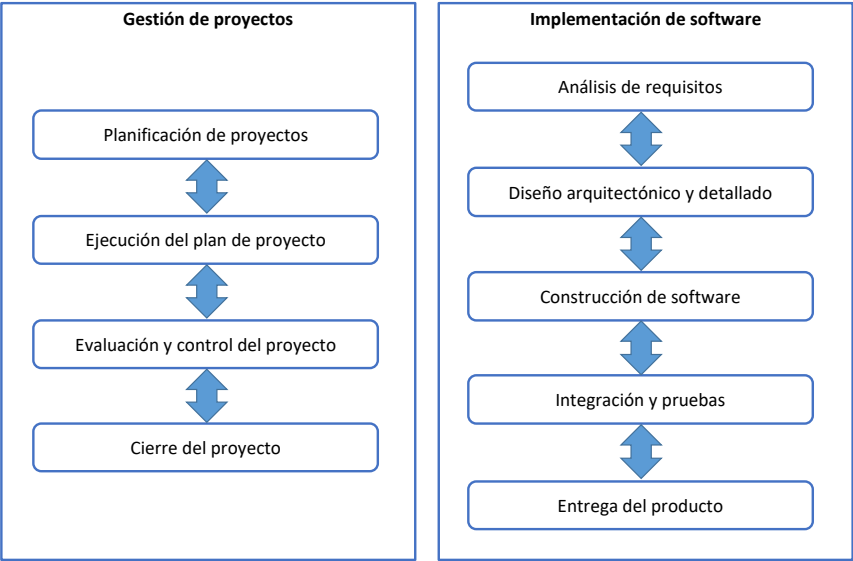


Figura 10.3. Diagramas de procesos del perfil básico de ISO/IEC 29110

Fuente: elaboración propia con base en O'Connor (2014).

1. Descripción técnica

- Importancia de la gestión de proyectos
 - Éxitos y fracasos de la gestión de proyectos

2. Definiciones (definiciones genéricas y específicas)

3. Relaciones con ISO/IEC 29110

- Proceso de gestión de proyectos
 - Tareas y roles

4. Descripción detallada

- Roles, productos y artefactos

5. Plantillas

- WBS, plantilla de estado del proyecto, etc.

6. Ejemplos

- Prácticas de ciclo de vida de gestión de proyectos, etc.

7. Listas de verificación

- Lista de verificación de revisión del plan del proyecto, etc.

8. Herramientas

9. Referencia a otras normas y modelos

- ISO 9000, ISO/IEC 12207 y CMMI-DEV

10. Referencias

11. Formulario de evaluación del paquete de despliegue

Figura 10.4. Contenido de un paquete de despliegue de gestión de proyectos

Fuente: elaboración propia con base en O'Connor (2014).

► 10.4. Infraestructuras para SPI en pymes

De acuerdo con la norma **ISO/IEC 42010: 2007**, “una infraestructura establece una práctica común para crear, organizar, interpretar y analizar descripciones arquitectónicas utilizadas dentro de un dominio particular de aplicación o comunidad de partes interesadas” (**ISO/IEC 42010, 2007**). A continuación, se describen diversas infraestructuras propuestas, desarrolladas y orientadas a la mejora del proceso en pymes. Se destacan MoProSoft, Competisoft, **MPS.BR**, ITmark y Tutelkan.

10.4.1. Infraestructura MoProSoft

El modelo de procesos para la industria del software (MoProSoft) es un modelo para la mejora y evaluación de los procesos de desarrollo y mantenimiento de productos de la industria del software en México desarrollado en 2003. Su propósito es apoyar la introducción de las prácticas de ingeniería de software y de gestión en pymes (Oktaba, 2005). El modelo considera nueve procesos encerrados en tres categorías (figura 10.5):

- *Alta dirección*: Aborda las prácticas relativas a la gestión del negocio. Incluye un único proceso: gestión del negocio.
- *Gerencia intermedia*: Incluye los procesos de soporte para que el plan de negocios pueda desarrollarse y considera las alineaciones para usar y controlar los recursos de la organización de manera eficiente. Incluye los procesos de gestión de proyectos, gestión de procesos y gestión de recursos (con los subprocesos recursos humanos y ambiente de trabajo; bienes, servicios e infraestructura, y conocimiento de la organización).
- *Operaciones*: Abarca prácticas concretas de desarrollo de software. Incluye los procesos de desarrollo y mantenimiento de software, y gestión de proyectos específicos.

Cada uno de los procesos de MoProSoft se define de acuerdo con un patrón de proceso en el que se describen los objetivos, los recursos, las actividades y otros elementos necesarios para implementarlos.

MoProSoft es el modelo de referencia para un método de evaluación que permite medir el nivel de capacidad de los procesos implantados en las organizaciones, llamado EvalProSoft (Oktaba *et al.*, 2004) basado en las recomendaciones de **ISO/IEC 15504-2**. Los resultados ofrecen a una organización la evaluación de sus procesos que sirven como base para dirigir los

esfuerzos de mejora. El proceso de evaluación reúne las actividades de preparación, planificación, implementación, generación y entrega de resultados. Por consiguiente, EvalProSoft define el proceso de evaluación de acuerdo con un patrón dividido en tres secciones: definición general del proceso, prácticas y guías de ajuste (García-Mireles y Rodríguez-Castillo, 2009).

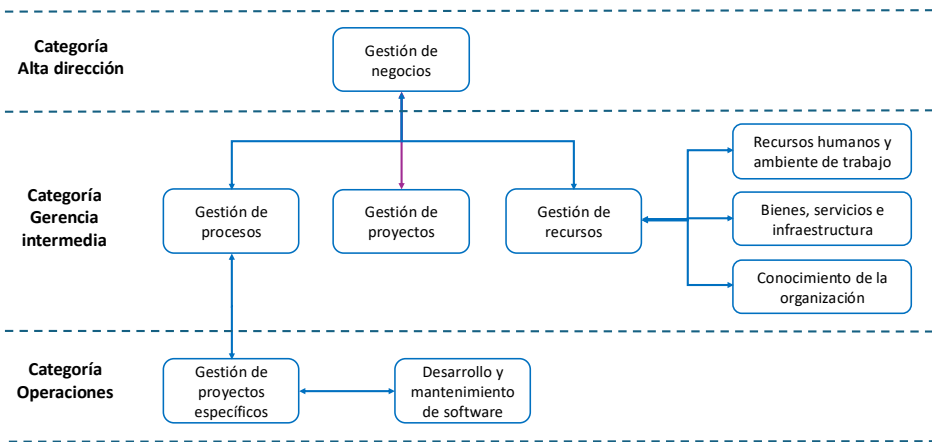


Figura 10.5. Categorías y procesos de MoProSoft

Fuente: elaboración propia con base en Oktaba (2005).

EvalProSoft se realizó de acuerdo con la norma **ISO/IEC 15504-2** y trabaja con:

- Seis niveles de capacidad de proceso de 0 a 5. La tabla 10.4 muestra los niveles y sus atributos de proceso que se utilizan para determinar si un proceso ha alcanzado un determinado nivel de capacidad. Cada atributo mide un aspecto particular del proceso.
- Un marco de medición que incluye la evaluación del atributo del proceso basado en cuatro niveles:
 - *No logrado*: Representa entre el 0% y el 15% del cumplimiento del propósito del proceso.
 - *Parcialmente logrado*: Representa entre el 15% y el 50% del cumplimiento del propósito del proceso.
 - *Mayoritariamente logrado*: Representa entre el 50% y el 85% del cumplimiento del propósito del proceso.

- *Completamente logrado*: Representa entre el 85 % y el 100 % del cumplimiento del propósito del proceso.
- Nueve atributos de proceso, tomando uno para el primer nivel y dos para cada uno de los siguientes niveles de capacidad.
- Indicadores para cada atributo utilizado en un proceso de evaluación.

Tabla 10.4. Niveles y atributos de procesos del modelo EvalProSoft

Nivel	Descripción	Áreas de proceso
Nivel 0: Incompleto	Significa que no alcanza el propósito del proceso.	
Nivel 1: Realizado	El proceso implementado logra su propósito y obtiene los resultados definidos.	1.1. Realización del proceso
Nivel 2: Gestionado	El proceso realizado es implementado y los productos de trabajo son apropiadamente establecidos, controlados y mantenidos.	2.1. Gestión de la realización 2.2. Gestión de productos de trabajo
Nivel 3: Establecido	El proceso de gestión se implementa a través del proceso definido, capaz de lograr el resultado del proceso.	3.1. Definición del proceso 3.2. Implementación del proceso
Nivel 4: Predecible	El proceso establecido opera dentro de ciertos límites para lograr sus resultados.	4.1. Proceso de medición 4.2. Control de proceso
Nivel 5: Optimizado	El proceso predecible se mejora continuamente para lograr los objetivos comerciales actuales y futuros.	5.1. Proceso de innovación 5.2. Optimización de procesos

Fuente: elaboración propia con base en Oktaba (2005).

El patrón de proceso MoProSoft consiste en una definición general del proceso, guías de adaptación y una sección de prácticas (Oktaba, 2005). Las organizaciones utilizan el patrón para documentar el proceso.

- *La definición general del proceso*: Incluye el nombre del proceso, la categoría y el propósito; un resumen de las actividades del proceso; objetivos e indicadores de objetivos; responsabilidades y roles de autoridad; subprocesos y procesos relacionados; entradas, salidas y productos internos, y referencias bibliográficas.
- *Las guías de adaptación*: Sugieren las posibles modificaciones del proceso que no deberían afectar el logro de los objetivos del proceso.

- *La sección de prácticas:* Incluye prácticas recomendadas de capacitación, manejo de situaciones excepcionales, uso de lecciones aprendidas y un diagrama de actividades Unified Modeling Language (UML).

En agosto de 2005, México aprobó MoProSoft y EvalProSoft como el estándar **NMX-059-NYCE-2005**.

10.4.2 Infraestructura Competisoft

Competisoft es una iniciativa de investigadores de Iberoamérica financiada por el Programa Iberoamericano de Ciencia y Tecnología para el desarrollo (CYTED) que se desarrolló entre 2006 y 2008. El objetivo del proyecto fue incrementar el nivel de competitividad de las pymes iberoamericanas productoras de software mediante la creación y difusión de un marco metodológico común que, ajustado a sus necesidades específicas, pueda llegar a ser la base sobre la cual establecer un mecanismo de evaluación y certificación de la industria de software (López *et al.*, 2012). El modelo de Competisoft incluye los siguientes elementos (figura 10.6):

- *Modelo de referencia de procesos:* Está basado en MoProSoft con sus tres categorías de procesos: alta dirección, gerencia intermedia y operaciones.
- *Modelo de evaluación:* Se basa en el modelo EvalProSoft y define cinco niveles de madurez para la organización: realizado, gestionado, establecido, predecible y optimizado. El modelo de evaluación soporta tres usos: evaluación para la acreditación de capacidades, evaluación de capacidades de un proveedor y autoevaluación de capacidades de un proceso. Las medidas de evaluación se agrupan en dos tipos principales:
 - *Medidas de capacidad:* Utilizan indicadores de atributos de proceso para evaluar la capacidad del proceso (del nivel 1 al 5) sobre la base de prácticas genéricas, recursos y productos de trabajo.
 - *Medidas de rendimiento:* Se basan en el propósito, la descripción, los productos de trabajo y las actividades del modelo de referencia de Competisoft.
- *Modelo de mejora:* El modelo define PmCompetisoft, un proceso de mejora que sigue el patrón de proceso definido en Competisoft y está basado en Agile SPI, el cual establece los elementos necesarios

para ejecutar económicamente programas de mejora en pequeñas organizaciones en cinco fases (Piattini y Garzás-Parra, 2007):

- *Instalación:* Se crea una propuesta de mejora a partir de las necesidades del negocio, la cual debe ser aprobada por la alta dirección. La propuesta incluye objetivos de mejora e infraestructura de gestión con un equipo de gestión, un equipo de tecnología de procesos y un equipo de mejora.
- *Diagnóstico:* Se realizan actividades de valoración de los procesos de software para conocer su estado actual. Como resultado del análisis del proceso actual, se priorizan los casos de mejora para crear el plan general de mejora.
- *Formulación:* Se seleccionan los casos de mejora prioritarios para planificar la primera iteración de mejora.
- *Mejora:* Los casos de mejora son ejecutados. Además, se realizan planificaciones de las próximas iteraciones de mejora.
- *Revisión del programa:* Se realiza un análisis *post mortem* del ciclo de mejora ejecutado. Se recolectan métricas y lecciones aprendidas como insumos para los próximos ciclos de mejora.

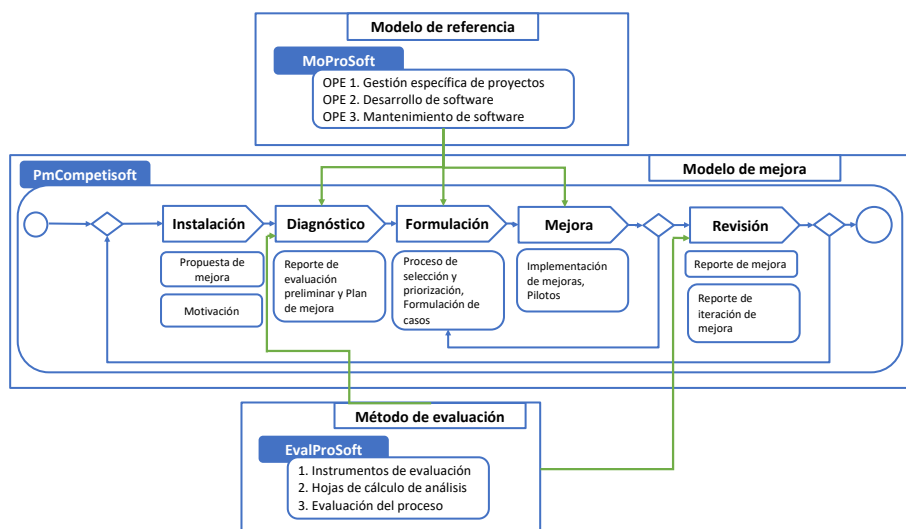


Figura 10.6. Marco de trabajo metodológico de Competisoft

Fuente: elaboración propia con base en Villarroel *et al.* (2009).

El modelo de mejora incluye un conjunto de disciplinas que puede ser aplicado en mayor o menor medida a cada fase. Las disciplinas definidas para el modelo son gestión del programa, evaluación, análisis de resultados, diseño, implementación, gestión de configuración, aprendizaje y entrenamiento.

10.4.3. Infraestructura MPS.BR

El modelo mejora de procesos de software brasileño (**MPS.BR**) fue propuesto en 2003 por la Asociación para la Promoción de la Excelencia Brasileña del Software (Softex), una organización privada sin ánimo de lucro cuyo objetivo es promover la competitividad de la industria brasileña de software. **MPS.BR** tiene como objetivo definir e implementar un modelo para la mejora de procesos orientado a las pymes y mipymes, de tal manera que estas obtengan un nivel de madurez 2 o 3 de **CMMI** a un costo accesible (Weber *et al.*, 2005).

MPS.BR cubre diferentes perfiles de empresas con distintos tamaños y características, aunque presta especial atención a las pymes y mipymes (**MPS.BR**, 2012). Centrándose en las pymes y mipymes de software, el modelo **MPS.BR** está diseñado para ser compatible con los estándares de calidad internacionalmente aceptados.

El modelo consta de varios modelos de referencia, como se observa en la figura 10.7 (Santos *et al.*, 2012; Softex, 2020):

- **MPS-SW**, *modelo MPS de software*: Se basa en los requisitos de proceso definidos en los modelos de mejora de procesos y satisface la necesidad de implementar los principios de ingeniería de software de una manera apropiada al contexto de las empresas, según los principales enfoques internacionales para definir, evaluar y mejorar los procesos de software. El modelo está documentado en forma de tres guías:
 - *Guía general de MPS*: Proporciona una definición general del modelo **MPS** y definiciones comunes a todas las demás guías.
 - *Guía de adquisición de MPS*: Describe un proceso de adquisición de software y servicios relacionados.
 - *Guía de implementación de MPS*: Proporciona orientación técnica para implementar el modelo. Además, describe los conceptos teóricos para fundamentar los procesos definidos en los niveles de madurez.

- ▶ **MPS-SV**, *modelo MPS para servicios*: Se desarrolló para complementar el modelo **MPS-SW** tanto para apoyar la mejora de los procesos de servicio como para ofrecer un proceso de evaluación que acredite el cumplimiento de las prácticas de la organización en relación con las mejores prácticas del sector.
- ▶ **MPS-RH**, *modelo MPS para gestión de personal*: Se basa en los requisitos de gestión de personas de los modelos y estándares que se centran en definir, evaluar y mejorar esta gestión, ofreciendo a las compañías directrices para la implementación gradual de prácticas de gestión de recursos humanos en la industria de las tecnologías de la información y de la comunicación (**TIC**). Está orientado a seleccionar, desarrollar y retener el talento humano para cumplir con los objetivos de la organización.
- ▶ **MA-MPS**, *método de evaluación*: Su objetivo es verificar la madurez de una unidad organizacional en la ejecución de sus procesos de software. Describe los requisitos para la acreditación de organizaciones a fin de proporcionar servicios de evaluación de **MPS**, evaluadores competentes y provisionales de **MPS**, así como describe el proceso de evaluación definido para apoyar la aplicación del método **MA-MPS**.
- ▶ **MN-MPS**, *modelo de negocio*: Describe las reglas de negocio. Su objetivo es mejorar el compromiso de todos los participantes mediante la relación definida como un triángulo de servicios entre la empresa (programa **MPS.BR**), los colaboradores de primera línea (responsables de apoyar las implementaciones de **MPS** o realizar las evaluaciones) y los clientes (organizaciones que adoptaron el modelo de **MPS**).

MR-MPS y **MA-MPS** son compatibles con **ISO/IEC 12207**, **ISO/IEC 15504** y **CM-MI-DEV**. **MPS.BR** define siete niveles de madurez del proceso para las empresas de desarrollo de software: A (en optimización), B (gestionado cuantitativamente), C (definido), D (definido en gran medida), E (definido parcialmente), F (gestionado) y G (parcialmente gestionado). La capacidad del proceso está representada por un conjunto de atributos del proceso descritos en términos de resultados esperados y expresa el nivel de refinamiento e institucionalización del proceso que se ejecuta en la organización (**MPS.BR**, 2012).

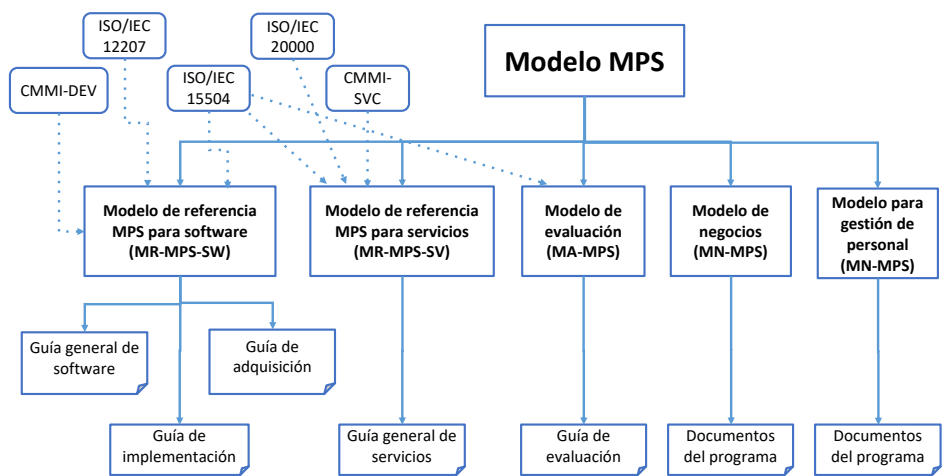


Figura 10.7. Componentes del modelo MPS

Fuente: elaboración propia con base en MPS.BR (2012).

La tabla 10.5 muestra la estructura de los niveles de madurez **MR-MPS** que se definen en dos dimensiones (según **ISO/IEC 15504**): dimensión del proceso y dimensión de la capacidad del proceso.

El nivel de madurez inicial de **MR-MPS** es el nivel G constituido por los dos procesos de software más críticos para las pymes: gestión de requisitos y gestión de proyectos. Los niveles A y B son los niveles de madurez **MR-MPS** más altos que se centran en la mejora continua del proceso (Santos *et al.*, 2012). El enfoque en las pymes hace importante la definición de más niveles de madurez (en especial, el perfil básico del nivel G) que **CMMI**.

Para cada uno de los siete niveles de madurez, se asigna un conjunto de procesos, y para cada proceso, se establece su propósito y los resultados esperados. Además, para cada nivel de madurez, los resultados esperados de los atributos del proceso se establecen definiendo la capacidad de los procesos (Hauck *et al.*, 2015).

Tabla 10.5. Estructura de los niveles de madurez del modelo MR-MPS

Nivel de madurez	Procesos	Atributos del proceso
A	No se agregan nuevos procesos	1.1, 2.1, 2.2, 3.1, 3.2, 4.1*, 4.2*, 5.1*, 5.2*
B	Gestión de proyectos (nuevos resultados)	1.1, 2.1, 2.2, 3.1, 3.2, 4.1*, 4.2*
C	Gestión de decisiones, gestión de riesgos y desarrollo para reutilización	1.1, 2.1, 2.2, 3.1, 3.2
D	Desarrollo de requisitos, diseño y construcción de productos, integración de productos, verificación y validación	1.1, 2.1, 2.2, 3.1, 3.2
E	Gestión de recursos humanos, establecimiento de procesos, evaluación y mejora de procesos, gestión de proyectos (nuevos resultados) y gestión de reutilización	1.1, 2.1, 2.2, 3.1, 3.2
F	Medición, gestión de configuración, adquisición, control de calidad y gestión de cartera de proyectos	1.1, 2.1, 2.2
G	Gestión de requisitos y gestión de proyectos	1.1, 2.1

* Son aplicables solo en los procesos seleccionados. Los otros deben aplicarse en todos los procesos.

Fuente: elaboración propia con base en Santos *et al.* (2012).

10.4.4. Infraestructura ITmark

ITmark es un esquema de certificación diseñado para pymes de tecnologías de la información (TI) desarrollado por el European Software Institute (ESI). Engloba varios modelos de mejora en un único esquema de certificación. Se evalúan y certifican los procesos de las pymes en tres áreas principales (ITmark, 2020): gestión del negocio, ingeniería de software, sistemas y servicios, y gestión de seguridad.

El modelo busca que las pymes establezcan un proceso de mejora continua totalmente compatible con el modelo **CMMI**. Distingue tres niveles posibles de madurez en los procesos de cada pyme (Valencia *et al.*, 2009):

- *IT Mark*: Acredita a una empresa que es consciente de los problemas relacionados con la gestión técnica, de seguridad y del negocio, y que los mantiene habitualmente bajo control. Admite que

algunas de las áreas de proceso puedan no estar suficientemente elaboradas, trabajando con los procesos técnicos de **CMMI** nivel 2.

- *IT Mark premium*: Acredita a una empresa que ha conseguido una buena madurez en sus procesos de trabajo técnico, de seguridad y de negocio.
- *IT Mark élite*: Acredita a una empresa que ha conseguido un nivel superior en la definición e institucionalización de sus procesos de trabajo técnico, de seguridad y de negocio.

Aunque el modelo está basado en **CMMI**, no se ha reducido para aplicarlo más fácilmente en las pymes, sino que se aplica adecuando sus requisitos a las características de estas y reduciendo su grado de exigencia. En la figura 10.8, se presenta la equivalencia de los niveles de ITmark con **CMMI-DEV**.

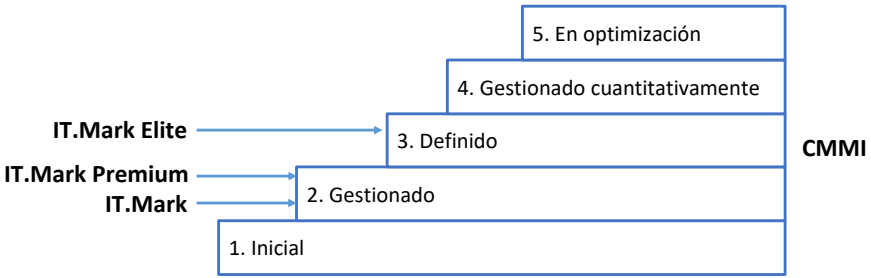


Figura. 10.8. Equivalencia de niveles ITmark con **CMMI**

Fuente: elaboración propia con base en ITmark (2020).

10.4.5. Infraestructura Tutelkan

Tutelkan es un esfuerzo conjunto de la academia chilena y la industria local del software financiado por la Agencia Chilena de Cooperación Internacional para el Desarrollo (**AGCID**). Su objetivo general es crear un mecanismo sostenible que permita a las pequeñas organizaciones de software definir, implementar, institucionalizar y mejorar sus procesos de desarrollo, incorporando las mejores prácticas internacionales (Valdés *et al.*, 2011).

Las ideas principales para lograr ese objetivo son: a) proporcionar una biblioteca de activos de procesos reutilizables, b) ofrecer herramientas de composición para describir los procesos de pequeñas organizaciones que

utilizan estos activos y c) capacitar sistemáticamente a consultores enfocados en pequeñas organizaciones para estas bibliotecas y conjuntos de herramientas.

Tutelkan contiene un repositorio de activos de proceso reutilizables que cumplen con los modelos de calidad. Cualquier organización puede reutilizarlos para elaborar y mantener su propio conjunto de activos de proceso. Cada activo reutilizable contiene información sobre las prácticas **CMMI** específicas y las cláusulas **ISO 9001** a las que se ajusta (figura 10.9).

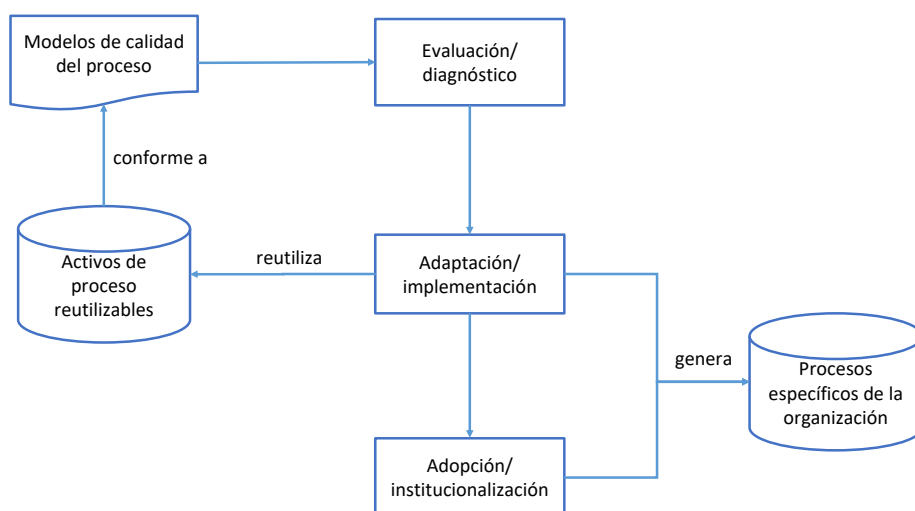


Figura 10.9. Proceso Tutelkan

Fuente: elaboración propia con base en Valdés *et al.* (2011).

Tutelkan incluye varios elementos (Valdés *et al.*, 2011):

- ▶ *El proceso de referencia de Tutelkan:* Define un modelo de proceso de software específico cuyos elementos y activos individuales pueden reutilizarse y modificarse para crear nuevos procesos de software. Su versión inicial está alineada con los modelos **CMMI** (nivel de madurez 2 y 3), **ISO 9001: 2000** y Competisoft.
- ▶ *La plataforma web de Tutelkan:* Permite describir, acceder y componer los activos de proceso. Su arquitectura basada en servicios incluye herramientas web para explorar y construir colaborativamente activos de proceso de software públicos y privados.

- *El marco de procesos de Tutelkan*: Un metamodelo para procesos de software que define los tipos y las relaciones válidos de activos y procesos.
- *El proceso de implementación de Tutelkan*: Una metodología que facilita la implementación de procesos de software en pequeñas organizaciones. Especifica cómo llevar a cabo un proyecto SPI utilizando los contenidos y las herramientas de Tutelkan.

El proceso de referencia de Tutelkan (Tutelkan Reference Process [TRP]) se compone de 23 áreas de proceso agrupadas en siete categorías, que a su vez se agrupan en cuatro partes (tabla 10.6).

Tabla 10.6. Componentes del proceso de referencia de Tutelkan

Parte	Categoría	Áreas de proceso
TRP básico	Gestión de proyectos	Gestión de requisitos, planificación de proyectos, monitoreo y control de proyectos, gestión de acuerdos con proveedores, medición y análisis, gestión de calidad del producto y proceso, gestión de configuración.
	Desarrollo y mantenimiento de software	Desarrollo de requisitos, análisis y diseño, programación, pruebas, instalación.
TRP avanzado	Gestión avanzada de proyectos	Gestión de riesgos, evaluación de decisiones formales, gestión integrada de proyectos.
	Gestión de proceso	Definición del proceso organizacional, mejora del proceso organizacional, formación organizacional.
TRP de alta madurez	Gestión de procesos de alta madurez	Rendimiento del proceso organizacional, innovación y despliegue organizacional, análisis y resolución causal.
	Gestión de proyectos de alta madurez	Gestión cuantitativa de proyectos.
TRP de negocios	Gestión del negocio	Planificación estratégica.

Fuente: elaboración propia con base en Valdés *et al.* (2010).

El **TRP** básico contiene prácticas recomendadas para garantizar que los proyectos se gestionen de manera efectiva. El **TRP** avanzado se enfoca en asegurar que la organización tenga un proceso de desarrollo de software

que se defina, mantenga y mejore continuamente. El **TRP** de alta madurez se preocupa por incorporar análisis cuantitativos a la gestión de procesos y proyectos. El **TRP** de alta madurez es un superconjunto de **TRP** avanzado, y este último es un superconjunto de **TRP** básico. El **TRP** de negocios se centra en la capacidad de gestión estratégica de la organización y es más independiente, pero se recomienda que se implemente en paralelo con el **TRP** de alta madurez (Valdés *et al.*, 2011).

► 10.5. Modelos y métodos para SPI en pymes

Entre los principales modelos/métodos para realizar proyectos o iniciativas de mejora de procesos de software en pymes se encuentran: **OWPL**, **MARES**, **EPA**, Adept, Impact, MesoPyme, **ASPE/MS**, iFlap, Processus, **SPM**, **RAPID** y **XPMM**. Estos modelos generalmente definen una colección de etapas/tareas organizadas o clasificadas en procesos, fases o categorías.

10.5.1. Modelo OWPL

OWPL se centra en simplificar las prácticas de **CMMI** e **ISO 15504** para que sean aplicables a las pymes (Stambollian *et al.*, 2006). Su estructura involucra procesos, prácticas y factores de éxito (figura 10.10).

El modelo define 10 procesos: gestión de requisitos, planificación de proyectos, seguimiento y supervisión de proyectos, desarrollo, documentación, pruebas, gestión de configuración, gestión de subcontratistas, gestión de calidad y capitalización de experiencia. Cada uno de estos procesos está descompuesto en una serie de 3 a 12 prácticas.

A cada uno de los procesos se le asigna una meta general de acuerdo con los objetivos definidos de la organización. Cada proceso implica una serie de prácticas y está respaldado por varios factores de éxito. Cada práctica se define por su objetivo, sus entradas y salidas, los recursos asignados para apoyarla y su peso, y un indicador de la importancia de la práctica para la mejora del proceso (Alexandre *et al.*, 2006).

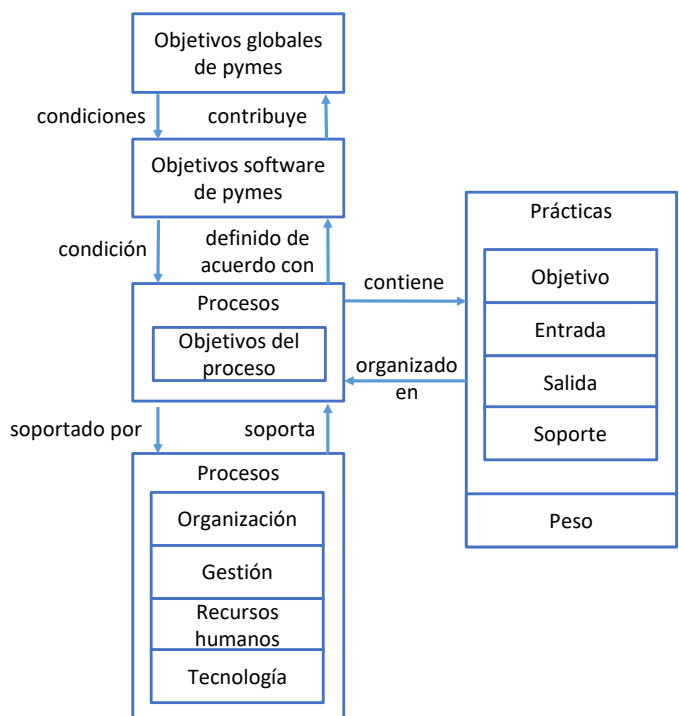


Figura 10.10. Estructura de OWPL.

Fuente: elaboración propia con base en Alexandre *et al.* (2006).

El modelo **OWPL** propone un enfoque gradual basado en un marco de mejora de procesos de software de tres etapas, como se observa en la figura 10.11 (Alexandre *et al.*, 2006).

- **Micro-evaluación:** Se utiliza un cuestionario muy simplificado para recopilar información sobre las prácticas actuales de software en estructuras pequeñas. El cuestionario cubre seis ejes: garantía de calidad, gestión de proyectos, gestión de clientes, gestión de productos, gestión de subcontratistas y gestión de recursos humanos.
- **Evaluación OWPL:** La información recopilada se utiliza como punto de partida para determinar los objetivos de una evaluación más precisa de acuerdo con el modelo OWPL.
- **Evaluación SPICE:** Las empresas más grandes con un nivel de calidad medio/alto son eventualmente invitadas a realizar una evaluación ISO/IEC 15504 o CMMI.

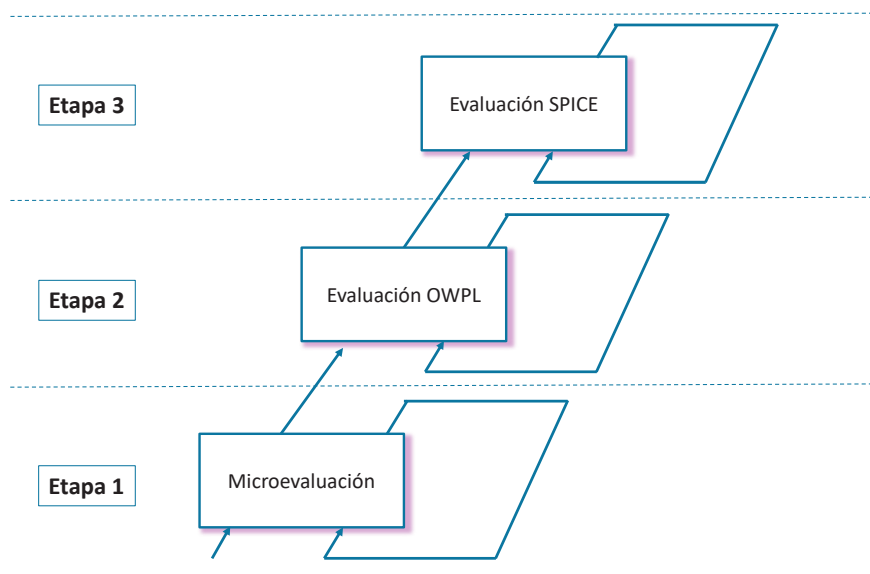


Figura 10.11. Mejora de procesos en OWPL

Fuente: elaboración propia con base en Alexandre *et al.* (2006).

10.5.2. Modelo MARES

La metodología para la evaluación de procesos de software en pequeñas empresas de software (Methodology for Software Process Assessment in Small Software Companies [**MARES**]) fue desarrollada por investigadores de la Universidade do Vale do Itajaí (Univali) y el Centro de Pesquisas Renato Archer (CenPRA) en Brasil. **MARES** está diseñado para apoyar la mejora de procesos en el contexto de pequeñas organizaciones de software considerando sus características y limitaciones específicas. Fue construido según la norma ISO 15504. Integra un modelo de proceso para respaldar la selección de procesos relevantes y la identificación de potenciales riesgos y sugerencias de mejora (Zarour *et al.*, 2011).

El método de evaluación **MARES** se divide en cinco partes principales, como se observa en la figura 10.12 (Von Wangenheim *et al.*, 2006; Zarour *et al.*, 2011):

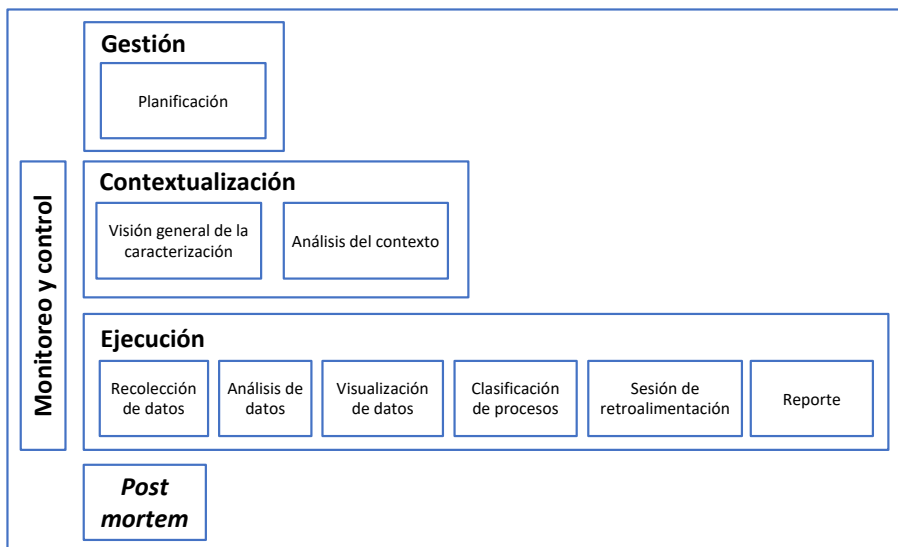


Figura 10.12. Visión general del proceso de evaluación MARES

Fuente: elaboración propia con base en Anacleto *et al.* (2004).

- **Planificación:** La evaluación se organiza, planifica y finaliza con un plan de evaluación revisado y documentado que contiene propósito, alcance, recursos, responsabilidades, cronograma e infraestructura.
- **Contextualización:** Se caracteriza la organización para comprender sus objetivos, productos, objetivos de negocio y su proceso de software. Los cuestionarios y las entrevistas se utilizan como un medio para recopilar datos. Se analizan los datos utilizando una descripción del modelo de procesos de alto nivel, perfiles de proceso objetivo y se seleccionan procesos para la evaluación.
- **Ejecución:** Los procesos seleccionados se evalúan en detalle. Se elaboran resultados y reportes de la evaluación.
- **Monitoreo y control:** Las actividades son monitoreadas y controladas. Si es necesario, las acciones correctivas se inician y se actualiza el plan.
- **Post mortem:** Se realiza una breve sesión entre los evaluadores para analizar, discutir y documentar el desempeño de la evaluación.

10.5.3. Modelo EPA

El método evaluación express del proceso (Express Process Appraisal [EPA]) es un método de evaluación de **CMMI** de clase C desarrollado en 2003 por el Centre for Software Process Technologies (**CSPT**) en la Universidad de Ulster en Irlanda del Norte. La figura 10.13 muestra las ocho etapas del método **EPA**. Consiste en un conjunto de preguntas surgidas desde **CMMI** para garantizar una cobertura adecuada del modelo durante las sesiones de preguntas (Wilkie *et al.*, 2007).

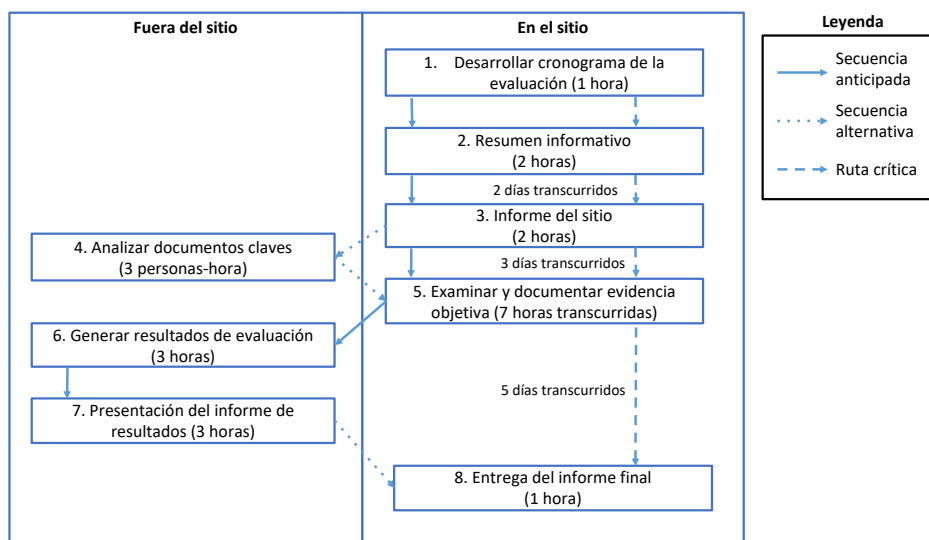


Figura 10.13. Etapas del método EPA

Fuente: elaboración propia con base en Wilkie *et al.* (2007).

El método **EPA** evalúa 17 de las 21 áreas de proceso de **CMMI**. La evaluación generalmente cubre de cinco a siete áreas de proceso (Lester *et al.*, 2007). A continuación, se realiza una descripción de cada etapa del modelo (McCaffery *et al.*, 2005):

- *Etapas 1: Desarrollar el cronograma de evaluación.* Es una reunión preliminar para establecer la logística y determinar el cronograma. Involucra a dos empleados de **CSPT** y al menos un representante de la empresa.

- *Etapas 2: Resumen informativo.* El evaluador principal proporciona una visión general del método. Involucra a dos empleados de **CSPT** y un promedio de siete empleados de la compañía.
- *Etapas 3: Informe del sitio.* La empresa evaluada explica los elementos estructurales de la organización al equipo de evaluación. Esta sesión informativa involucra a dos empleados del **CSPT** y a dos empleados de la empresa en promedio.
- *Etapas 4: Analizar documentos clave.* Se solicitan cinco muestras de documentos: un plan de proyecto típico, un informe de progreso de un proyecto típico, una declaración de requisitos aprobada típica, pautas/manuales de garantía de calidad de la compañía y cualquier documentación relacionada con la política de la compañía sobre la gestión de configuración. Esta etapa involucra a dos miembros del personal del **CSPT** y generalmente a un miembro del personal de la compañía.
- *Etapas 5: Examinar y documentar evidencia objetiva.* Se entrevista a miembros clave del personal de la organización evaluada. Hay seis entrevistas y cada entrevista está programada para durar aproximadamente una hora. Cada entrevista se enfoca en una de las seis áreas del proceso. Las entrevistas involucran a un equipo de evaluación compuesto por dos empleados de **CSPT** y al menos un representante de la compañía.
- *Etapas 6: Generar resultados de evaluación y Etapas 7: Crear informe final.* El informe final consiste en una lista de fortalezas, problemas y acciones sugeridos para cada una de las áreas de proceso evaluadas. Las etapas 6 y 7 involucran a dos miembros del equipo de evaluación **CSPT** que colaboran juntos durante tres horas.
- *Etapas 8: Presentación del informe de resultados:* Consiste en presentar el informe de resultados al grupo de personas de la organización evaluada que participaron en las entrevistas. Esta presentación involucra a dos empleados de **CSPT** y un promedio de siete empleados de la compañía.

Para cada área de proceso, se desarrolló un conjunto de preguntas básicas. Estas son un conjunto definido de preguntas utilizadas para obtener información sobre la práctica **CMMI** relevante a los entrevistados. El método **EPA** contiene múltiples preguntas para cada práctica **CMMI** (figura 10.14).

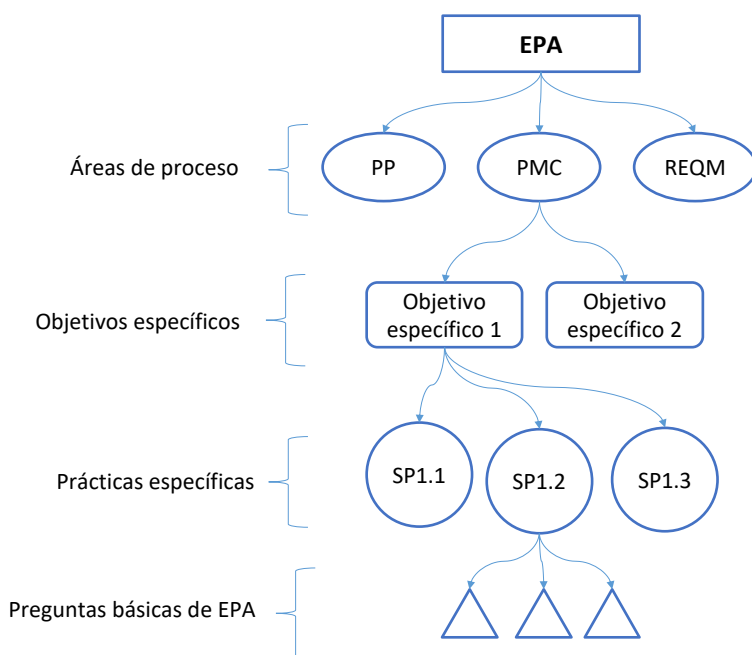


Figura 10.14. Organización de preguntas básicas del método EPA

Fuente: elaboración propia con base en Lester *et al.* (2007).

Los problemas abordados en las preguntas básicas reflejan las prácticas relevantes en **CMMI**. Las preguntas iniciales tienden a ser exploratorias y conducen a preguntas más específicas y directas. El modelo no proporciona ninguna forma de calificación; por el contrario, una lista de fortalezas, debilidades y acciones sugeridas para cada área de proceso (Lester *et al.*, 2010).

10.5.4. Modelo Adept

El modelo Adept fue desarrollado por Fergal McCaffery en 2006 para proporcionar una evaluación ligera de áreas de proceso, y así alentar a las pymes irlandesas a mejorar sus prácticas de desarrollo de software (McCaffery *et al.*, 2006).

Adept fue desarrollado en atención a los siguientes aspectos (McCaffery y Coleman, 2009):

- La mejora es más importante que la certificación.

- La cantidad de tiempo de preparación requerida por la empresa para la evaluación debe ser mínima.
- La evaluación debe realizarse en un corto periodo.
- El método de evaluación debe permitir a las empresas seleccionar las áreas de proceso que sean más relevantes para sus objetivos de negocio.
- La evaluación se basa en **CMMI** e **ISO/IEC 15504**.

El método se aplica en las siguientes 12 áreas de proceso: gestión de requisitos, gestión de configuración, planificación de proyectos, monitoreo y control de proyectos, medición y análisis, aseguramiento de la calidad del proceso y del producto, gestión de riesgos, solución técnica, verificación, validación, desarrollo de requisitos e integración de productos. Cualquiera de estas 12 áreas de proceso puede evaluarse utilizando el método Adept. Sin embargo, cuatro se clasifican como obligatorios: gestión de requisitos, gestión de configuración, planificación de proyectos, y monitoreo y control de proyectos. Los pasos de ejecución del método utilizan los mismos pasos que el método **EPA** descritos en el apartado anterior (McCaffery y Coleman, 2009).

10.5.5. Modelo Impact

El modelo Impact es un marco de trabajo práctico desarrollado en la Universidad de Nueva Gales del Sur (Australia) diseñado para respaldar una **SPI** rápida, iterativa y eficaz en pymes aprovechando al máximo las tecnologías existentes. El modelo Impact está basado en el modelo **IDEAL** de mejora del proceso de software.

El modelo Impact diferencia dos niveles: nivel del proceso y nivel del proyecto (Scott *et al.*, 2001), como se observa en la figura 10.15.

- *Nivel de proyecto*: Los proyectos se realizan con buenas prácticas de gestión a través de un ciclo del proyecto, por ejemplo, el ciclo pequeño planificar-hacer-verificar-actuar de la figura 10.15.
- *Nivel de proceso*: La experiencia y la comprensión de muchos proyectos se utilizan para comprender y mejorar el modelo de proceso genérico, que luego se utiliza para guiar proyectos futuros, por ejemplo, el ciclo grande de la figura 10.15 denominado ciclo del proceso.

El modelo Impact contiene las siguientes etapas (Scott *et al.*, 2001):

- ▶ *Entender*: Consiste en comprender los procesos existentes que proporcionan la base para la futura mejora, la ejecución de un proceso definido e identificar posibles áreas de mejora.
- ▶ *Modelo de proceso*: Se refiere a los modelos de los procesos realizados en los proyectos. Se utilizan para guiar proyectos y se mejoran con el tiempo. El modelo de proceso se documenta utilizando un lenguaje y herramientas de modelado de procesos adecuados.
- ▶ *Mejora*: Se analiza el modelo de proceso, se identifican las posibilidades de mejora y luego se realiza la mejora del modelo de proceso.
- ▶ *Guía de proceso*: La guía del proceso facilita la aplicación del proceso en proyectos. Las tecnologías web ofrecen la capacidad de búsquedas y navegación fáciles, la adición de plantillas de ejemplos, enlaces a herramientas y facilidad de actualización de contenidos.
- ▶ *Medir*: Los proyectos son monitoreados a través de medidas que registran los efectos de las mejoras.

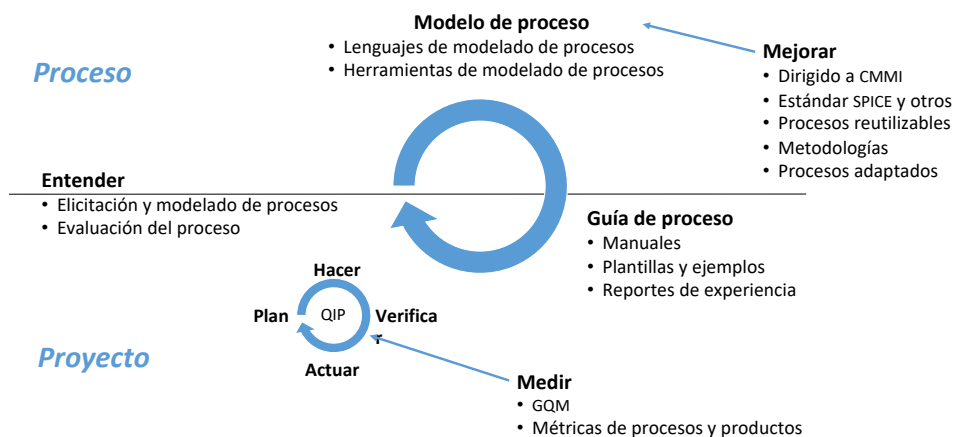


Figura 10.15. Modelo Impact

Fuente: elaboración propia con base en Scott *et al.* (2001).

El ciclo del proceso impulsa la ejecución de los proyectos y estos a su vez mejoras en el proceso.

10.5.6. Modelo MESOPYME

MESOPYME es un modelo **SPI** genérico con cuatro etapas similares a los del modelo **IDEAL**, como se observa en la figura 10.16 (Calvo-Manzano *et al.*, 2002).

- *Etapas 1: Compromiso con la mejora del proceso de software.* Se obtiene el apoyo de la alta gerencia para llevar a cabo el proyecto de mejora.
- *Etapas 2: Evaluación del proceso de software.* Se obtienen fortalezas y debilidades del proceso evaluado con respecto a un modelo de proceso de software, por ejemplo, **CMMI**.
- *Etapas 3: Infraestructura y planes de mejora del proceso de software.* Proporciona la infraestructura necesaria para llevar a cabo mejoras en los procesos seleccionados y crear un plan a seguir para definir e implementar mejoras en estos.
- *Etapas 4: Implementación de la mejora del proceso de software.* Sus objetivos son definir los nuevos procesos elegidos (siguiendo el plan anterior) e implementarlos en proyectos piloto. Finalmente, la mejora debe ser institucionalizada.

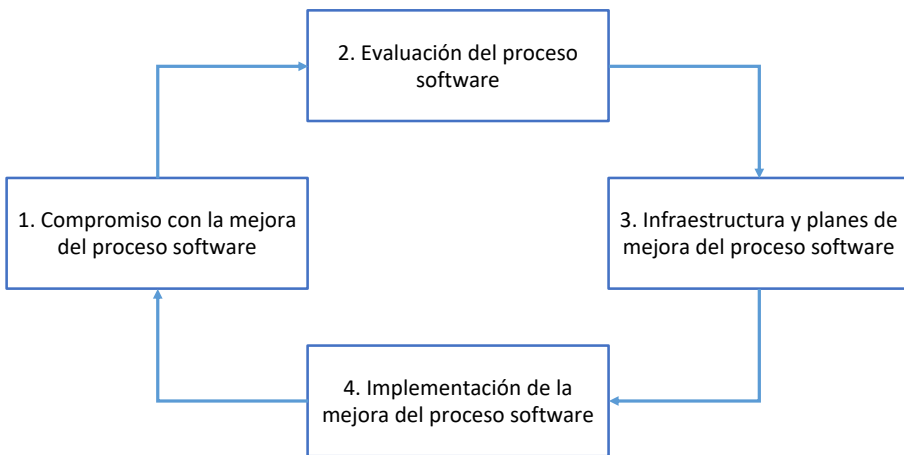


Figura 10.16. Modelo genérico de SPI

Fuente: elaboración propia con base en Calvo-Manzano *et al.* (2002).

La tercera etapa se realiza mediante la aplicación de un conjunto genérico de componentes denominado paquete de acción. Este es una solución general para un área de proceso de software particular que debe

personalizarse para una empresa en atención a sus objetivos de negocio y los resultados de la evaluación.

La arquitectura de cada paquete de acción se representa gráficamente en la figura 10.17. A continuación, se realiza una breve descripción de los componentes de un paquete de acción (Calvo-Manzano *et al.*, 2002):

- ▶ *Plan de acción*: Tiene un plan genérico para el proyecto de mejora. Se describen las actividades, los resultados y el borrador del cronograma.
- ▶ *Política*: Son los principios que rigen las operaciones de la empresa con respecto al dominio de mejora. En cada paquete, se describe una guía de la política con su propósito, objetivos y responsabilidades de las diferentes estructuras organizacionales.
- ▶ *Infraestructura*: Estructura y roles organizacionales específicos en la implementación de las mejoras propuestas.
- ▶ *Proceso*: Contiene las etapas y actividades relacionadas con el proceso, los documentos o las entradas utilizadas y las salidas generadas, los roles de los participantes para cada actividad y las técnicas utilizadas para llevar a cabo cada etapa del proceso con éxito.
- ▶ *Productos y documentos*: La estructura global del paquete se divide en productos y documentos para la administración general (reuniones y actas), dominio del objetivo y normas/guías.
- ▶ *Técnicas*: Se aplicarán en las actividades descritas en el proceso, tales como herramientas, métricas, aseguramiento de la calidad del software, entrenamiento y logros tempranos.

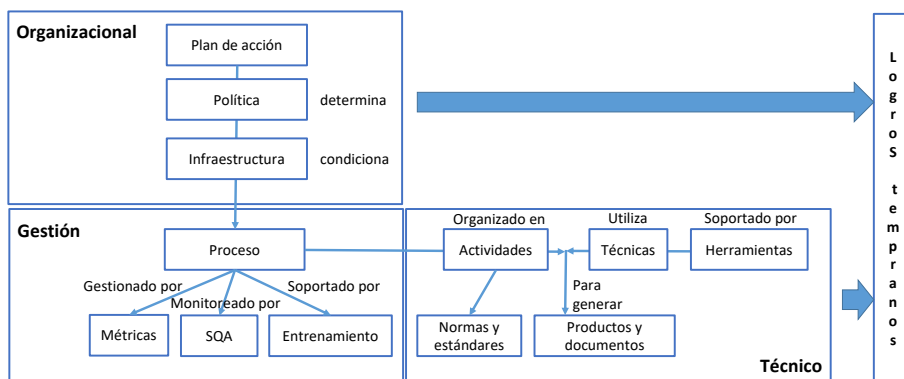


Figura 10.17. Arquitectura de un paquete de acción

Fuente: elaboración propia con base en Calvo-Manzano *et al.* (2002).

10.5.7 Modelo ASPE/MSc

El enfoque para el establecimiento de procesos de software en mypes (Approach for Software Process Establishment in Micro and Small Companies [ASPE/MSc]) se desarrolló en Brasil en 2005. El propósito del enfoque es apoyar un establecimiento rentable y efectivo de procesos de software como parte de una iniciativa **SPI** en atención a la adaptación de los enfoques existentes al contexto de las pequeñas empresas de software. Las fases principales del enfoque se ejecutan de forma iterativa e incremental, y se presentan la figura 10.18. Además, el enfoque cubre la gestión del establecimiento de los procesos de software junto con la planificación, el seguimiento, el control y el *post mortem* (Von Wangenheim *et al.*, 2006).

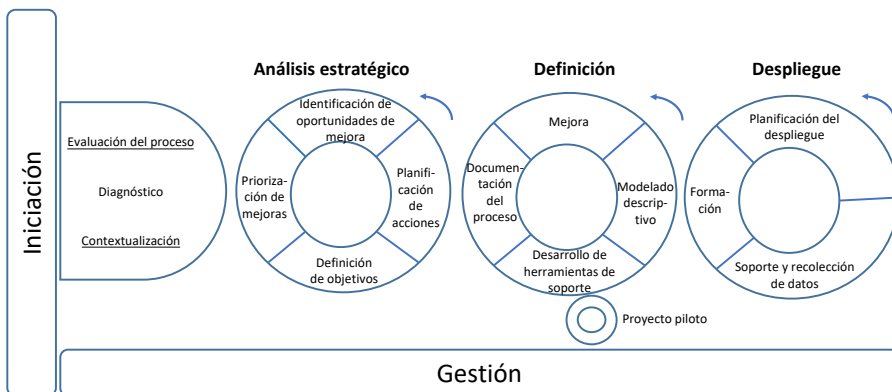


Figura 10.18. Definición del proceso ASPE/MSc

Fuente: elaboración propia con base en Hauck *et al.* (2008).

Las cinco fases del enfoque **ASPE/MSc** son (Hauck *et al.*, 2008; Von Wangenheim *et al.*, 2006):

- **Instanciación:** Se prepara la iniciativa **SPI** y se crea la infraestructura necesaria.
- **Diagnóstico:** Se realiza una evaluación del proceso de software real cuyos resultados pueden incluir:
 - Una descripción del contexto de la organización.
 - Una descripción de alto nivel del proceso de software real que indica qué procesos se ejecutan y cómo.

- Perfiles de capacidad del proceso objetivo que indican procesos de alta prioridad y su nivel de capacidad a lograr para cumplir objetivos de negocio.
- Perfiles de capacidad de proceso evaluados con fortalezas y debilidades de los procesos de software identificados, incluso riesgos potenciales y oportunidades de mejora.
- ▶ *Análisis estratégico:* Se especifica el alcance de los procesos que se establecerán y mejorarán. Se priorizan los procesos de acuerdo con los objetivos de negocio. Se definen los ciclos de mejora y los procesos a mejorar en cada ciclo.
- ▶ *Definición:* Cada proceso elegido se modela, mejora y documenta en forma de una guía de proceso organizacional para apoyar a los usuarios del proceso. Una vez que se ha definido el proceso, las herramientas utilizadas durante la ejecución del proceso deben adaptarse o desarrollarse de acuerdo con la definición del proceso.
- ▶ *Implementación:* El proceso estándar que se definió se aplica y evalúa en proyectos piloto y luego durante la implementación se institucionaliza en la organización. Incluye la definición de medidas para determinar la efectividad de los procesos.

10.5.8. Modelo iFlap

El marco de mejora utilizando evaluación de peso ligero y planificación de mejora (Improvement Framework Utilizing Light Weight Assessment and Improvement Planning [iFlap]) es una guía paso a paso desarrollada en Suecia para la evaluación de procesos y la planificación de la mejora. Las principales etapas del marco iFlap (Pettersson *et al.*, 2008) se presentan en la figura 10.19.

- ▶ *Etapas 1: Selección.* Se eligen proyectos a estudiar que sean representativos de la población de proyectos actuales. Luego, se seleccionan los roles que participan en las actividades o que se ven afectados por los productos resultantes del proceso que se está evaluando. Finalmente, se identifican las personas que puedan representar cada rol. Esta etapa puede ser realizada por evaluadores que tengan conocimiento sobre el modelo de negocio, el dominio, los productos, los clientes, las principales actividades realizadas, los roles, los proyectos y el vocabulario.

- *Etapla 2: Evaluación.* Se ocupa de la obtención de información a través de entrevistas y estudio de documentos, el análisis de los datos obtenidos y la triangulación de los resultados. La evaluación consta de dos partes:
 - *Un estudio de proyecto:* Escrutinio de uno o más proyectos a través de entrevistas y un análisis de la documentación del proyecto.
 - *Un estudio de línea:* Examina las partes relevantes de la organización que no forman parte de un proyecto en particular. Consiste en entrevistas y un análisis de la documentación del proceso.
- *Etapla 3: Planificación de la mejora:* Implica las siguientes actividades:
 - *Priorización:* Se elige qué se debe mejorar. Para la priorización, se pueden utilizar métodos, como:
 - *Proceso de jerarquía analítica* (Saaty y Vargas, 2001): Se realizan comparaciones por pares entre todos los elementos que se priorizarán.
 - *Votación acumulativa* (Berander y Andrews, 2005): Una cantidad predefinida de puntos, a menudo denotada como dinero, se distribuye entre los elementos.
 - *El enfoque de los diez mejores y el ranking* (Berander y Andrews, 2005): No proporcionan resultados en la escala de ratio, solo dan un orden relativo de elementos de la mejora.
 - *Identificación de dependencias entre problemas de mejora:* El orden establecido al priorizar los problemas de mejora no es necesariamente el orden en que las mejoras se implementan mejor. Los participantes identifican dependencias entre los problemas de mejora para reconocer las restricciones prácticas y establecer una forma más práctica de lograr la mejora.
 - *Análisis de datos:* Se puede realizar utilizando técnicas, como:
 - *Gráficos de desacuerdo* (Regnell et al., 2001): Para visualizar la variación de prioridad entre los participantes individuales.
 - *Gráficos de satisfacción* (Regnell et al., 2001): Ilustran cómo se compara la clasificación de prioridad de cada partici-

pante o rol individual con la clasificación resultante de los problemas de mejora para todo el grupo.

- *Índice de consistencia* (Saaty y Vargas, 2001): Indica la cantidad de comparaciones contradictorias.
- *Análisis de componentes principales* (Kachigan, 1986): Para analizar si los grupos identificables intuitivamente explican las diferencias de opinión.
- *Empaquetamiento*: Consiste en empaquetar los problemas de mejora para guiar la planificación y la implementación. El tamaño y el contenido de cada uno de los paquetes se determinan a partir de las prioridades y dependencias entre los problemas de mejora, los efectos de las soluciones candidatas y el costo de implementación. El tiempo y los recursos necesarios para implementar las mejoras del proceso influyen en el tamaño de cada paquete y son factores decisivos para elegir qué implementar primero.

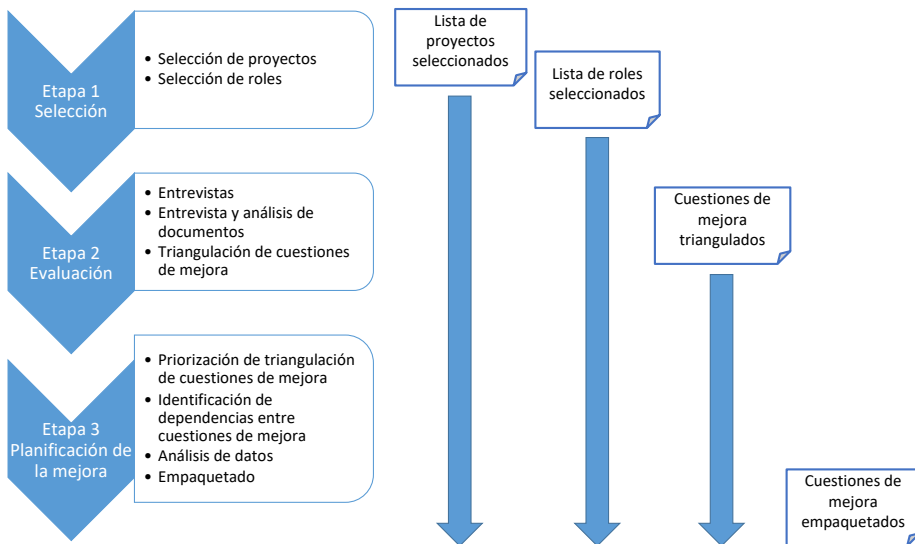


Figura 10.19. Visión general del método iFlap

Fuente: elaboración propia con base en Pettersson *et al.* (2008).

10.5.9. Modelo Processus

Processus (evaluación y mejora de la calidad de los procesos de software) es un proyecto iniciado en 1994 y financiado por el Ministerio de Ciencia y Tecnología de Eslovenia y 13 empresas eslovenas (Rozman *et al.*, 1997). Processus se deriva de los requisitos y características de las normas **ISO 9001**, **ISO 9000-3** y **CMM**. La figura 10.20 presenta el marco del modelo. En el primer círculo, se definen cinco pasos, que deben realizarse durante la introducción de cada procedimiento (Horvat *et al.*, 2000):

- **Análisis:** Su objetivo es definir el estado actual de las actividades en la empresa. Se puede realizar como una evaluación externa o interna de un área específica o se pueden definir los problemas y los problemas críticos.
- **Definición:** Según los resultados del análisis y el conocimiento del área descrita, los procedimientos se definen. El procedimiento resultante debe discutirse con el personal que realiza las actividades del procedimiento.
- **Formación:** Los procedimientos definidos de trabajo y los cambios incorporados en el procedimiento mejorado deben presentarse al personal.
- **Promulgación:** Los procedimientos definidos se utilizan en algunos proyectos piloto para establecer la rutina requerida y realizar las tareas y actividades involucradas.
- **Seguimiento:** Para analizar la aplicabilidad, eficiencia y aceptación de los procedimientos definidos. Si se requieren cambios importantes, se debe repetir todo el círculo. De lo contrario, se debe comenzar el análisis del siguiente procedimiento.

Las fases del modelo son las siguientes (Horvat *et al.*, 2000):

- **Fase de introducción:** Su propósito es preparar al personal para las actividades **SPI** y definir el marco de la estructura organizacional. Incluye los siguientes grupos de actividades: asignación y capacitación del gerente de calidad, definición del plan **SPI**, definición de la estructura organizacional, definición de la estructura de la documentación del proceso, introducción de conceptos de **SPI** al personal y definición de algunas métricas simples para evaluar la mejora durante el proyecto **SPI**.
- **Fase de definición del proceso:** Se divide en cuatro subfases:

- *Gestión de la relación con el cliente*: Incluye los procedimientos con gran influencia en la satisfacción del usuario final: gestión de contratos, gestión de requisitos, entrega del producto y mantenimiento.
 - *Gestión de proyectos*: Incluye las actividades para preparar el plan del proyecto con énfasis especial en actividades de gestión de calidad, revisiones de entrada y salida de fases del proyecto, revisiones de hitos particulares, etc.
 - *Ingeniería de software*: Incluye la definición e introducción de todos los procedimientos para la ingeniería de software (como análisis, diseño, implementación y pruebas) y la gestión de configuración.
 - *Actividades de soporte*: Se definen aquellos procedimientos que no están directamente relacionados con el desarrollo de un producto de software, pero son necesarios para asegurar la calidad de las actividades. Los procedimientos incluidos son formación, control de documentos y gestión de productos incluidos. También se puede incluir *marketing*.
- *Fase de optimización del proceso*: Su objetivo principal es proporcionar una mejora continua del proceso de software. Para ello, se realizan revisiones de los procedimientos existentes, la definición de su mejora y la automatización de esos procedimientos. Incluye dos subfases:
- *Gestión de procesos*: Incluye la definición de los procedimientos métricas y gestión de procesos (revisiones internas y acciones correctivas).
 - *Automatización de procesos*: Incluye el desarrollo de aplicaciones para admitir algunos procedimientos críticos o utilizar software colaborativo disponible u otras herramientas similares.

La definición de los procedimientos requiere un conjunto de documentos necesarios. El modelo utiliza la estructura de documentos sugerida por la International Organization for Standardization (ISO), u Organización Internacional de Normalización (Horvat *et al.*, 2000):

- *Manual de calidad*: Documento general del sistema de calidad en la empresa.
- *Procedimientos estándar*: Documentos que establecen una descripción detallada de las actividades realizadas dentro del procedimiento.

- *Documentos estándar*: Formularios, plantillas y manuales utilizados para realizar y documentar las entradas y los resultados de los procedimientos.

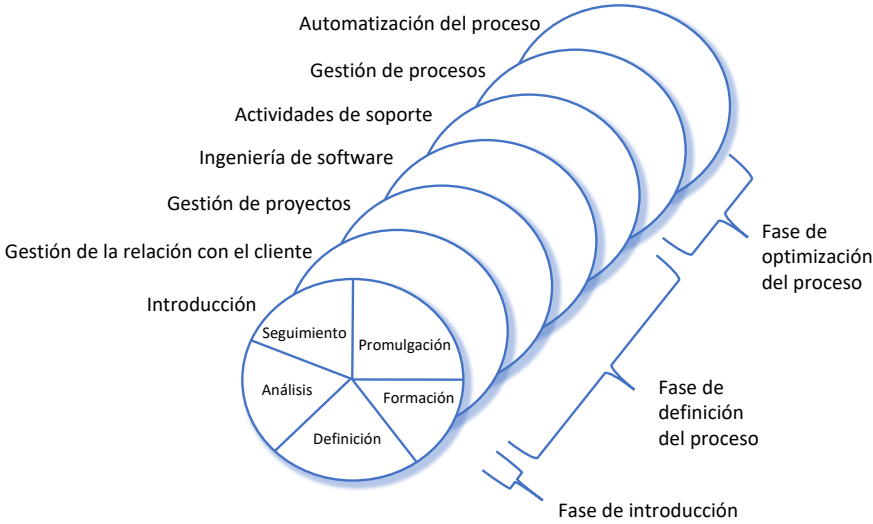


Figura 10.20. Modelo Processus para SPI en pymes

Fuente: elaboración propia con base en Horvat *et al.* (2000).

10.5.10. Matriz del SPM

La matriz del proceso de software (Software Process Matrix [SPM]) es un método SPI desarrollado en Irlanda basado en el despliegue de la función de calidad (Quality Function Deployment [QFD]) que puede ser utilizado en pequeñas empresas de desarrollo de software en la derivación de sus estrategias de SPI (Richardson y Ryan, 2001).

QFD es un método japonés utilizado principalmente en manufactura que se centra en mantener la voz del cliente durante todo el ciclo de vida de producción. Utiliza una serie de matrices para convertir las necesidades del cliente en un producto final. Las matrices QFD conforman un modelo de cuatro fases: matriz de planificación de productos (casa de calidad), despliegue de partes, planificación de procesos y planificación de producción. SPM está basado en la primera matriz de QFD denominada casa de calidad (Richardson, 2002).

El uso de esta matriz inicia con la recolección de las necesidades del cliente midiendo la importancia relativa de cada requisito del cliente. Estos requisitos se utilizan para identificar las características de diseño que tienen mayor impacto en los requisitos del cliente. La matriz se divide en seis “habitaciones”, como se observa en la figura 10.21 (Richardson, 2002): requisitos del cliente (qué), características del diseño (cómo), importancia general de los requisitos del cliente, relaciones entre los requisitos del cliente y las características del diseño, importancia de las características del diseño e interrelaciones entre las características del diseño.

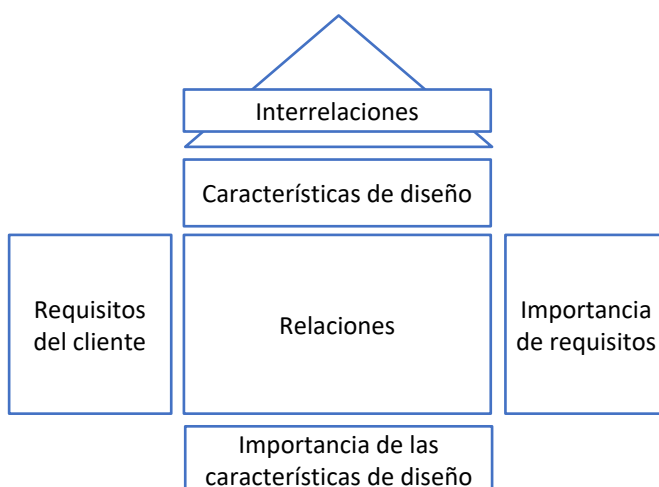


Figura 10.21. Estructura de la casa de calidad

Fuente: elaboración propia con base en Richardson y Ryan (2001).

En **SPM**, las características de diseño son las prácticas que deben seguirse para que los procesos tengan éxito. Algunos ejemplos de prácticas son probar la operación del cliente antes de la implementación del software, prototipo o simulación de elementos críticos del software y mantener y realizar seguimiento de historias de productos. **SPM** incluye un total de 135 prácticas. Una parte crucial del desarrollo de **SPM** fue identificar las relaciones entre procesos y prácticas. En la figura 10.22, se presenta un ejemplo de matriz **SPM** (Richardson y Ryan, 2001).

Las prácticas con los valores más altos son las más importantes y se establecen las prioridades que se incluirán en el plan de acción de mejora de procesos de software.



Figura 10.22. Ejemplo de matriz SPM

Fuente: elaboración propia con base en Richardson y Ryan (2001).

10.5.11. Modelo RAPID

El método de evaluaciones rápidas para la mejora de procesos para el desarrollo de software (Rapid Assessments for Process Improvement for software Development [RAPID]) es una propuesta australiana de mejora de procesos de software basado en ISO/IEC 15504 que involucró 24 organizaciones. Para evaluar un nivel de capacidad particular, los atributos del proceso relacionados con el nivel deben evaluarse como amplia o totalmente logrados, y los atributos en cualquier nivel inferior deben lograrse completamente. La tabla 10.7 muestra los atributos para cada nivel de capacidad (Cater-Steel, 2004).

Tabla 10.7. Niveles de capacidad y atributos del proceso

Nivel	Atributos del proceso
Nivel 0: Incompleto	1.1 Atributo de rendimiento del proceso
	2.1 Atributo de gestión del rendimiento
Nivel 1: Realizado	2.2 Atributo de gestión de productos de trabajo
	3.1 Atributo de definición y adaptación del proceso
Nivel 2: Administrado	3.2 Atributo de recursos del proceso
	4.1 Atributo de medición del proceso
Nivel 3: Establecido	4.2 Atributo de control de proceso
	5.1 Atributo de cambio del proceso
Nivel 4: Predecible	5.2 Atributo de mejora continua
Nivel 5: Optimizado	

Fuente: elaboración propia con base en Cater-Steel (2004).

El alcance de la evaluación incluye ocho procesos clave: elicitación de requisitos, desarrollo de software, gestión de configuración, aseguramiento de la calidad, resolución de problemas, gestión de proyectos, gestión de riesgos y establecimiento del proceso, como se observa en la tabla 10.8 (Cater-Steel *et al.*, 2006).

Tabla 10.8. Categoría de procesos en RAPID

	Proceso	Categoría de proceso	ISO/IEC TR 15504 ID
RE	Elicitación de requisitos	Cliente-proveedor	CUS.3
SD	Desarrollo de software	Ingeniería	ENG.1
CM	Gestión de configuración	Soporte	SUP.2
QA	Aseguramiento de la calidad	Soporte	SUP.3
PM	Gestión de proyectos	Gestión	MAN.2
PR	Resolución de problemas	Soporte	SUP.8
RM	Gestión de riesgos	Gestión	MAN.4
PE	Establecimiento del proceso	Organización	ORG.2.1

Fuente: elaboración propia con base en Cater-Steel *et al.* (2006).

Para el modelo de evaluación de **RAPID**, solo se incluyeron las preguntas relacionadas con los niveles 1-3, permitiendo niveles de calificación de nivel 0 (incompleto), nivel 1 (realizado), nivel 2 (gestionado) y nivel 3 (establecido). El método **RAPID** recopila evidencia solo por entrevistas, pero los participantes pueden ilustrar los temas en discusión referenciando los documentos. Además, incluye un conjunto de procedimientos y plantillas (Cater-Steel *et al.*, 2006).

10.5.12. Modelo **xPMM**

El modelo de madurez de programación extrema (eXtreme Programming Maturity Model [**xPMM**]) es un modelo polaco que adopta prácticas de programación extrema a los requisitos de los modelos basados en la **ISO** y se creó con base en las intersecciones entre eXtreme Programming (**xP**) y **CMMI-DEV** (Nawrocki *et al.*, 2002).

Los niveles del modelo se observan en la figura 10.23. El primer nivel, llamado “No cumple en absoluto”, significa que el equipo aplica algunas prácticas de **xP** o ninguna. El segundo nivel, “Inicial”, se centra en los equipos de proyecto y define dos áreas de proceso: gestión de la relación con el cliente y aseguramiento de la calidad del producto. El tercer nivel, “Avanzado”, se centra en la codificación y, por tanto, su única área de proceso es la programación por pares. El último nivel, denominado “Maduro”, tiene

una sola área de proceso, “Rendimiento del proyecto”, relacionada con la satisfacción de clientes. Para ser asignado a un nivel específico, el equipo tiene que cumplir las prácticas en ese nivel y todas las prácticas de los niveles inferiores (Fontana *et al.*, 2015).

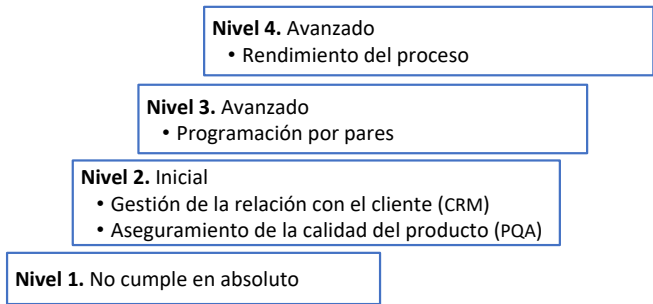


Figura 10.23. Niveles del modelo xPMM

Fuente: elaboración propia con base en Nawrocki *et al.* (2001).

En la tabla 10.9, se presentan las prácticas específicas por nivel recomendadas por el modelo.

Tabla 10.9. Prácticas específicas por nivel en el modelo xPMM

Nivel	Práctica XP	Prácticas específicas
Nivel 2: Inicial	Pruebas de aceptación	Identificar inconsistencias entre el trabajo del proyecto y los requisitos (Requisitos, SP 1.5).
		Monitorear los riesgos del proyecto (Monitoreo de proyecto, SP 1.3).
		Especificar medidas (Medición, SP 1.2).
		Recopilar datos de medición (Medición, SP 2.1).
		Comunicar resultados (Medición, SP 2.4).
	Juego de planificación	Evaluar objetivamente servicios y productos de trabajo (Aseguramiento de la calidad, SP 1.2).
		Obtener compromiso con los requisitos (Requisitos, SP 1.2).
		Gestionar cambios de requisitos (Requisitos, SP 1.3).
		Estimar el alcance del proyecto (Planificación del proyecto, SP 1.1).
		Establecer estimaciones de los atributos del proyecto (Planificación del proyecto, SP 1.2).
		Conciliar el trabajo con los niveles de recursos (Planificación del proyecto, SP 3.2).
		Obtener el compromiso del plan (Planificación del proyecto, SP 3.3).

Nivel	Práctica XP	Prácticas específicas
Nivel 2: Inicial	Gestión de la relación con el cliente	PO. El juego de planificación se usa para crear planes de proyecto.
		P2. Las historias de los usuarios son escritas.
		P3. La planificación de la entrega crea el cronograma.
		P4. Realizar entregas pequeñas frecuentes.
		P5. La velocidad del proyecto es medida.
		P6. El proyecto está dividido en iteraciones.
		P7. La planificación de la iteración inicia en cada iteración.
	Garantía de calidad del producto	D1. Escoger una metáfora del sistema.
		D3. Crear soluciones para reducir el riesgo.
		D4. Ninguna funcionalidad es agregada tempranamente.
		C0a. Cliente eficazmente colaborador.
		C2. Codificar primero las pruebas unitarias.
		C5a. Integrar frecuentemente.
		C7. Dejar la optimización para lo último.
		A. Todo el código debe tener pruebas unitarias.
		T1a. Todo el código debe pasar por pruebas unitarias y la puntuación no debe publicarse antes de que pueda liberarse.
		T2. Cuando se encuentra un error, se debe crear una prueba.
		T3. Las pruebas de aceptación se ejecutan con frecuencia y se publica el puntaje.
Nivel 3: Avanzado	Programación por pares	P8. Mover a las personas.
		C0b. El cliente realiza visitas frecuentes al equipo de desarrollo.
		C1. El código debe estar escrito según los estándares acordados.
		C3. Todo el código de producción está programado por pares.
		C4. Solo un par integra código a la vez.
		C5b. Integrar frecuentemente.
		C6. Utilizar la propiedad colectiva de código.
		C8b. Las horas extras se recopilan y publican.
		C9. Se utiliza un sistema de gestión de versiones.
		T4. Las pruebas automatizadas se utilizan para apoyar las pruebas de integración frecuentes.
Nivel 4: Maduro	Rendimiento del proyecto	F0. Crear un espacio de trabajo abierto para el equipo.
		C0. Cliente en el sitio.
		C8. No hay horas extras.
		T1. Todo el código debe pasar todas las pruebas unitarias antes de que se pueda liberar.
		La satisfacción del cliente se logra.

Fuente: elaboración propia con base en Nawrocki *et al.* (2001).

10.5.13. Otros modelos

Otros modelos relacionados con la generación de propuestas **SPI** en pymes desarrolladoras de software que se pueden encontrar en la literatura son:

- ▶ **PRISMS** (Allen *et al.*, 2003): Modelo de proceso desarrollado en el Reino Unido. Incluye la definición de objetivos de negocio, una revisión del modelo de procesos actual y una actividad de consulta para que el equipo de desarrolladores tome posesión del programa **SPI**. Luego, se identifican las áreas clave del proceso y se definen métricas que permitirán realizar un plan de mejora. Después, el plan es implementado y se obtiene retroalimentación que se reflejará en ajustes del modelo de procesos. El modelo se diseñó específicamente para pymes de software.
- ▶ **TAPISTRY** (Kuvaja *et al.*, 1999): Es una guía **SPI** elaborada en Finlandia que consiste en que un grupo de expertos en **SPI** orienta la autoevaluación de los procesos y la planificación de la mejora en un seminario de dos días. Se compone de cinco fases: definición de las necesidades del negocio, evaluación de los procesos, definición de las metas de mejora y de las prioridades, definición de las acciones de mejora y definición del presupuesto y de los tiempos de la mejora.
- ▶ **TOPS** (Bucci *et al.*, 2000): Modelo desarrollado en Italia. Permite evaluar procesos basados en **ISO/IEC 15504** centrándose específicamente en pymes. La evaluación se realiza con base en un cuestionario que consta de tres partes: información sobre la empresa, información sobre la unidad de desarrollo de software y evaluación de los procesos de software. Se evalúan tres procesos específicos del modelo **SPICE**: desarrollo de requisitos de software, integración y pruebas de software y ejecución. Para cada proceso específico, se realizan cinco preguntas. Cada pregunta corresponde a un nivel de madurez y resume las mejores prácticas en ese nivel.

▶ 10.6. Técnicas para **SPI** en pymes

Las técnicas son conjuntos de procedimientos con pasos específicos para cumplir un objetivo dentro de las iniciativas de mejora del proceso de software. En este apartado, se describirá la técnica **PISKO** junto con su extensión: la técnica **LAPPI**.

La técnica **PISKO** se basa en la idea de que cualquier intento de modelar el proceso de software real debe enfatizar la opinión y la experiencia de los expertos de la organización objetivo. Esta técnica está diseñada para ser fácil de utilizar y requiere una cantidad mínima de recursos (Valtanen y Ahonen, 2008).

La técnica de modelado **PISKO** incluye seis fases presentadas en la figura 10.24 (Ahonen *et al.*, 2002):

- Modelar el proceso con la técnica de gráfico de pared.
- Analizar la información recopilada y definir los problemas y puntos de mejora.
- Crear una versión electrónica de las descripciones del proceso.
- Inspeccionar y aprobar las versiones electrónicas.
- Analizar y mejorar las descripciones de procesos aprobados.
- Inspeccionar los resultados.

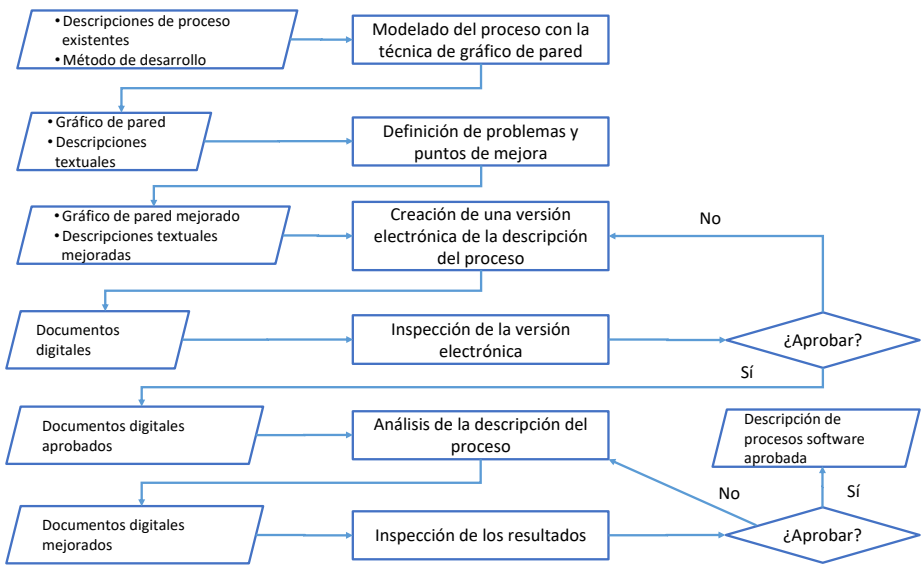


Figura 10.24. Descripción del proceso de la técnica **PISKO**

Fuente: elaboración propia con base en Ahonen *et al.* (2002).

La técnica del diagrama de pared consiste de sesiones en las que se construye el modelo del proceso actual y se identifican problemas relacionados con él. En la mayoría de los casos, participan siete personas:

cinco expertos de la organización (que conocen el proceso) y dos consultores (con los roles del presidente de las sesiones y el secretario). Los expertos identifican las fases principales y flujos de información del proceso de software. Un ejemplo de diagrama de pared se muestra en la figura 10.25. El resultado de la primera fase es un diagrama de pared que presenta el proceso del software actual (*as-is*).

La definición de los problemas es la segunda fase del proceso, en la que se encuentran y describen los problemas existentes en el proceso de software. Cada problema se nombra y enumera. Los expertos localizan la fase o el flujo de información en la que ocurre el problema. Los expertos describen el problema y su importancia e identifican las fases que lo afectan.

La siguiente fase es la documentación de proceso e inspección. Los consultores crean una versión digital del gráfico de pared y de las notas del secretario. La descripción del proceso se refina y se completa con descripciones textuales. La descripción del proceso digital se envía a la organización para su inspección. Los malentendidos, los errores y las omisiones se aclaran y corrigen de acuerdo con los comentarios proporcionados por los expertos de la organización.

Se continúa con el análisis del proceso e inspección de resultados. El presidente y el secretario analizan el proceso. El análisis debe considerar modelos y estándares de procesos existentes. El resultado del análisis es un informe completo del proceso y una lista de sugerencias para mejorar.

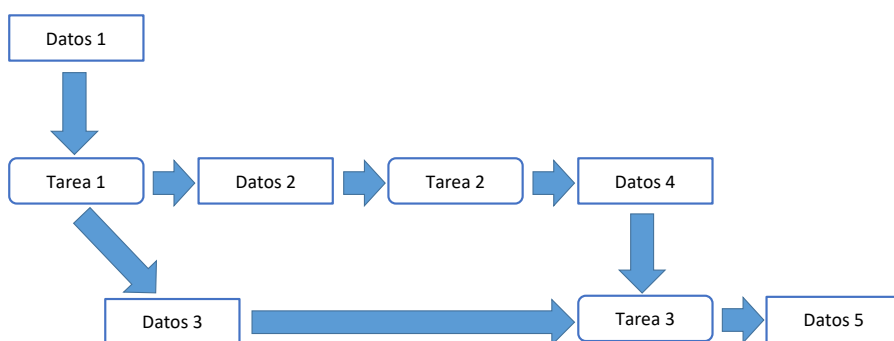


Figura 10.25. Un fragmento de proceso con la técnica PISKO

Fuente: elaboración propia con base en Ahonen *et al.* (2002).

La técnica **PISKO** ha sido extendida por la técnica **LAPPI** (Raninen *et al.*, 2013). La técnica se presenta en la figura 10.26. A diferencia de la técnica **PISKO**, todo antes del paso 7 es nuevo en la técnica **LAPPI**. En pocas palabras, el modelado del proceso tiene los siguientes pasos: elegir al personal adecuado, motivarlos, modelar los roles, los flujos de información y el proceso (aplicando la técnica de diagrama de pared), combinar los roles, los flujos de información y el modelo de proceso para proporcionar una visión general del proceso, analizar, proporcionar sugerencias de mejora, proponer un seguimiento y organizarlo.

Dos talleres de modelado de procesos son la columna vertebral de la técnica **LAPPI**. En el primer taller de modelado, se modelan los roles y los flujos de información relacionados con el proceso. El segundo taller se enfoca en modelar el proceso. El resultado de la técnica **LAPPI** es una visión general del proceso y sugerencias de mejora generadas a través de los talleres de modelado de procesos. Además, aplicarlo solo requiere una cantidad moderada de recursos (Raninen *et al.*, 2013).

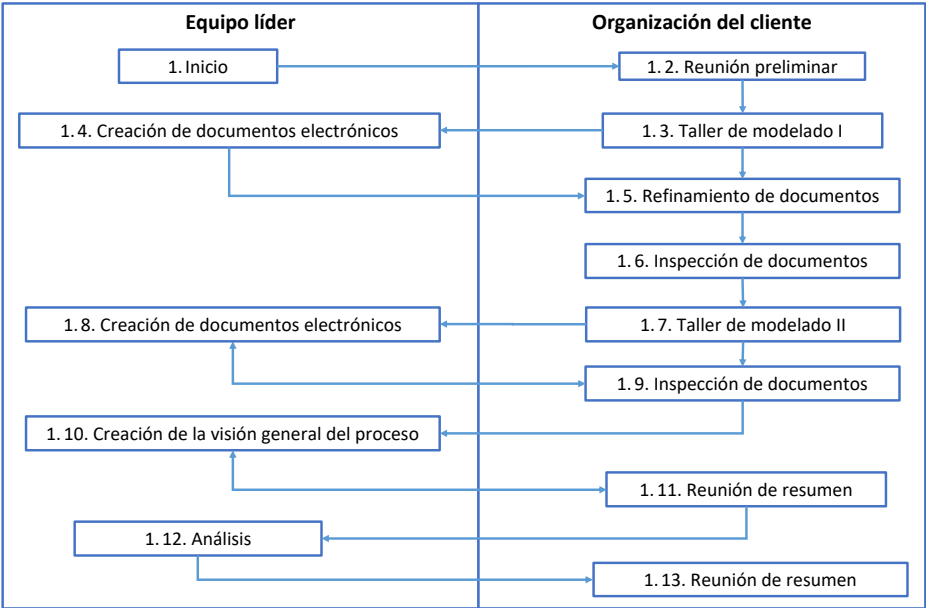


Figura 10.26. Etapas de la técnica LAPPI

Fuente: elaboración propia con base en Raninen *et al.* (2013).

► Desafíos

Las pequeñas organizaciones desarrolladoras de software parecen estar más orientadas a la práctica, mientras las organizaciones grandes parecen más preocupadas por los procesos formales de desarrollo de software (Dybå, 2003). Algunos desafíos de las pequeñas organizaciones y pequeños proyectos de desarrollo de software son gestión de requisitos, generación de la documentación, gestión de proyectos, asignación de recursos, medición del progreso, realización de revisiones y provisión de capacitación (Johnson y Brodman, 1998).

Kautz *et al.* (2000) concluyeron en sus hallazgos algunas lecciones para pymes que quieren realizar actividades **SPI**: a) utilizar un modelo estructural para organizar el proceso, b) el modelo debe ajustarse a las condiciones particulares de las organizaciones y c) las actividades **SPI** se deben gestionar como un proyecto con roles, responsabilidades y recursos asignados y documentados.

Como las pymes tienen presupuestos y recursos limitados, los siguientes factores son importantes a considerar en este tipo de organizaciones (Mishra y Mishra, 2009):

- Iniciar con el proceso “como es”, no con el proceso “como debería ser”, para aprovechar las prácticas efectivas y eliminar la resistencia. El proceso “como debería ser” puede o no ser factible en la cultura y el entorno dados (Paulk, 1998).
- Para que una organización pequeña implemente con éxito las iniciativas SPI, debe responder preguntas clave, como ¿dónde estoy?, ¿qué hay que mejorar? y ¿cómo mejorarlo?
- Aunque los procesos estándar brindan una base, cada proyecto tiene necesidades y restricciones únicas para su adaptación que pueden generar una resistencia significativa. Se puede seguir el aforismo: “No requieras procesos que no tienen sentido” (Hoffman, 1998).
- Los procesos de software definidos e implementados en pymes deben combinar la necesidad de flexibilidad y creatividad.
- Los factores culturales desempeñan un papel importante en que una iniciativa **SPI** de un país puede no tener el mismo éxito en otro. Estos factores, como las características geográficas, la cultura corporativa y los objetivos de negocios, influyen en la selección del método SPI (Suwanya y Kurutach, 2008).

Los modelos para pymes presentados en este capítulo se basan en algunos modelos estándar existentes orientados para grandes empresas. Sin embargo, estos enfoques se pueden adaptar y simplificar, implementando guías ligeras de proceso o paquetes de acción o de mejora gradual con diferentes métodos de evaluación para que puedan ser utilizados por estas organizaciones. Además, los modelos para pymes son de naturaleza continua y no requieren mucha experiencia en las etapas de evaluación y planificación de actividades SPI. Sin embargo, a veces se requiere contar con la ayuda de asesores/expertos externos, lo que podría ser difícil para las pymes, ya que tienen que asumir esos costos adicionales (Hall *et al.*, 2002).

► Ejemplos

Como ejemplo, se presenta el caso de una pyme peruana que implementó la norma **ISO/IEC 29110**. La empresa estaba conformada por cuatro personas, entre las que se encontraban sus propietarios que actuaban como director ejecutivo y director de tecnología. La empresa se especializaba en brindar servicios de desarrollo de software y automatización de procesos de negocios (García Paucar *et al.*, 2015).

La implementación del perfil básico comenzó con una reunión inicial de coordinación en la que se obtuvo el compromiso de la dirección y el equipo técnico. Después de analizar los requisitos obligatorios definidos por el estándar, se realizó un análisis de brechas del proceso actual frente a los requisitos del estándar.

La pyme usaba prácticas ágiles para implementar soluciones software. Los requisitos definidos para la implementación del perfil básico de la **ISO/IEC 29110** fueron:

- Cumplir con las condiciones de entrada establecidas en **ISO/IEC 29110-4-1** (especificaciones de perfil).
- Cumplir con los procesos establecidos por **ISO/IEC 29110-4-1**: Proceso de gestión de proyectos y proceso de implementación de software.
- Cumplir con las actividades de proceso establecidas por **ISO/IEC 29110-4-1**.
- Cumplir con los objetivos de los procesos establecidos en **ISO/IEC 29110-4-1**.
- Cumplir con los productos enumerados en **ISO/IEC 29110-4-1**.

Utilizando listas de verificación desarrolladas previamente, se realizó un análisis inicial para verificar que la pyme cumpliera con las condiciones de ingreso al perfil básico. Como resultado del análisis preliminar, se determinó que la pyme cumplía con las condiciones de ingreso del perfil básico.

Se realizaron reuniones en las que se utilizaron listas de verificación, una por actividad de proceso, para verificar que las tareas de cada actividad, realizadas por la pyme, contribuían al logro de los objetivos de su proceso asociado.

De igual manera, se analizaron los productos de trabajo producidos por la pyme. Se definieron plantillas de productos de trabajo para los procesos de gestión de proyectos e implementación del sistema. Estas plantillas fueron adaptadas para soportar el desarrollo ágil de software y que fueran compatibles con herramientas de la empresa.

Se implementaron herramientas para soportar nuevas tareas y flujos de trabajo que no formaban parte de la pyme antes de la implementación del estándar. Estas herramientas se integraron con otras de la organización.

Como parte del proceso de implementación, se documentaron los nuevos procesos de gestión de proyectos e implementación de sistemas utilizando un lenguaje de modelado de procesos de negocio. También los nuevos procesos se documentaron detallando cada una de sus actividades, roles y artefactos de entrada y salida.

También se definieron indicadores y métricas para ambos procesos, como el número de solicitudes de cambio y el porcentaje de errores después de probar los componentes del software. Finalmente, se elaboró una matriz de trazabilidad entre cada producto de trabajo **ISO/IEC 29110** y los documentos o herramientas equivalentes utilizados por la pyme.

► Resumen

Las pymes desarrolladoras de software, al igual que las grandes, requieren modelos que permitan mejorar sus procesos de software. Sin embargo, los modelos **SPI** implican grandes esfuerzos, costos e inversiones. Los modelos como **CMMI** son costosos de implementar e incluyen muchas áreas de procesos que no son de vital importancia para las pymes.

Para afrontar estos desafíos, se han generado estándares, infraestructuras, modelos y técnicas de **SPI** en pymes desarrolladoras de software.

El estándar más conocido y utilizado es **ISO/IEC 29110** “Perfiles del ciclo de vida para pequeñas organizaciones”. Incluye cuatro grupos de perfiles genéricos: básico, entrada, intermedio y avanzado. El perfil básico contiene los procesos de gestión de proyectos e implementación de software. Además, para ayudar con el despliegue e implementación del estándar, se han desarrollado una serie de paquetes y guías de implementación.

Las infraestructuras establecen prácticas comunes para crear, organizar, interpretar y analizar descripciones arquitectónicas utilizadas dentro de un dominio particular de aplicación o comunidad de partes interesadas. Entre las infraestructuras para implementar iniciativas **SPI** en pymes se encuentran: MoProSoft, Competisoft, **MPS.BR**, ITmark y Tutelkan. Cada una incluye sus modelos de referencia y generalmente un modelo de evaluación.

También existen diversos modelos y métodos específicos para implementar iniciativas **SPI** en pymes, entre los cuales se encuentran **OWPL**, **MARES**, **EPA**, Adept, Impact, MesoPyme, **ASPE/MS**, iFlap, Processus, **SPM**, **RAPID** y **XPMM**.

PISKO y **LAPPI** son técnicas que permiten implementar un plan **SPI** en jornadas cortas y con pocos recursos.

► Ejercicios

1. ¿Qué características tienen las pequeñas y empresas desarrolladoras de software?
2. ¿Qué factores críticos cree que se deben considerar en las pymes para implementar prácticas **SPI**?
3. Realice un cuadro comparativo de las diferentes infraestructuras y modelos de procesos de software orientados a pymes en atención a roles participantes, entradas, salidas, restricciones y método de evaluación.
4. Describa los principales componentes de estándar **ISO/IEC 29110**.
5. ¿En qué se diferencia la técnica **PISKO** de la técnica **LAPPI** para **SPI** en pymes?

► Referencias

- Ahonen, J. J., Forsell, M. y Taskinen, S. K. (2002). A modest but practical software process modeling technique for Software Process Improvement. *Software Process: Improvement and Practice*, 7(1), 33-44. <https://doi.org/10.1002/spip.152>
- Alexandre, S., Renault, A. y Habra, N. (2006). OWPL: A gradual approach for Software Process Improvement in SMEs. En *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)* (pp. 328-335). **IEEE**. <https://doi.org/10.1109/EUROMICRO.2006.48>
- Allen, P., Ramachandran, M. y Abushama, H. (2003). **PRISMS**: An approach to Software Process Improvement for small to medium enterprises. En *Third International Conference on Quality Software, 2003. Proceedings* (pp. 211-214). **IEEE**. <https://doi.org/10.1109/QSIC.2003.1319105>
- Anacleto, R., Von Wangenheim, C. G., Salviano, C. F y Savi, R. (2004). A method for process assessment in small software companies. En *Proceedings of 4th International Software Process Improvement and Capability Determination Conference (SPICE04)* (pp. 69-76). Springer. https://www.inf.ufsc.br/~c.wangenheim/download/MARES-Method_spice2004_vref.pdf
- Baskerville, R. y Pries-Heje, J. (1999). Knowledge capability and maturity in software management. *ACM SIGMIS Database: The DATABASE for Advances in Information Systems*, 30(2), 26-43. <https://doi.org/10.1145/383371.383374>
- Basri, S. y O'Connor, R. V. (2010). Understanding the perception of very small software companies towards the adoption of process standards. En A. Riel, R. O'Connor, S. Tichkiewitch y R. Messnarz (eds.), *Systems, Software and Services Process Improvement. EuroSPI 2010. Communications in Computer and Information Science* (pp. 153-164). Springer. https://doi.org/10.1007/978-3-642-15666-3_14
- Berander, P. y Andrews, A. (2005). Requirements prioritization. En A. Aurum y C. Wohlin (eds.), *Engineering and managing software requirements* (pp. 69-94). Springer. https://doi.org/10.1007/3-540-28244-0_4
- Bucci, G., Campanai, M. y Cignoni, G. A. (2000). Rapid assessment to solicit process improvement in SMEs. En *Proceedings of 7th European Software Process Improvement Conference (EuroSPI)*. Springer. <http://groups.di.unipi.it/~giovanni/CV/Pubb/GAC-2001-SQP-Doc.pdf>
- Calvo-Manzano Villalón, J. A., Cuevas Agustín, G., San Feliu Gilabert, T., De Amescua Seco, A., García Sánchez, L. y Pérez Cota, M. (2002). Experiences in the application of Software Process Improvement in SMEs. *Software Quality Journal*, 10, 261-273. <https://doi.org/10.1023/A:1021638523413>
- Cater-Steel, A. P. (2004). Low-rigour, rapid software process assessments for small software development firms. En *2004 Australian Software Engineering Conference. Proceedings* (pp. 368-377). **IEEE**. <https://doi.org/10.1109/ASWEC.2004.1290490>
- Cater-Steel, A., Toleman, M. y Rout, T. (2006). Process improvement for small firms: An evaluation of the **RAPID** assessment-based method. *Information and Software Technology*, 48(5), 323-334. <https://doi.org/10.1016/j.infsof.2005.09.012>

- Chen, X. y Staples, M. (2007). Using practice outcome areas to understand perceived value of **CMMI** specific practices for SMEs. En P. Abrahamsson, N. Baddoo, T. Margaria y R. Messnarz (eds.), *Software Process Improvement. EuroSPI 2007. Lecture Notes in Computer Science* (pp. 59-70). Springer. https://doi.org/10.1007/978-3-540-75381-0_6
- Chin, A. (2000). 10 factors on fostering innovation in small and medium-sized organisations. En *Proceedings of the 2000 IEEE International Conference on Management of Innovation and Technology. ICMIT 2000. 'Management in the 21st Century'* (Cat. No. 00EX457) (vol. 1, pp. 473-478). **IEEE**. <https://doi.org/10.1109/ICMIT.2000.917383>
- Clarke, P. y O'Connor, R. V. (2012a). The influence of **SPI** on business success in software SMEs: An empirical study. *Journal of Systems and Software*, 85(10), 2356-2367. <https://doi.org/10.1016/j.jss.2012.05.024>
- Coleman, G. y O'Connor, R. (2008). Investigating software process in practice: A grounded theory perspective. *Journal of Systems and Software*, 81(5), 772-784. <https://doi.org/10.1016/j.jss.2007.07.027>
- Dybå, T. (2003). Factors of Software Process Improvement success in small and large organizations: An empirical study in the scandinavian context. *ACM SIGSOFT Software Engineering Notes*, 28(5), 148-157. <https://doi.org/10.1145/949952.940092>
- European Commission. (2020). *What is a SME?* https://ec.europa.eu/growth/smes/business-friendly-environment/sme-definition_en/
- Fontana, R. M., Meyer, V., Reinehr, S. y Malucelli, A. (2015). Progressive outcomes: A framework for maturing in agile software development. *Journal of Systems and Software*, 102, 88-108. <https://doi.org/10.1016/j.jss.2014.12.032>
- García Paucar, L. H., Laporte, C. Y., Arteaga, Y. y Bruggmann, M. (2015). Implementation and Certification of **ISO/IEC** 29110 in an IT Startup in Peru. *Software Quality Professional Journal*, 17(2), 16-29. <https://profs.etsmtl.ca/claporte/Publications/Publications/iso-29110-in-an-it-startup-in-peru.pdf>
- García-Mireles, G. A. y Rodríguez-Castillo, I. (2009). Software engineering area curricular evaluation method based in Moprosoft. En *2009 Mexican International Conference on Computer Science* (pp. 272-279). **IEEE**. <https://doi.org/10.1109/ENC.2009.19>
- Hall, T., Rainer, A. y Baddoo, N. (2002). Implementing Software Process Improvement: An empirical study. *Software Process: Improvement and Practice*, 7(1), 3-15. <https://doi.org/10.1002/spip.150>
- Hauck, J. C. R., Almeida, I., Araujo, R., Dymow, J. y Neto, M. F. (2015). Harmonizing **MPS.BR** and certics: A case study in a maturity level f organization. En *2015 29th Brazilian Symposium on Software Engineering* (pp. 61-70). **IEEE**. <https://doi.org/10.1109/SBES.2015.22>
- Hauck, J. C. R., Gresse Von Wangenheim, C., de Souza, R. H. y Thiry, M. (2008). Process reference guides: Support for improving software processes in alignment with reference models and standards. En R. V. O'Connor, N. Baddoo, K. Smolander y R. Messnarz (eds.), *Software Process Improvement. EuroSPI 2008. Communications in Computer and Information Science* (pp. 70-81). Springer. https://doi.org/10.1007/978-3-540-85936-9_7

- Hoffman, L. (1998). Small projects and the CMM. En *Key Practices to the CMM: Inappropriate for Small projects? Proceedings of the 1998 Software Engineering Process Group Conference* (pp. 9-12). Chicago.
- Horvat, R. V., Rozman, I. y Györkös, J. (2000). Managing the complexity of **SPI** in small companies. *Software Process: Improvement and Practice*, 5(1), 45-54. [https://doi.org/10.1002/\(SICI\)1099-1670\(200003\)5:1%3C45::AID-SPIP110%3E3.0.CO;2-2](https://doi.org/10.1002/(SICI)1099-1670(200003)5:1%3C45::AID-SPIP110%3E3.0.CO;2-2)
- ISO/IEC 29110-1. (2010). *Software engineering - lifecycle profiles for very small entities (VSE) - Part 1: VSE profiles overview*. International Organization for Standardization.
- ISO/IEC 29110-2. (2010). *Software engineering - lifecycle profiles for very Small entities (VSE) - Part 2: Framework and taxonomy*. International Organization for Standardization.
- ISO/IEC 29110-3. (2010). *Software engineering - lifecycle profiles for very small entities (VSE) - Part 3: Assessment guide*. International Organization for Standardization.
- ISO/IEC 29110-4. (2010). *Software engineering - lifecycle profiles for very small entities (VSE) - Part 4: Specifications of VSE profiles*. International Organization for Standardization.
- ISO/IEC 29110-5. (2010). *Software engineering - lifecycle profiles for very small entities (VSE) - Part 5: Management and engineering guide*. International Organization for Standardization.
- ISO/IEC 42010. (2007). *Systems and software engineering - recommended practice for architectural description of software-intensive systems*. International Organization for Standardization.
- ITmark. (2020). *Modelo ITmark*. <http://it-mark.eu/>
- Järvi, A., Mäkilä, T. y Hakonen, H. (2006). Changing role of SPI: Opportunities and challenges of process modeling. En I. Richardson, P. Runeson y R. Messnarz (eds.), *Software Process Improvement. EuroSPI 2006. Lecture Notes in Computer Science* (pp. 135-146). Springer. https://doi.org/10.1007/11908562_13
- Jeners, S., Clarke, P., O'Connor, R. V., Buglione, L. y Lepmets, M. (2013). Harmonizing software development processes with software development settings: A systematic approach. En F. McCaffery, R. V. O'Connor y R. Messnarz (eds.), *Systems, Software and Services Process Improvement. EuroSPI 2013. Communications in Computer and Information Science* (pp. 167-178). Springer. https://doi.org/10.1007/978-3-642-39179-8_15
- Johnson, D. L. y Brodman, J. G. (1998). Applying the CMM to small organizations and small projects. En *Proceedings of the 1998 Software Engineering Process Group Conference*.
- Kachigan, S. K. (1986). *Statistical analysis: An interdisciplinary introduction to univariate & multivariate methods*. Radius Press.
- Kautz, K., Hansen, H. W. y Thaysen, K. (2000). Applying and adjusting a Software Process Improvement model in practice: The use of the **IDEAL** model in a small software enterprise. En *Proceedings of the 22nd international conference on Software engineering* (pp. 626-633). <https://doi.org/10.1145/337180.337492>
- Kuvaja, P., Palo, P. y Bicego, A. (1999). **TAPISTRY**: A Software Process Improvement approach tailored for small enterprises. *Software Quality Journal*, 8(2), 149-156. <https://doi.org/10.1023/A:1008909011736>

- Lester, N. G., Wilkie, F. G., McFall, D. y Ware, M. P. (2007). Evaluating the internal consistency of the base questions in the Express Process Appraisal. En *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)* (pp. 289-296). **IEEE**. <https://doi.org/10.1109/EUROMICRO.2007.30>
- Lester, N. G., Wilkie, F. G., McFall, D. y Ware, M. P. (2010). Investigating the role of **CMMI** with expanding company size for small-to medium-sized enterprises. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(1), 17-31. <https://doi.org/10.1002/spip.409>
- López, O., Esquivel-Vega, G., Valerio, A. L., Víquez-Acuña, L., Víquez-Acuña, O. y Umaña, D. (2012). *Mejora de procesos para fomentar la competitividad de la pequeña y mediana industria del software de Iberoamérica*. Instituto Tecnológico de Costa Rica. <https://repositoriotec.tec.ac.cr/bitstream/handle/2238/3358/mejora-procesos-fomentar-competitividad.pdf?sequence=1&isAllowed=y>
- McCaffery, F. y Coleman, G. (2009). Lightweight **SPI** assessments: What is the real cost? *Software Process: Improvement and Practice*, 14(5), 271-278. <https://doi.org/10.1002/spip.430>
- McCaffery, F., McFall, D. y Wilkie, F.G. (2005). Improving the Express Process Appraisal method. En F. Bomarius y S. Komi-Sirviö (eds.), *Product Focused Software Process Improvement. PROFES 2005. Lecture Notes in Computer Science* (pp. 286-298). Springer. https://doi.org/10.1007/11497455_24
- McCaffery, F., Richardson, I. y Coleman, G. (2006). A Adept: A software process appraisal method for small to medium-sized Irish software development organisations. En *Proceedings of the European Software Process Improvement and Innovation Conference (Euro-SPI06)*. <https://eprints.dkit.ie/173/>
- Mishra, D. y Mishra, A. (2009). Software process improvement in SMEs: A comparative view. *Computer Science and Information Systems*, 6(1), 111-140. <https://doi.org/10.2298/CSIS0901111M>
- MPS.BR. (2012). *Melhoria de Processo de Software Brasileiro: Guia Geral*. Softex.
- Nawrocki, J. R., Jasiński, M., Walter, B. y Wojciechowski, A. (2002). Combining eXtreme Programming with ISO 9000. En H. Shafazand y A. M. Tjoa (eds.), *EurAsia-ICT 2002: Information and Communication Technology. EurAsia-ICT 2002. Lecture Notes in Computer Science* (pp. 786-794). Springer. https://doi.org/10.1007/3-540-36087-5_91
- Nawrocki, J., Walter, B. y Wojciechowski, A. (2001). Toward maturity model for eXtreme Programming. En *Proceedings 27th EUROMICRO Conference. 2001: A Net Odyssey* (pp. 233-239). **IEEE**. <https://doi.org/10.1109/EURMIC.2001.952459>
- O'Connor, R. V. (2014). Early stage adoption of **ISO/IEC 29110** software project management practices: A case study. En A. Mitasiunas, T. Rout, R. V. O'Connor y A. Dorling (eds.), *Software Process Improvement and Capability Determination. SPICE 2014. Communications in Computer and Information Science* (pp. 226-237). Springer. https://doi.org/10.1007/978-3-319-13036-1_20
- O'Connor, R. V. y Laporte, C. Y. (2014). An innovative approach to the development of an international software process lifecycle standard for very small entities. *International Journal Information Technologies and Systems Approach (IJITSA)*, 7(1), 1-22. <https://doi.org/10.4018/ijitsa.2014010101>

- Oktaba, H. (dir.) (2005). *Modelo de procesos para la industria de software MoProSoft por niveles de capacidad de procesos. Versión 1.3. NMX-059/01-NYCE-2005*. Organismo Nacional de Normalización y Evaluación de la Conformidad.
- Oktaba, H., Alquicira Esquivel, C., Ramos, A. S., Palacios Elizalde, J., Pérez Escobar, C. J. y López Lira Hinojo, F. (2004). *Método de evaluación de procesos para la industria del software, EvalProSoft V1.1*. Secretaría de Economía de México.
- Oktaba, H., García, F., Piattini, M., Ruiz, F., Pino, F. J. y Alquicira, C. (2007). Software process improvement: The Competisoft project. *Computer*, 40(10), 21-28. <https://doi.org/10.1109/MC.2007.361>
- Paulk, M. C. (1998). Using the software CMM in small organizations. En *The Joint 1998 Proceedings of the Pacific Northwest Software Quality Conference and the Eighth International Conference on Software Quality* (pp. 350-361). Carnegie Mellon University. <http://www.iso.staratel.com/iso/CMM/Article/cmm-small.pdf>
- Pettersson, F., Ivarsson, M., Gorschek, T. y Öhman, P. (2008). A practitioner's guide to light weight software process assessment and improvement planning. *Journal of Systems and Software*, 81(6), 972-995. <https://doi.org/10.1016/j.jss.2007.08.032>
- Piattini, M. y Garzás-Parra, J. (2007). *Fábricas de software: Experiencias, tecnologías y organización. RA-MA*.
- Pino, F. J., García, F. y Piattini, M. (2008). Software process improvement in small and medium software enterprises: A systematic review. *Software Quality Journal*, 16, 237-261. <https://doi.org/10.1007/s11219-007-9038-z>
- Raninen, A., Ahonen, J. J., Sihvonen, H. M., Savolainen, P. y Beecham, S. (2013). **LAPPI**: A light-weight technique to practical process modeling and improvement target identification. *Journal of Software: Evolution and Process*, 25(9), 915-933. <https://doi.org/10.1002/smr.1571>
- Regnell, B., Höst, M., och Dag, J. N., Beremark, P. y Hjelm, T. (2001). An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6, 51-62. <https://doi.org/10.1007/s007660170015>
- Richardson, I. (2002). SPI models: What characteristics are required for small software development companies? En J. Kontio y R. Conradi (eds.), *Software Quality: ECSQ 2002. ECSQ 2002. Lecture Notes in Computer Science* (pp. 100-113). Springer. https://doi.org/10.1007/3-540-47984-8_14
- Richardson, I. y Ryan, K. (2001). Software process improvements in a very small company. *Software Quality professional*, 3(2), 23-35. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=37f9a4ce4e41816901f3c9f99aeca880f18f18c2>
- Rozman, I., Vajde Horvat, R., GyÖrkÖs, J. y Hericüko, M. (1997). Processus: Integration of **SEI CMM** and **ISO** quality models. *Software Quality Journal*, 6, 37-63. <https://doi.org/10.1023/A:1018539413913>
- Saaty, T. L. y Vargas, L. G. (2012). *Models, methods, concepts & applications of the analytic hierarchy process*. Springer.
- Sánchez-Gordón, M. L., Colomo-Palacios, R., de Amescua Seco, A. y O'Connor, R. V. (2016). The route to Software Process Improvement in small-and medium-sized enterprises.

- En M. Kuhrmann, J. Münch, I. Richardson, A. Rausch y H. Zhang (eds.), *Managing software process evolution*. Springer. https://doi.org/10.1007/978-3-319-31545-4_7
- Santos, G., Kalinowski, M., Rocha, A. R., Travassos, G. H., Weber, K. C. y Antonioni, J. A. (2012). MPS. BR program and MPS model: Main results, benefits and beneficiaries of Software Process Improvement in Brazil. En *2012 Eighth International Conference on the Quality of Information and Communications Technology* (pp. 137-142). **IEEE**. <https://doi.org/10.1109/QUATIC.2012.42>
- Savolainen, P., Sihvonen, H. M. y Ahonen, J. J. (2007). SPI with lightweight software process modeling in a small software company. En P. Abrahamsson, N. Baddoo, T. Margaria y R. Messnarz (eds.), *Software Process Improvement. EuroSPI 2007. Lecture Notes in Computer Science* (pp. 71-81). Springer. https://doi.org/10.1007/978-3-540-75381-0_7
- Scott, L., Jeffery, R., Carvalho, L., D'ambra, J. y Rutherford, P. (2001). Practical Software Process Improvement-the IMPACT project. En *Proceedings 2001 Australian Software Engineering Conference* (pp. 182-189). **IEEE**. <https://doi.org/10.1109/ASWEC.2001.948512>
- Softex. (2020). *Modelos de referência*. <https://softex.br/mpsbr/modelos/>
- Stambollian, A., Habra, N., Laporte, C. Y., Desharnais, J. M. y Renault, A. (2006). **OWPL**: A light model & methodology for initiation Software Process Improvement. En *Proceedings of the 6th SPICE Conference on Process Assessment and Improvement* (pp. 97-105).
- Suwanya, S. y Kurutach, W. (2008). An analysis of Software Process Improvement for sustainable development in Thailand. En *2008 8th IEEE International Conference on Computer and Information Technology* (pp. 724-729). **IEEE**. <https://doi.org/10.1109/CIT.2008.4594764>
- Turgeon, J. (2006). *CMMI on the sly for the CMMI shy: Implementing Software Process Improvement in small teams and organizations*. Presentation in **SEPG**.
- Vahaniitty, J. y Rautiainen, K. (2005). Towards an approach for managing the development portfolio in small product-oriented software companies. En *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (pp. 314c-314c). **IEEE**. <https://doi.org/10.1109/HICSS.2005.636>
- Valdés, G., Astudillo, H., Visconti, M. y López, C. (2010). The Tutelkan SPI framework for small settings: A methodology transfer vehicle. En A. Riel, R. O'Connor, S. Tichkewitch y R. Messnarz (eds.), *Systems, Software and Services Process Improvement. EuroSPI 2010. Communications in Computer and Information Science* (pp. 142-1529). Springer. https://doi.org/10.1007/978-3-642-15666-3_13
- Valdés, G., Visconti, M. y Astudillo, H. (2011). The Tutelkan Reference Process: A reusable process model for enabling SPI in small settings. En R. V. O'Connor, J. Pries-Heje y R. Messnarz (eds.), *Systems, Software and Service Process Improvement. EuroSPI 2011. Communications in Computer and Information Science* (pp. 179-190). Springer. https://doi.org/10.1007/978-3-642-22206-1_16
- Valencia, L. S., Villas, P. A. y Ocampo, C. A. (2009). Modelo de calidad de software. *Scientia et Technica*, 2(42), 172-176. <https://www.redalyc.org/pdf/849/84916714032.pdf>
- Valtanen, A. y Ahonen, J. J. (2008). Big improvements with small changes: Improving the processes of a small software company. En A. Jedlitschka y O. Salo (eds.), *Product-Focused Software Process Improvement. PROFES 2008. Lecture Notes in Computer Science*

- (pp. 258-272). Springer. https://doi.org/10.1007/978-3-540-69566-0_22
- Villarroel, R., Gómez, Y., Gajardo, R. y Rodríguez, O. (2009). Implementation of an improvement cycle using the competisoft methodological framework and the Tutelkan platform. En *2009 International Conference of the Chilean Computer Science Society* (pp. 97-104). **IEEE**. <https://doi.org/10.1109/SCCC.2009.20>
- Von Wangenheim, C. G., Anacleto, A. y Salviano, C. F. (2006). Helping small companies assess software processes. **IEEE Software**, 23(1), 91-98. <https://doi.org/10.1109/MS.2006.13>
- Von Wangenheim, C. G., Weber, S., Hauck, J. C. R. y Trentin, G. (2006). Experiences on establishing software processes in small companies. *Information and Software Technology*, 48(9), 890-900. <https://doi.org/10.1016/j.infsof.2005.12.010>
- Weber, K. C., Araújo, E. E. R., da Rocha, A. R. C., Machado, C. A. F., Scalet, D. y Salviano, C. F. (2005). Brazilian software process reference model and assessment method. En P. Yolum, T. Güngör, F. Gürgen y C. Özturan (eds.), *Computer and Information Sciences - ISCIS 2005. ISCIS 2005. Lecture Notes in Computer Science* (pp. 402-411). Springer. https://doi.org/10.1007/11569596_43
- Wheelen, T. L., Hunger, J. D., Hoffman, A. N. y Bamford, C. E. (2017). *Strategic management and business policy*. Pearson.
- Wilkie, F. G., Mc Caffery, F., McFall, D., Lester, N. y Wilkinson, E. (2007). A Low-overhead method for software process appraisal. *Software Process: Improvement and Practice*, 12(4), 339-349. <https://doi.org/10.1002/spip.321>
- Zarour, M., Abran, A. y Desharnais, J. M. (2011). Evaluation of software process assessment methods: Case study. En R. V. O'Connor, T. Rout, F. McCaffery y A. Dorling (eds.), *Software Process Improvement and Capability Determination. SPICE 2011. Communications in Computer and Information Science* (pp. 42-51). Springer. https://doi.org/10.1007/978-3-642-21233-8_4

Reúso del proceso de software

► Objetivos

Los objetivos de este capítulo están orientados a comprender los beneficios que conlleva la reutilización de procesos de software en la ejecución exitosa de proyectos futuros. Al finalizar, el lector estará en capacidad de:

- Comprender el reúso de software como punto de partida para entender el reúso de procesos de software.
- Conocer diferentes tecnologías orientadas al reúso de procesos, como las líneas de procesos y los patrones de procesos.
- Entender cómo adaptar el proceso de software organizacional a las características específicas de un proyecto de desarrollo de software.
- Aplicar técnicas de gestión del conocimiento para el reúso de procesos de software.

Los aspectos de desarrollo de software basado en componentes (apartado 2.3.5), los cuales hacen énfasis en la reutilización de software, también se han aplicado en la definición de procesos de software utilizando componentes reutilizables, es decir, componentes o elementos del proceso. Estos componentes del proceso están organizados mediante estructuras genéricas que establecen los elementos del proceso involucrados, sus interfaces, ordenamiento, interdependencias y otras relaciones entre ellos (Chrissis *et al.*, 2006).

Sobre la base de analogías válidas entre productos de software y procesos de software, las técnicas de reutilización de software se están aplicando en el área del proceso para contribuir a las mejoras en la tarea de definición del proceso (Teixeira *et al.*, 2019). Además, la reutilización del conocimiento proveniente de proyectos anteriores y a partir de los conocimientos de los miembros de los equipos de desarrollo o de fuentes externas son aspectos clave a considerar durante el desarrollo de proyectos. Con la incorporación de estándares, técnicas y métodos de reutilización del proceso, no hay que definir y ejecutar los procesos desde cero, sino conocer, aprender y utilizar las mejores prácticas de desarrollo, las cuales redundarán en la reducción de costos, tiempos de entrega más cortos y la calidad del proceso y producto de software desarrollado.

► 11.1. Conceptos sobre reúso de software

El reúso se puede definir como el uso posterior o uso repetido de un artefacto (o activo). La idea de reutilización en el desarrollo de software fue introducida por primera vez por McIlroy *et al.* (1968), quienes propusieron un catálogo de componentes de software a partir del cual se podía ensamblar el software.

11.1.1. Reúso de software

Antes de definir el reúso del proceso de software, se debe conocer qué es el reúso de software en general, un área relevante en la ingeniería de software. El reúso de software se puede definir como “la práctica sistemática de desarrollar software desde un *stock* de bloques de construcción, así que las similitudes en requisitos o arquitectura entre aplicaciones pueden ser explotadas para lograr beneficios substanciales en productividad, calidad y desempeño del negocio” (Ezran *et al.*, 2012). Por tanto, el reúso de software está relacionado con la reutilización de la información producida durante la ejecución de los proyectos de desarrollo de software anteriores, con el fin de disminuir el esfuerzo necesario para un nuevo proyecto.

Un aspecto clave para mejorar la madurez de una organización desarrolladora de software consiste en la definición de procesos reutilizables, que pueden utilizarse para cumplir las políticas estándar que se adoptan durante la ejecución de cualquier proceso de software en la organización.

11.1.2. Activos de software

Los activos reutilizables están compuestos de una colección de productos de trabajo relacionados que pueden ser reutilizados de una aplicación a otra. La figura 11.1 muestra, utilizando un modelo de clases UML, la estructura de un activo de software. Los activos tienen dos tipos de información: su cuerpo (que contiene una colección de productos de trabajo reutilizables asociados al proceso) y su descripción (que contiene la información necesaria para soportar el proceso de reúso). Cada empresa debe definir su propio modelo de activo para reutilizarlo.

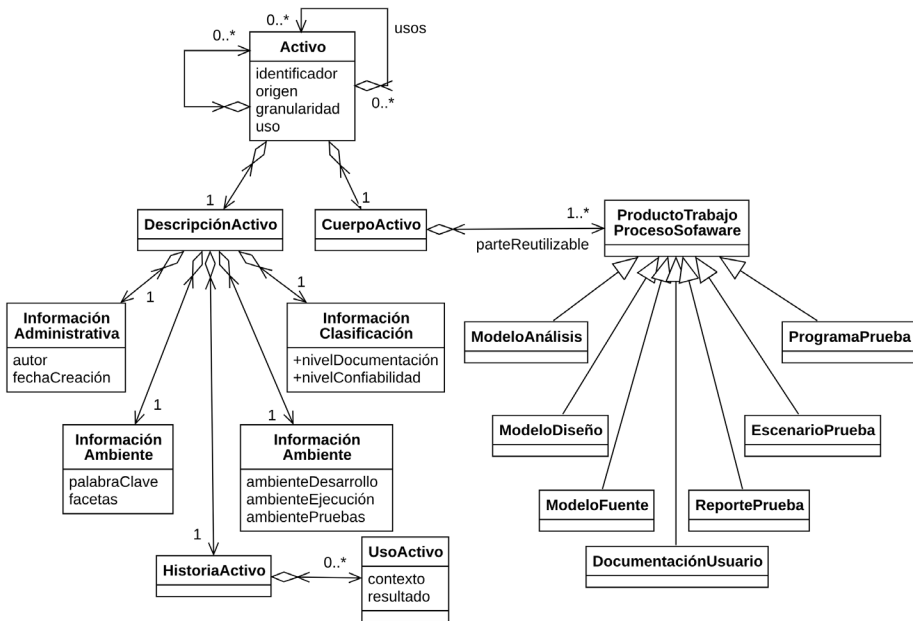


Figura 11.1. Diagrama de clases que representa el modelo de un activo

Fuente: elaboración propia con base en Ezran *et al.* (2012) desarrollado con la herramienta StarUML.

Cualquier activo a ser reutilizado debe respetar algunos criterios generales (Ezran *et al.*, 2012):

- ▶ Cumplimiento de estándares, guías y reglas relacionados con el diseño, la codificación, la documentación, etc.
- ▶ Cumplimiento del proceso de ingeniería de activos con estándares y mejores prácticas.

- Completitud de artefactos e información proporcionada y según el modelo de activo.
- Simplicidad y entendimiento.
- Modularidad de los elementos.

La figura 11.2 presenta los posibles estados por los que pasa un activo durante su ciclo de vida, desde su identificación hasta su validación, clasificación y publicación. Un activo primero es especificado, luego pasa a desarrollarse. Cuando finaliza su desarrollo, se le asigna un nombre (subtitulado). Luego, puede ser certificado, es decir, se valida que cumpla su especificación y ciertos estándares. Después es clasificado de acuerdo con algún sistema de categorías y posteriormente publicado para su respectiva utilización durante la ejecución de proyectos. El activo puede ser evaluado, de tal manera que pueden surgir solicitudes de modificación que permitan generar una nueva versión del activo.

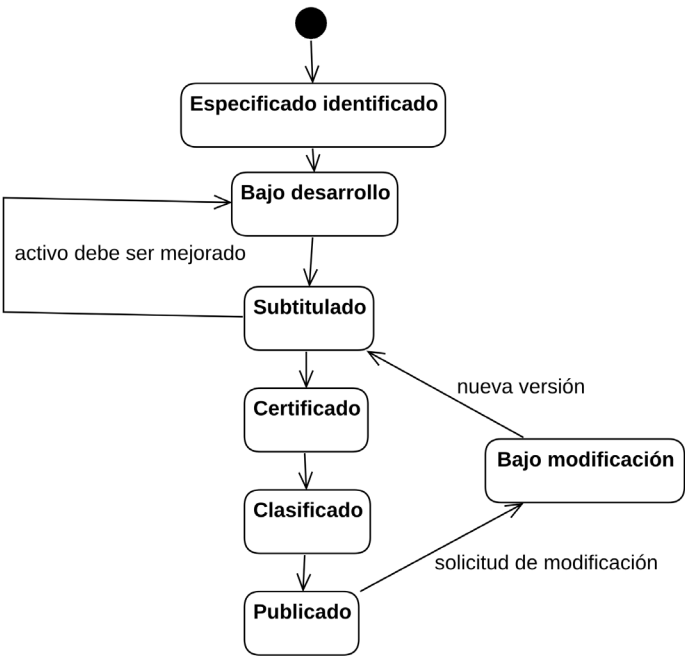


Figura 11.2. Diagrama de máquinas de estado para un activo

Fuente: elaboración propia con base en Ezran *et al.* (2012) desarrollado con la herramienta StarUML.

11.1.3. Repositorios de reúso

A medida que una organización desarrolle una gran cantidad de activos de reúso, es necesario que sean correctamente gestionados de una forma organizada y definida. Un repositorio permitirá almacenar los activos y ofrece las siguientes ventajas:

- ▶ La definición y el reconocimiento de un único lugar para depositar y buscar los activos de la organización.
- ▶ Una forma homogénea de documentar, buscar y contabilizar los activos.
- ▶ Una forma definida de gestionar los cambios y las mejoras a los activos.

La figura 11.3 extiende el modelo de activo de la figura 11.1 para considerar un conjunto de activos agrupados en un repositorio. El modelo extendido incorpora los conceptos de repositorio y catálogo. Un repositorio está compuesto de dos partes: un catálogo y un conjunto de activos. El catálogo está compuesto de un conjunto de descripciones de activos. Un repositorio debe ofrecer algunas de las siguientes funcionalidades:

- ▶ *Identificación y descripción de activos:* Cada activo debe ser identificado de una manera homogénea, por ejemplo, con un identificador y un conjunto estándar de atributos.
- ▶ *Inserción de activos:* Un gestor del reúso debe poseer permisos para insertar un nuevo activo en el repositorio con su respectiva clasificación y versión.
- ▶ *Navegación y búsqueda en el catálogo:* Para encontrar y acceder a la descripción de los activos almacenados. La búsqueda está basada en cadenas de texto que deben contener la descripción de activos específicos.
- ▶ *Historia:* Cada descripción del activo debe tener un registro histórico con su autor, fechas de creación y actualización, y lista de modificaciones que permita realizar un seguimiento de la vida del activo.
- ▶ *Medición:* Capacidad de recolectar datos acerca del activo para generar métricas sobre acceso y uso de los activos almacenados.
- ▶ *Otras funcionalidades:* Como control de acceso, gestión de versiones y control de cambios de los activos almacenados.

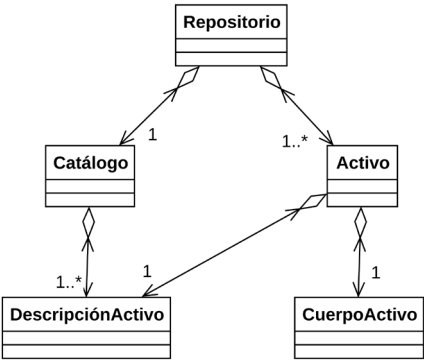


Figura 11.3. Diagrama de clases que representa el modelo de un repositorio
Fuente: elaboración propia con base en Ezran *et al.* (2012) desarrollado con la herramienta StarUML.

El reúso de software diferencia entre tareas relacionadas con la creación de activos reutilizables (ejecutadas por el rol de productor) y tareas relacionadas con la utilización de los activos en una actividad específica (ejecutadas por el rol de consumidor). Las actividades que realizan ambos roles se muestran en la figura 11.4.

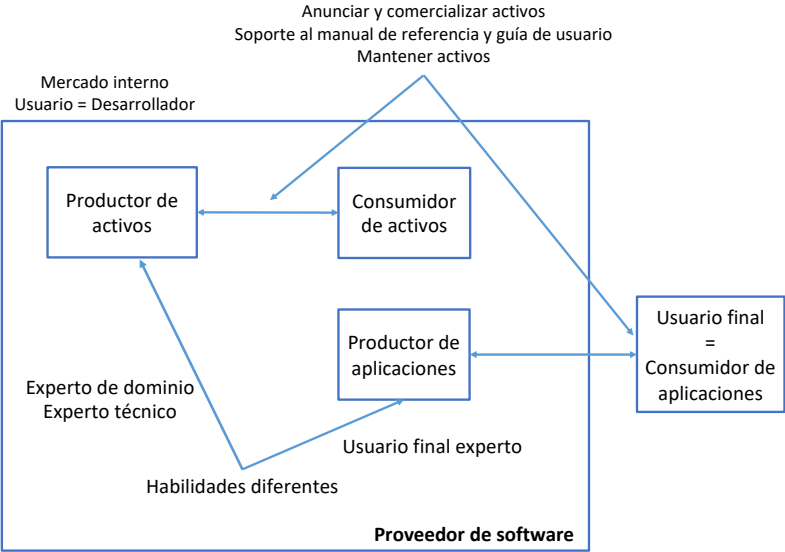


Figura 11.4. Paradigma producto-consumidor de activos
Fuente: elaboración propia con base en Ezran *et al.* (2012).

El enfoque de reúso de procesos de software utilizando repositorios se puede observar en el ejemplo de la figura 11.5 en la que se almacenan activos de proceso reutilizables y mediante un sistema de razonamiento basado en casos se presentan soluciones a un posible problema relacionado con procesos. Esta representación implica un conjunto de propiedades relevantes para describir cada activo y los valores para estas propiedades. El motor de búsqueda recupera casos similares a través de la medición de similitudes.

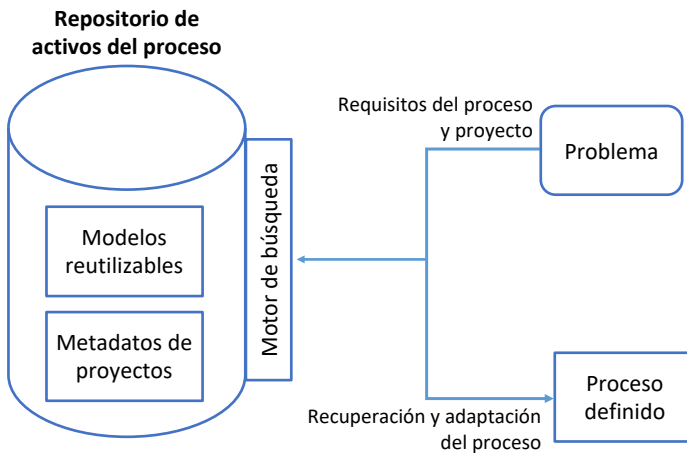


Figura 11.5. Enfoque de reúso basado en repositorios

Fuente: elaboración propia con base en Santos *et al.* (2009).

En la tabla 11.1, se representan diferentes activos que pueden contener un repositorio de procesos de software.

Tabla 11.1. Ejemplos de activos de procesos almacenados en un repositorio

Alcance	Característica	Descripción
Proyecto	Modelo de ciclo de vida	Modelo de ciclo de vida del proyecto como cascada, incremental iterativo, evolutivo, espiral, etc.
	Complejidad	Complejidad del proyecto: alta (funcionalidades críticas y avanzadas), media (funcionalidades factibles) y baja (funcionalidades simples).
	Tamaño	Tamaño del proyecto con respecto a la cantidad de funcionalidades: grande, mediano o pequeño.
	Tamaño del equipo	Número de integrantes del proyecto.
	Tiempo	Duración del proyecto.
	Conocimiento de ingeniería de software	Nivel de conocimiento en ingeniería de software: alto (teoría y práctica), medio (solo teoría), bajo (ningún conocimiento).
	Paradigma de desarrollo	Paradigma de desarrollo de proyectos, como estructurado, orientado a objetos, etc.
Proceso	Modelo de desarrollo	Modelos de desarrollo de software como Rational Unified Process (RUP), eXtreme Programming (XP), <i>scrum</i> , etc.
	Modelo de madurez	Modelo de madurez como Capability Maturity Model Integration (CMMI), MPS.BR, etc.
	Nivel de madurez	Nivel de madurez específico relacionado con el modelo de madurez. Por ejemplo, 1-5 en CMMI.
	Complejidad	Complejidad del proceso basada en los niveles de madurez: alto (niveles avanzados), medio (niveles intermedios), bajo (niveles bajos).
	Proceso	Procesos específicos, como gestión de requisitos, planificación de proyectos, etc.
	Experiencia en utilización del proceso	Experiencia del equipo en el uso del proceso: alto (proceso utilizado en más de 15 proyectos), medio (5-14 proyectos), bajo (0-4 proyectos).

Fuente: elaboración propia con base en Santos *et al.* (2009).

11.1.4. Proceso de reúso

Se han identificado cinco procesos de reúso que son necesarios para caracterizar y practicar el reúso a nivel organizacional (como se observa en la figura 11.6):

- *Producción de activos*: Identificación, desarrollo y clasificación de activos.

- ▶ *Uso de activos*: Localización y reúso real de activos integrándolos en las actividades que están siendo desarrolladas.
- ▶ *Gestión de activos*: Almacenamiento de activos, gestión del repositorio y diseminación de activos.
- ▶ *Mantenimiento y soporte*: Soporte para el uso de activos (para consumidores), soporte metodológico (para productores) y mantenimiento correctivo y evolutivo de activos.
- ▶ *Gestión del reúso*: Monitoreo del reúso en la organización evaluando sus resultados, coordinando actividades de reúso y definiendo y mejorando el proceso de reúso.

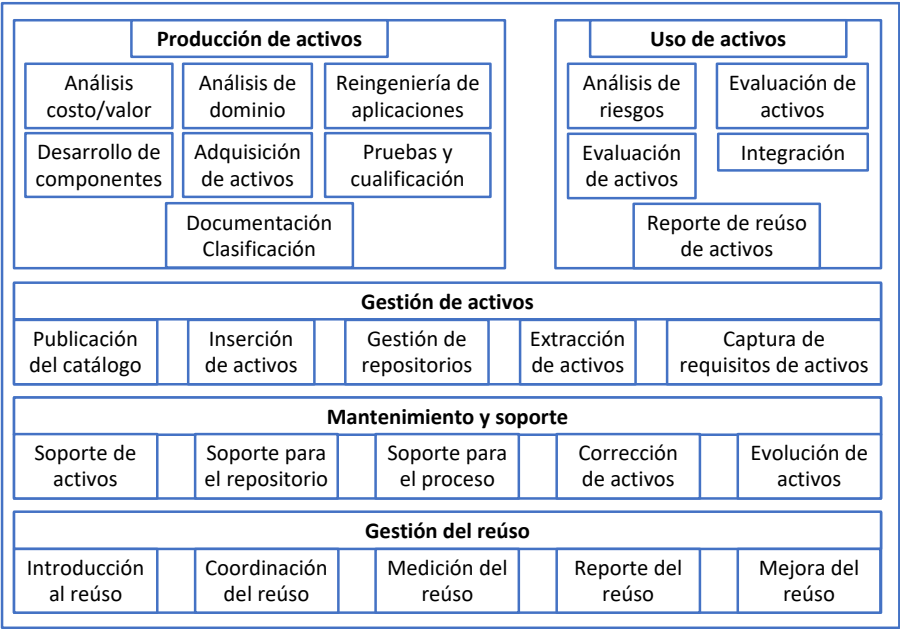


Figura 11.6. Procesos principales de reúso

Fuente: elaboración propia con base en Ezran *et al.* (2012).

El proceso de reúso requiere la definición de un conjunto de roles que cumplan tareas específicas que garanticen la ejecución correcta del proceso. Entre los principales roles orientados al reúso se encuentran:

- ▶ *Administrador del reúso*: Responsable de gestionar las actividades de reúso en la organización.

- *Gestor de la biblioteca de activos*: Gestiona el almacenamiento y configuración de activos.
- *Personal de soporte de reúso*: Apoya los proyectos en la aplicación de procesos y métodos de reúso.
- *Propietario de activos*: Realiza mantenimiento a los activos.
- *Productor de activos*: Encargado de crear los activos.
- *Reutilizador de activos*: Reutiliza realmente los activos existentes en proyectos específicos.

El proceso de reúso se divide comúnmente en dos actividades principales (Anguswamy y Frakes, 2012):

- *Para reutilización*: Consiste en desarrollar software para ser reutilizado en otros sistemas.
- *Con reutilización*: Consiste en desarrollar software con componentes reutilizables.

La figura 11.7 ilustra las actividades para y con reutilización y sus relaciones entre sí.

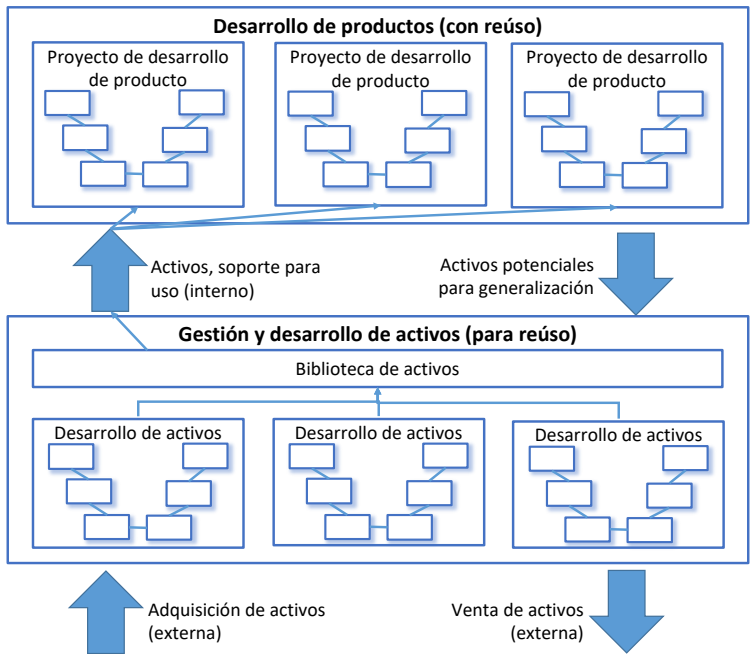


Figura 11.7. Actividades de reúso

Fuente: elaboración propia con base en Kuczka *et al.* (2001).

A continuación, se describen los pasos generales de ambas actividades de reutilización. Los pasos generales a seguir en el desarrollo de software para la reutilización son (Karlsson, 1995; Lim, 1998):

1. Recolectar los requisitos de los posibles reutilizadores.
2. Analizar requisitos y dominio.
3. Seleccionar y priorizar los requisitos.
4. Desarrollar y documentar la solución para los requisitos seleccionados.
5. Evaluar los activos (desarrollar y aplicar pruebas).
6. Almacenar activos con documentación.
7. Dar soporte a los reutilizadores.

Para realizar la actividad de desarrollo con reutilización, se crea un nuevo producto tomando los activos aplicables de la biblioteca de activos, adaptándolos según sea necesario, agregando nuevos activos y ensamblando la colección resultante. Los pasos generales de la actividad de desarrollo con reutilización son (Karlsson, 1995; Lim, 1998):

1. Encontrar los activos candidatos para la reutilización de la biblioteca de activos.
2. Evaluar los activos candidatos con respecto a los requisitos.
3. Seleccionar los activos que se reutilizarán.
4. Adaptar/modificar los activos si es necesario.
5. Integrar activos en el producto.
6. Probar el producto de software.
7. Generar retroalimentación sobre el desarrollo para reutilización.

► 11.2. Reúso del proceso de software

El reúso del proceso de software se puede definir como “el uso de una descripción de proceso en la creación de otras descripciones de proceso” (Hollenbach y Frakes, 1996). El reúso del proceso de software consiste entonces en diseñar procesos a partir de componentes de proceso reutilizables y otros artefactos para cumplir con los requisitos específicos del proceso (Forrester, 2006). Al reutilizar un proceso, se está realizando una reiteración de un conjunto de elementos de un proceso ya ejecutado a un nuevo ambiente.

El modelo 3C de diseño de la reutilización proporciona un marco de trabajo para el diseño de activos reutilizables. El modelo indica tres aspectos de un componente reutilizable (Latour *et al.*, 1991):

- *Concepto*: Especifica la semántica abstracta del componente.
- *Contenido*: Especifica la implementación del componente.
- *Contexto*: Especifica el entorno necesario para utilizar el componente.

La tabla 11.2 muestra un ejemplo del modelo 3C aplicado a un componente de software. Análogamente, un proceso reutilizable tendrá por concepto una especificación informal de su información general, una descripción del cliente del proceso y una descripción de la interfaz del proceso. El contenido incluirá la descripción del procedimiento utilizando tanto una representación textual como gráfica. El contexto del proceso reutilizable se definirá en una descripción contextual (Hollenbach y Frakes, 1996).

Tabla 11.2. Ejemplos del modelo 3C

Aspecto	Componente software	Proceso
Concepto	Tipo abstracto de dato	Especificación informal de:
		• Información general • Descripción del cliente • Descripción de interface
Contenido	Implementado en lenguaje C con una lista enlazada	Descripción procedimental (gráfica y textual)
Contexto	Sistema operativo = Unix, Hardware = Intel, compilador = ANSI C	Descripción contextual

Fuente: elaboración propia con base en Hollenbach y Frakes (1996).

La reutilización sistemática es la práctica de la reutilización de acuerdo con un proceso consistente y repetible. La práctica de la reutilización sistemática requiere un enfoque en el uso de principios de ingeniería para todos los activos de proceso reutilizados involucrados en el desarrollo. Los principales beneficios obtenidos al ofrecer la reutilización sistemática son (Institute of Electrical and Electronics Engineers [IEEE], 2010):

- Aumenta la productividad del equipo de desarrollo de software.
- Acorta el tiempo de desarrollo y mantenimiento de software.

- ▶ Reduce la duplicación de esfuerzos.
- ▶ Traslado de personal, herramientas y métodos más fácilmente entre proyectos.
- ▶ Reduce los costos de desarrollo y mantenimiento de software.
- ▶ Produce productos de software de mayor calidad.
- ▶ Aumenta la confiabilidad del software.
- ▶ Mejora la interoperabilidad y confiabilidad del software.

El reúso de procesos de software, así como el reúso de software, puede reducir costos y tiempos necesarios para entregar software en comparación con productos desarrollados de forma independiente. Además de los artefactos relacionados con el producto desarrollado, el conocimiento sobre el dominio de la aplicación, la asignación de recursos y la programación de actividades realizadas debe almacenarse para definir una base de conocimiento potencialmente reutilizable para desarrollos posteriores.

Por tanto, la reutilización de procesos de software es importante para (Hollenbach y Frakes, 1996):

- ▶ Difundir las experiencias de proyectos anteriores.
- ▶ Reducir los costos y el tiempo de desarrollo.
- ▶ Aumentar la calidad y adecuación de los procesos generados.
- ▶ Promover la reutilización del conocimiento experto.
- ▶ Hacer la definición del proceso accesible para personas menos experimentadas.

En la práctica, la reutilización de procesos tiene dos enfoques (Chrissis *et al.*, 2006):

- ▶ Adaptación del proceso de un proyecto de software al proceso estándar de la organización: Este enfoque se revisará en el apartado 11.7.
- ▶ Composición del proceso: Sintetiza el proceso que se utilizará para llevar a cabo un proyecto determinado basado en unidades más pequeñas y reutilizables, conocidas como elementos o componentes del proceso. Este enfoque se revisará en el apartado 11.4.1.

▶ 11.3. Estándar de reúso

La reutilización exitosa requiere la integración de actividades de reutilización con los procesos del ciclo de vida utilizados para crear los activos

de reutilización asociados con el desarrollo de software. A menos que la reutilización se defina explícitamente en los procesos del ciclo de vida del software, una organización no podrá explotar repetidamente las oportunidades de reutilización en múltiples proyectos o productos de software (IEEE, 2010).

El estándar IEEE 1517: 2010 Tecnología de la información - Proceso del ciclo de vida de software y sistemas - Proceso de reúso es una extensión de la norma ISO/IEC 12207 enfocándose en el reúso dentro del ciclo de vida del software. Este estándar proporciona la base para prácticas que permiten la incorporación de la reutilización en los procesos del ciclo de vida del software. El estándar puede usarse para (IEEE, 2010):

- Adquirir, suministrar, desarrollar, administrar y mantener activos reutilizables.
- Adquirir, suministrar, desarrollar, operar y mantener productos de software que se construyan en su totalidad o en parte con activos reutilizables.
- Gestionar y mejorar los procesos del ciclo de vida de la organización con respecto a la práctica de la reutilización.
- Facilitar el uso de activos reutilizables para desarrollar productos de software.

El documento del estándar está organizado en seis secciones, como se puede observar en la figura 11.8.

La estructura del estándar consta de cuatro categorías relacionadas con el reúso enfocadas en el ciclo de vida del software (IEEE, 2010):

- Desarrollo, operación y mantenimiento de productos de software utilizando activos reutilizables.
- Desarrollo y mantenimiento de activos reutilizables.
- Gestión del reúso.
- Gestión de activos reutilizables.

Además, el estándar describe la integración de la reutilización en los procesos del ciclo de vida del sistema. Las actividades y tareas de reutilización se integran con las de los procesos del ciclo de vida del sistema descritos en el estándar IEEE 12207 (IEEE, 2010):

- a. *Procesos de acuerdo*: Incluye los procesos de adquisición y suministro.

- b. *Procesos de activación de proyectos organizacionales*: Incluye los procesos de gestión del modelo de ciclo de vida, gestión de infraestructura, gestión de la cartera de proyectos y gestión de calidad.
- c. *Procesos del proyecto*: Incluye los procesos de planificación del proyecto, evaluación y control del proyecto, y gestión de decisiones.
- d. *Procesos técnicos*: Incluye los procesos de definición de requisitos de los *stakeholders*, análisis de requisitos del sistema, diseño arquitectónico del sistema, integración del sistema, prueba de calificación del sistema, instalación del software, aceptación de software, operación del software, mantenimiento de software y eliminación de software.

Además, el estándar define las actividades y tareas de reutilización requeridas para desarrollar, operar y mantener productos o servicios de software independientes o sistemas de software. Estas actividades y tareas de reutilización se integran en los siguientes procesos del ciclo de vida del software **IEEE 12207**: implementación de software, soporte de software y reutilización de software.

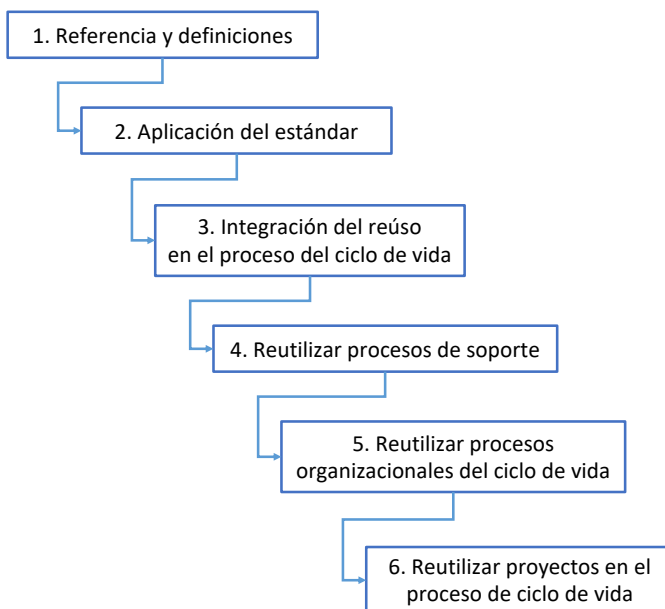


Figura 11.8. Estructura organizacional del estándar IEEE 1517: 2010

Fuente: elaboración propia.

► 11.4. Componentes del proceso de software

La reutilización basada en componentes es el núcleo de la ingeniería de software basada en componentes. Esta área es muy prometedora en la que se identifica, construye, cataloga y publica un conjunto de componentes de software en un dominio particular de aplicación (Pressman, 2005). El objetivo es migrar los conceptos de componentes de software al dominio de los procesos de software a través de componentes de procesos y luego generalizar su aplicación en líneas de procesos, arquitectura del proceso de software y patrones de proceso.

11.4.1. Componentes de proceso

Un componente de software se puede definir como una unidad de software independiente y portable que está completamente definida y al que se accede a través de un conjunto de interfaces (Sommerville, 2015). De igual manera, un componente de proceso puede considerarse una encapsulación de información del proceso y comportamientos en un nivel de granularidad dado (Gary y Lindquist, 1999). Los componentes del proceso también se denominan elementos del proceso. Los componentes de proceso son elementos reutilizables que se pueden conectar para crear procesos nuevos y más complejos (Barreto *et al.*, 2011). Por tanto, un proceso puede tratarse como la integración de los componentes del proceso en diferentes niveles de granularidad (Fusaro *et al.*, 1998).

Por tanto, un componente es algo relevante para ser reutilizado en otras definiciones de proceso y encapsula una cantidad de información acerca del proceso, como actividades, productos de trabajo requeridos y producidos, actores, entre otros. Un componente de proceso tiene los siguientes elementos (Magdaleno *et al.*, 2015):

- *Una interfaz requerida*: Identifica los artefactos que el componente utiliza para realizar su trabajo.
- *Una interfaz proporcionada*: Identifica los artefactos producidos por el componente después de su ejecución.

Un ejemplo de componente de proceso, en este caso, una actividad específica como la reunión diaria de *scrum*, se proporciona en la tabla 11.3.

Tabla 11.3. Estructura y ejemplo de componente del proceso

Componente de proceso, reunión diaria	
Rol	Equipo del proyecto
Opcionalidad	Obligatorio
Interfaces requeridas	<i>Product backlog</i>
Interfaces proporcionadas	Gráfica de avance del <i>sprint</i> (<i>burndown chart</i>)

Fuente: elaboración propia con base en Magdaleno *et al.* (2015).

Los componentes de proceso tienen interfaces de entrada y de salida, definidas por precondiciones y poscondiciones. Dentro de un componente del proceso, las actividades del proyecto y los elementos del subproceso se clasifican como variantes e invariantes.

Las actividades variantes pueden modificarse, incluso eliminarse del elemento del proceso, sin afectar significativamente los resultados del componente del proceso. Por otro lado, las actividades invariantes pueden ajustarse, pero no pueden modificarse de manera extensa, ya que el componente del proceso no puede garantizar la producción de los resultados prometidos. Además, incorporan comentarios de una base de conocimiento o experiencia que tiene información sobre planes de proyectos anteriores. La figura 11.9 muestra las interacciones generales de los componentes de proceso.

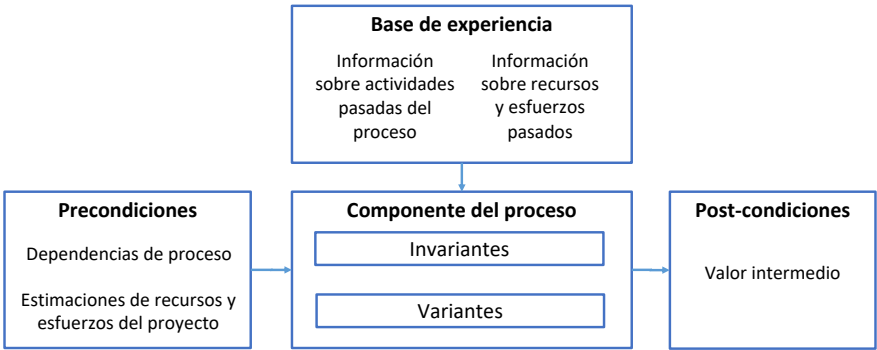


Figura 11.9. Interacciones en componentes de proceso

Fuente: elaboración propia con base en (Bhuta *et al.*, 2006).

Los desarrolladores de procesos deben diseñar los componentes de proceso en atención a los siguientes atributos de calidad (Bhuta *et al.*, 2006):

- *Adaptabilidad*: Es esencial que se puedan modificar los componentes para satisfacer las necesidades específicas del proyecto. Se pueden agregar, modificar y eliminar actividades variables con un impacto mínimo en el componente del proceso.
- *Facilidad de comprensión*: Cada elemento del proceso debe tener información mínima sobre qué hace el componente, valor agregado que proporciona, cómo se puede ejecutar y qué recursos se requieren para ejecutarlo. Además, la ayuda puede contener la información de contexto sobre los proyectos en los que el componente del proceso se ha ejecutado con éxito y el esfuerzo promedio o porcentual del proyecto requerido para implementarlo.
- *Independencia*: Los componentes del proceso deben ser independientes para que sea posible reutilizarlos sin adoptar otros no relacionados.
- *Robustez*: Las actividades dentro de los componentes deben ser correctas y tolerantes a fallos para manejar las discrepancias en sus condiciones previas.

11.4.2. Líneas de procesos de software

Aunque el uso de componentes del proceso ofrece muchas ventajas, comenzar con pequeñas unidades para componer procesos grandes puede ser insuficiente debido a la granularidad fina. Se recomienda el uso de estructuras más grandes reutilizables, como líneas de proceso de software (Barreto *et al.*, 2011).

El concepto de línea de proceso de software (**LPS**) fue propuesto por H. D. Rombach y O. Kaufman, y es una técnica para la reutilización sistemática de procesos. Surge de la idea de las líneas de producto de software. Una línea de productos es un conjunto de productos de software que comparten características comunes que satisfacen las necesidades específicas y que se desarrollan utilizando un conjunto común de activos centrales desde la perspectiva de la optimización general (Birk *et al.*, 2003).

Una **LPS** posee un conjunto de procesos similares dentro de un dominio particular y es una aplicación de la idea de líneas de productos a los procesos. Por consiguiente, una línea de procesos de software es un conjunto de procesos similares en un dominio particular o para un propó-

sito particular, que tiene características comunes y variabilidad controlada. Además, se basa en activos de proceso comunes y reutilizables (Washizaki, 2006). Las **LPS** preparan los procesos de una organización para satisfacer las necesidades futuras anticipadas mediante la determinación de un núcleo de proceso común y partes de proceso variables que satisfagan necesidades específicas (Armbrust *et al.*, 2009).

Los procesos que conforman una línea de proceso satisfacen un dominio o propósito particular y se derivan, de manera prescrita, de una estructura central reutilizable. Esa estructura se llama arquitectura de línea de proceso de software (**ALPS**). La **ALPS** se puede definir como “una estructura de proceso que refleja la similitud y la variabilidad en una colección de procesos que conforman una línea de proceso desde la perspectiva de la optimización general” (Washizaki, 2006). La **ALPS** sirve como base para crear instancias de todos los procesos que forman parte de la línea de proceso. En la figura 11.10, se presenta una infraestructura de línea de procesos en que una arquitectura de línea de procesos puede desarrollarse a partir de varias líneas específicas.

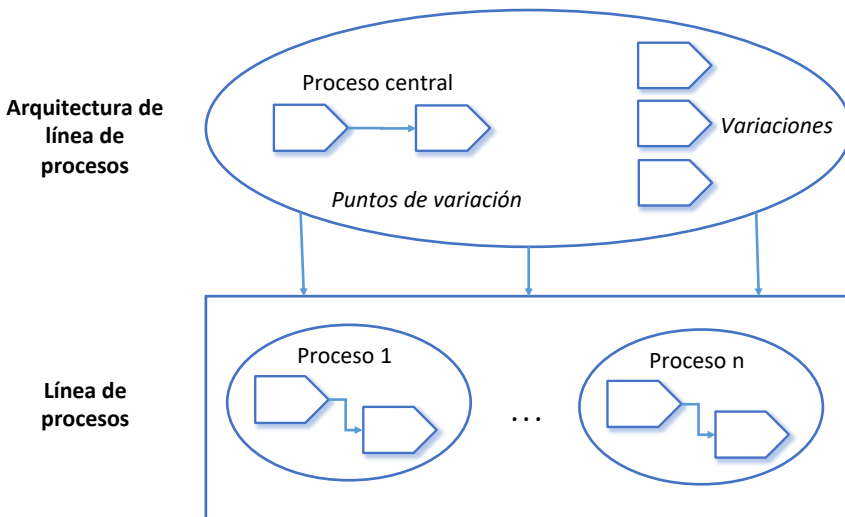


Figura 11.10. Infraestructura de línea de procesos

Fuente: elaboración propia con base en Washizaki (2006).

La similitud y la variabilidad en una **ALPS** se pueden representar como (Washizaki, 2006):

- *Similitud*: Está representada por el proceso central que se compone de las partes comunes del conjunto de procesos.
- *Variabilidad*: Está representada por los puntos de variación y las variantes del proceso:
 - *Puntos de variación*: Actividades (o entradas/salidas o roles que afectan las actividades) que se pueden cambiar de acuerdo con las características de un proyecto específico.
 - *Variantes de proceso*: Actividades candidatas concretas (o entradas/salidas, etc.) que se aplican a los puntos de variación.

Los puntos de variación típicos de los elementos genéricos del proceso son el dominio de la aplicación, los objetivos del proyecto y la competencia del equipo. Los componentes de proceso reutilizables de una línea de proceso requieren una mejora y evolución continuas (Forrester, 2006).

Las características principales de una **ALPS** son (Rombach, 2006):

- *Dos procesos de desarrollo separados*: El proceso de ingeniería de dominio que crea procesos para la reutilización y el proceso de ingeniería de aplicaciones que desarrolla procesos específicos del proyecto.
- *Un repositorio de procesos*: Los procesos reutilizables en todos los niveles de abstracción están disponibles.
- *Un proceso de reutilización sistemática*: La elección de los componentes del proceso está limitada por cada elección predefinida de variabilidades.
- *Un proceso sistemático de gestión de procesos*: Para cada comportamiento inesperado del proceso, se decidirá si esta excepción se incluirá en el proceso genérico o no.

El concepto de línea de procesos se generaliza a ingeniería de línea de procesos de software, la cual se puede definir como “un sistema de enfoques estratégicos y sistemáticos interrelacionados para construir, aplicar y gestionar líneas de proceso” (Washizaki, 2006). La ingeniería de línea de procesos de software consta de dos fases (figura 11.11):

- *Ingeniería de dominio*: Reúne los requisitos con la variabilidad para la línea de proceso. La arquitectura de la línea de procesos y las variantes se diseñan e implementan en función de estos requisitos.
- *Ingeniería de aplicaciones*: Los requisitos para un proceso de un proyecto específico se definen reutilizando los requisitos para la línea

de proceso. La arquitectura de la línea de procesos y las variantes crean un proceso específico del proyecto.

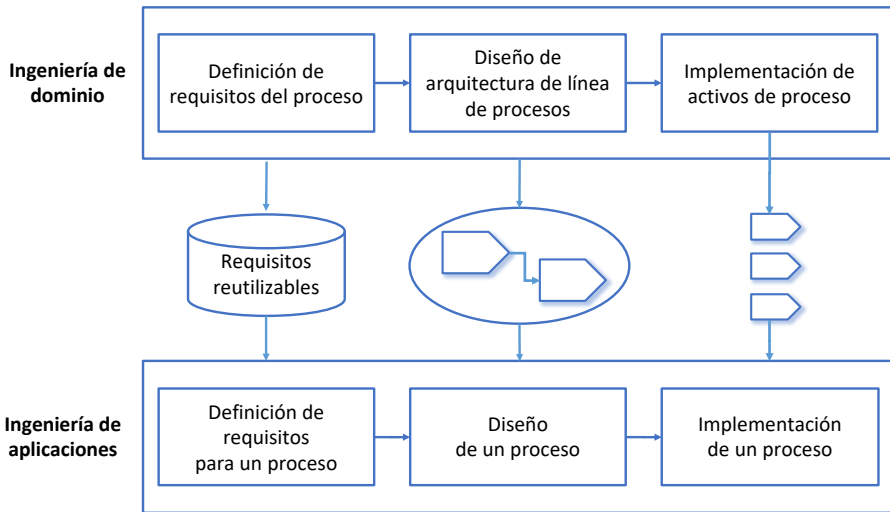


Figura 11.11. Infraestructura de línea de procesos

Fuente: elaboración propia con base en Washizaki (2006).

11.4.3. Arquitectura del proceso de software

Las arquitecturas son descritas en términos de sus componentes constituyentes, cómo esos componentes están conectados y la naturaleza de esas conexiones.

Una arquitectura de procesos de software se puede definir como “un marco conceptual para incorporar, relacionar y adaptar de manera consistente los elementos del proceso en procesos ejecutables. A menudo, se necesita una arquitectura de proceso cuando un proceso debe relacionarse con otros procesos existentes o futuros” (Feiler y Humphrey, 1993).

Como se puede observar en la figura 11.12, una organización desarrolladora de software que adopta el enfoque basado en procesos puede dividirse en tres arquitecturas (Pesantes *et al.*, 2012):

- *Arquitectura de negocios:* Se compone de una arquitectura organizacional, un modelo de negocios, objetivos y estrategias de negocio, modelos de mejora de procesos, modelos de ciclo de vida, guías de

adaptación y una arquitectura de procesos de software. Esta última arquitectura se divide en cinco vistas:

- *Vista conceptual*: Define la estructura de un componente de proceso como un conjunto de actividades (variantes e invariantes), precondiciones (artefactos e información para ejecutar elementos de proceso) y poscondiciones (resultados obtenidos de la ejecución de componentes de proceso).
 - *Vista de referencia*: Determina los modelos de mejora del proceso y mejores prácticas que se utilizarán.
 - *Vista operativa*: Define los procesos de software estándar.
 - *Base de conocimiento*: Para compartir información de proyectos pasados y otros activos de proceso.
 - *Proyecto de software*: Es un conjunto de procesos adaptados a un proyecto específico.
- *Arquitectura de aplicación*: Se compone de dos partes:
- *Panorama de aplicaciones*: Respalda la automatización de los procesos de negocio.
 - *Panorama de repositorio de datos*: Almacenamiento físico de los datos relevantes de la organización.
- *Arquitectura de infraestructura*: Comprende la infraestructura de *software*, *hardware* y red requerida para las operaciones de las aplicaciones.

En el reuso de procesos, la arquitectura de procesos desempeña un papel importante. Entre los propósitos de las arquitecturas de procesos de software se incluyen (Li, 2008):

- Describir los componentes significativos, la estructura, las relaciones e interfaces internas y externas.
- Definir caminos de evolución elegantes y las variantes de reutilización requeridas.
- Guiar la selección, adaptación, composición y unión de componentes.
- Permitir un ensamblaje suave de los componentes y conectarlos con el entorno circundante y proporcionar compatibilidad en múltiples instancias.

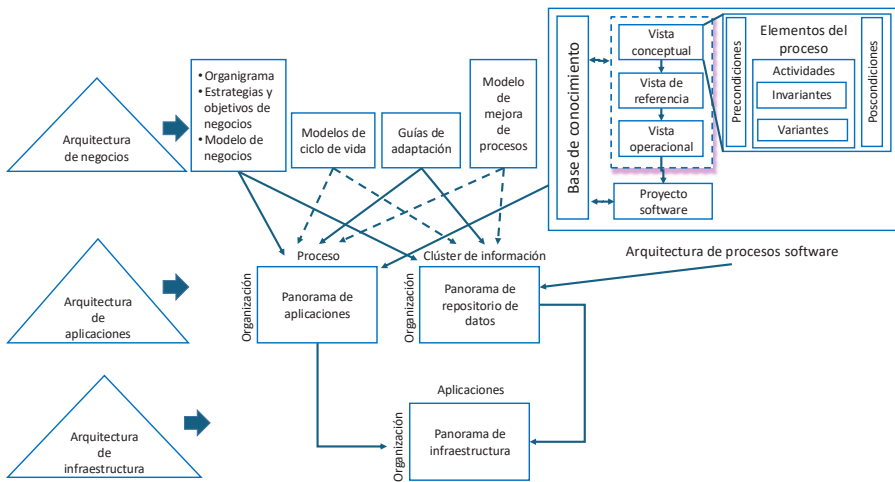


Figura 11.12. Arquitectura de procesos de software

Fuente: elaboración propia con base en Pesantes *et al.* (2012).

La arquitectura del proceso y la integración de los componentes del proceso son los mecanismos críticos para garantizar que los procesos personalizados se unan y formen un panorama de procesos sin problemas operativos.

11.4.4. Patrones de proceso

Un patrón describe un problema que ocurre una y otra vez en un entorno, describiendo el núcleo de la solución a este problema, de tal forma que se puede usar esta solución muchas veces sin hacerlo dos veces de la misma forma (Alexander, 1979).

En el campo de la ingeniería de procesos de software, los patrones representan soluciones a problemas que surgen en el desarrollo de aplicaciones, permitiendo su reutilización como respuesta a conflictos similares surgidos en contextos diferentes (Medina Domínguez, 2010). Los patrones han surgido desde el paradigma orientado a objetos como una técnica para documentar soluciones de diseño que incorporan las mejores prácticas de la industria y la academia. Los patrones describen un contexto de aplicación, un problema de diseño que ocurre en ese contexto y una solución al problema que ha sido probada efectivamente por la experiencia

(Alexander, 1979). Algunos autores extienden los elementos de un patrón con nombre y clasificación, propósito, alias, motivación, aplicabilidad, estructura, participantes, colaboraciones, consecuencias, aspectos de implementación, códigos de ejemplo, usos conocidos y patrones relacionados (Gamma *et al.*, 1994).

Un patrón de proceso captura un modelo de proceso que representa una solución que se puede aplicar en un contexto dado para resolver un problema recurrente de modelado de procesos. Se distinguen dos tipos de problemas de modelado de procesos a resolver por medio de patrones (Tran *et al.*, 2011):

- *Definir un componente de proceso*: Se refiere a cómo definir los detalles de un componente de proceso.
- *Organizar componentes de proceso*: Se refiere a cómo organizar un grupo de componentes de proceso para satisfacer una situación especial o ciertas restricciones.

Los diferentes problemas de modelado de procesos requieren patrones de proceso en diferentes niveles de abstracción. Por tanto, los patrones de proceso se clasifican en tres tipos (Tran *et al.*, 2005):

- *Patrones de proceso abstractos*: Capturan estructuras de proceso genéricas aplicables a cualquier proceso, como plantillas para modelar diferentes flujos de trabajo de proceso.
- *Patrones de proceso generales*: Capturan elementos de proceso de métodos o técnicas de desarrollo comprobados que pueden aplicarse a diferentes dominios de aplicación, como la técnica de inspección para verificar artefactos de software.
- *Patrones de procesos concretos*: Capturan experiencias prácticas concretas aplicables a un proceso particular para producir un tipo específico de producto, como un proceso para inspeccionar códigos de Java adoptados por un equipo en particular.

La figura 11.13 muestra un ejemplo de los tres tipos de patrones de proceso descritos.

El patrón “Ciclo iterativo” (figura 11.13a) describe una plantilla para gestionar una lista en una secuencia simple; es abstracto porque la semántica específica del elemento “Realizar una tarea” permanece indefinida. El patrón “Inspección de Fagan” en la figura 11.13b describe un método para inspeccionar un artefacto de software; es general porque los detalles de sus pasos no se pueden definir sin especificar el tipo concreto del artefacto. Finalmente, el patrón “Refactorización C#” (figura 11.13c) proporciona

pasos concretos usando el entorno Visual Studio para refactorizar códigos en C# (Tran *et al.*, 2011).

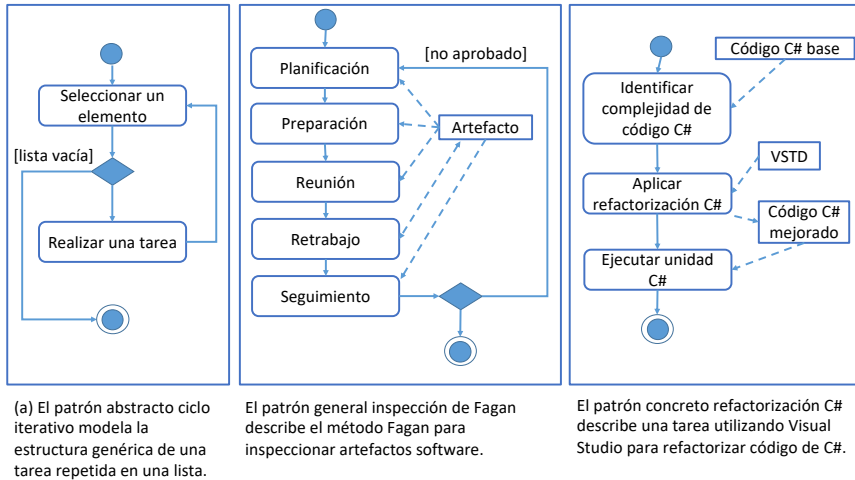


Figura 11.13. Patrones de proceso en diferentes niveles de abstracción

Fuente: elaboración propia con base en Tran *et al.* (2011).

También existe un metamodelo que proporciona los conceptos necesarios para describir un patrón de proceso y sus componentes internos y la reutilización de patrones de proceso en modelos de proceso. Los principales elementos del metamodelo (figura 11.14) son un patrón de proceso que se compone de un problema, una solución y un contexto.

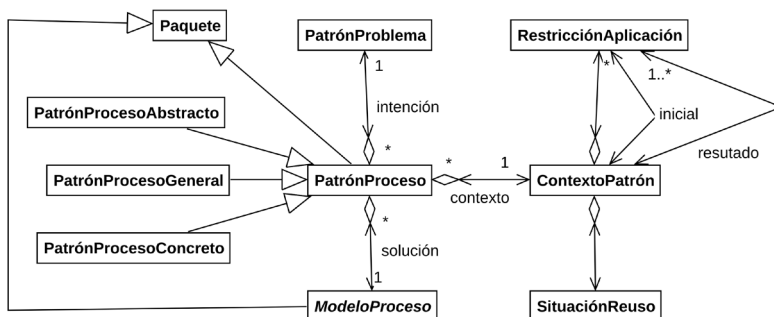


Figura 11.14. Metamodelo para patrones de proceso

Fuente: elaboración propia con base en Tran *et al.* (2011) desarrollado con la herramienta StarUML.

En la figura 11.15, se presenta un modelo estructural que refleja la composición de un elemento del proceso y las relaciones entre elementos del proceso. Los bordes estructurales muestran las diferentes relaciones estáticas definidas entre dos elementos de proceso.

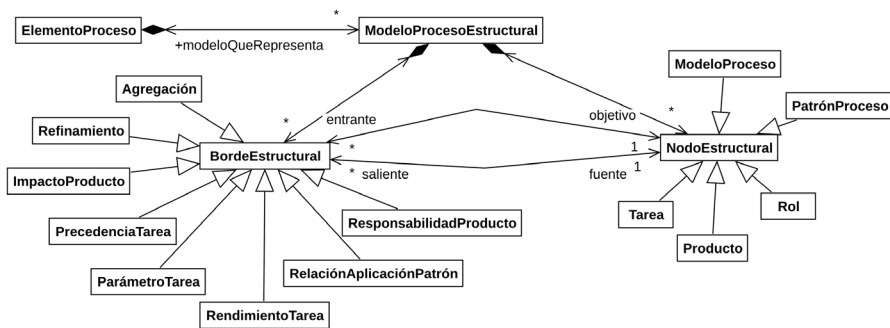


Figura 11.15. Sintaxis del modelo estructural

Fuente: elaboración propia con base en Tran *et al.* (2011) desarrollado con la herramienta StarUML.

La figura 11.16 muestra tres relaciones principales del proceso: *ParámetroTarea* permite representar los productos manipulados (entrada y salida) de una tarea; *RendimientoTarea* especifica los roles que participan en la ejecución de una tarea como ejecutor o asistente; *ResponsabilidadProducto* vincula un rol con los productos de los que es responsable. El modelo de comportamiento de los patrones de procesos se puede desarrollar por medio de diagramas de actividad UML.

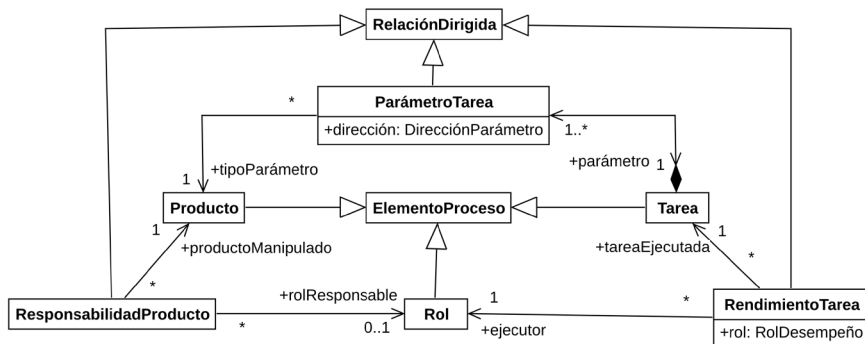


Figura 11.16. Relaciones entre elementos de proceso

Fuente: elaboración propia con base en Tran *et al.* (2011) desarrollado con la herramienta StarUML.

Con respecto a la descripción del patrón, se han definido diferentes formatos que especifican el patrón por medio de una plantilla dividida en secciones fijas y una estructura uniforme que facilita su aprendizaje, comparación y uso. Gnatz *et al.* (2003) definieron una plantilla denominada plantilla para patrones de proceso de desarrollo software, cuyos contenidos se presentan en la tabla 11.4.

Tabla 11.4. Plantilla de patrones de proceso

Sección	Descripción
Nombre	Nombre del patrón de proceso de desarrollo
Alias	Otros nombres para el patrón
Autor	El nombre del autor del patrón
Intención	Resumen de la intención y lógica o heurísticas del patrón
Problema	Elemento que dirige el patrón, incluso una discusión de restricciones asociadas proporcionando un escenario o ejemplo del mundo real
Contexto	Situación o estado de un proyecto en el que el patrón es aplicable
Solución	Estado o configuración del sistema después de haber aplicado el patrón
Consecuencias	Beneficios que el patrón proporciona y potenciales problemas
Usos conocidos	Descripción de ocurrencias conocidas del patrón y su aplicación
Patrones relacionados	Referencias a patrones que resuelvan problemas similares

Fuente: elaboración propia con base en Gnatz *et al.* (2003).

► 11.5. Métricas de reúso

Se pueden aplicar métricas para evaluar los programas de reúso del proceso que tiene una organización. Las entidades relacionadas con la medición del reúso de software son (Fenton y Bieman, 2014):

- *Productos*: En particular, productos de trabajo específicos de reúso (activos, repositorio).
- *Procesos*: En particular, procesos específicos de reúso (producción, uso, gestión de activos).
- *Recursos*: Todos los roles orientados al reúso.

En la tabla 11.5, se presenta un ejemplo de un programa de métricas de reúso del proceso de software basado en **GQM**.

Tabla 11.5. Ejemplo de programa de métricas de reúso de procesos

Objetivo	Propósito	Medida
O1	Caracterizar la utilización del repositorio desde el punto de vista del gestor del reúso en el contexto de una división de la empresa	M1.1. Número de activos en el repositorio
		M1. 2. Número de accesos al repositorio por periodo
		M1.3. Número de recuperaciones desde el repositorio por periodo
		M1.4. Número de búsquedas fallidas por periodo
		M1.5. Número acumulativo de reúsos por activo
		M1.6. Número acumulativo de reúsos para todos los activos en el repositorio
O2	Evaluar la extensión del reúso de un proyecto desde el punto de vista del gestor del proyecto y gestor del reúso en el contexto de una división de la empresa	M2.1. Nivel de reúso por proyecto
		M2.2. Número de accesos al repositorio por proyecto
		M2.3. Número de recuperaciones desde el repositorio por proyecto
O3	Evaluar la variación en el costo de los activos reutilizables desde el punto de vista del gestor sénior en el contexto de una división de la empresa	M3.1. Costo relativo de reúso
		M3.2. Costo relativo de escribir para reúso
O4	Evaluar los aspectos económicos del proceso de software que incluye reúso desde el punto de vista del gestor sénior en el contexto de la empresa total	M4.1. Umbral de pago
		M4.2. Retorno de inversión de la organización
O5	Evaluar la calidad de los activos reutilizables desde el punto de vista de los reutilizadores en el contexto de un proyecto	M5.1. Confiabilidad
		M5.2. Facilidad de entendimiento
O6	Predecir la reutilización de activos desde el punto de vista del gestor del reúso en el contexto de una división de la empresa	M6.1. Confiabilidad
		M6.2. Facilidad de entendimiento
		M6.3. Completitud funcional
		M6.4. Portabilidad

Fuente: elaboración propia con base en Ezran *et al.* (2012).

► 11.6. Adaptación del proceso de software

No existe un proceso de software único, ya que la idoneidad de un proceso depende de las organizaciones, los proyectos y las características del producto. Además, todas estas características evolucionan continuamente (Hurtado Alegría *et al.*, 2014). Otro desafío es que cada proyecto es único, lo que es paralelo a la afirmación “un tamaño no puede adaptarse a todo”.

Se debe asegurar que no existan diferencias fundamentales entre el proceso definido y el proceso ejecutado, es decir, que el proceso ejecutado se ajusta apropiadamente al proceso definido. Muchas organizaciones prefieren establecer unas guías con las cuales los proyectos definen sus propios procesos, es decir, lo adaptan a las particularidades de cada proyecto. Por ejemplo, dichas guías pueden agregar los roles de las partes interesadas, eliminar algunas de las actividades menos importantes y modificar los artefactos y procesos. Otras razones que argumenta el modelo **CMMI** para utilizar las guías de adaptación son (SEI, 2010):

- Adaptar el proceso a una nueva línea de producto o entorno de trabajo.
- Elaborar la descripción del proceso de tal manera que pueda ejecutarse el proceso definido.
- Personalizar el proceso para una aplicación o una clase de aplicaciones similares.

La adaptación del proceso (*process tailoring*) se define formalmente como “el acto de ajustar la definición o particularizar los términos de una descripción general a un entorno alternativo (menos general)” (Ginsberg y Quinn, 1995). En el dominio de los procesos de software, consiste en la adaptación del conjunto estándar de procesos de una organización a contextos de proyectos específicos en atención a variables del contexto, como tamaño del equipo, objetivos del proyecto y tecnologías utilizadas (Kalus y Kuhrmann, 2013). Por tanto, la adaptación del proceso altera o adapta una descripción del proceso para un fin particular. Si se utiliza el modelo **CMMI-DEV**, la adaptación está basada en un conjunto de guías que permiten adaptar el proceso a las características de cada proyecto.

La adaptación de procesos de software forma parte de los enfoques de mejora de estos. Las organizaciones, independiente de sus tamaños, adaptan un modelo de proceso de acuerdo con sus necesidades. Incluso un modelo de proceso adecuado para un proyecto puede no ser beneficioso para otro proyecto en la misma organización (Akbar *et al.*, 2011). La adap-

tación del proceso de software es una tendencia emergente debido al factor de incertidumbre involucrado en la selección de un proceso adecuado de acuerdo con los requisitos del proyecto y la organización.

Portanto, la adaptación del proceso debe generar un proceso de software específico del proyecto cada vez que se ejecuta y también puede considerarse una actividad de reutilización del proceso de software estándar (Gallina *et al.*, 2014).

Las guías de adaptación detallan cómo se puede adaptar el proceso estándar de la organización para producir el proceso de software definido del proyecto. Estas guías definen el nivel de formalidad y entregables que se producirán para las diversas categorías de proyectos en la organización.

En el modelo **CMMI-DEV**, en el área de proceso “Definición del proceso de la organización” del nivel 3, se incluye la práctica específica **SP 1.3** Establecer los criterios y las guías de adaptación. Esta práctica enfatiza que las guías de adaptación describen (SEI, 2010):

- Cómo utilizar el conjunto de procesos estándar de la organización y los activos de proceso para crear los procesos definidos.
- Los requisitos que deben satisfacer los procesos definidos.
- Las opciones que se pueden ejercer y los criterios para su selección.
- Los procedimientos que deben seguirse para realizar y documentar la adaptación del proceso.

En la figura 11.17, se presentan para la práctica “**SP 1.3** Establecer los criterios y las guías de adaptación”, sus ejemplos de productos de trabajo y subprácticas correspondientes.

Para la adaptación del proceso en los proyectos, generalmente se definen varias categorías. Cada categoría se define por criterios y el proyecto se incluirá en una de las categorías. Cada categoría de proyecto tiene un conjunto asociado de resultados que reflejan la adaptación del proceso.

La tabla 11.6 proporciona un ejemplo simple de adaptación basado en dos categorías de proyecto (A y B). Cada categoría se define en términos de criterios y tiene un conjunto asociado de entregables.

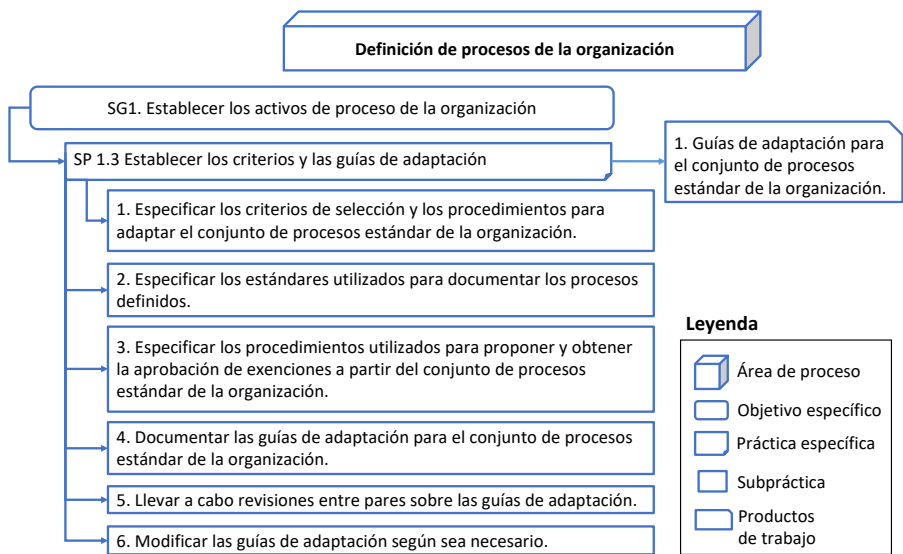


Figura 11.17. Práctica SP 1.3 Establecer los criterios y las guías de adaptación
Fuente: elaboración propia.

Tabla 11.6. Adaptación del proceso para varios proyectos

Factor/tipo		Categoría A	Categoría B
Complejidad	Baja		
Costo	Menor a US\$50.000	Mayor a US\$50.000	
Duración	Menos de 3 meses	Más de 3 meses	
Esfuerzo	Menos de 30 días	Más de 30 días	
Entregables	Plan pequeño de tareas	Plan del proyecto, calendario del proyecto, requisitos de negocio y del sistema, diseño y pruebas	

Fuente: elaboración propia con base en O'Regan (2017).

La implementación del nivel 3 de **CMMI** requiere que cada proceso se defina a nivel organizacional y luego se adapte para producir procesos de software definidos para cada proyecto (figura 11.18).

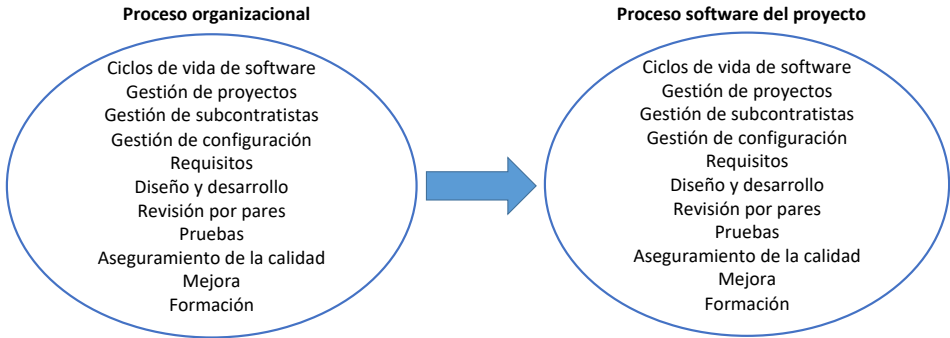


Figura 11.18. Adaptación del proceso para varios proyectos

Fuente: elaboración propia con base en O'Regan (2017).

La adaptación del conjunto de procesos estándar de la organización para producir el proceso de software definido del proyecto permite que este refleje las características únicas del proyecto. Es decir, los proyectos no están obligados a hacer las cosas exactamente de la misma manera en una organización de nivel 3, y los procesos utilizados por los proyectos pueden variar según lo que permita la adaptación. El proyecto se gestiona utilizando el proceso de software definido de este.

Las guías de adaptación deben definir el procedimiento mediante el cual se realiza la adaptación. Una plantilla de adaptación permite documentar la adaptación real realizada por el proyecto. Muestra cómo se seleccionan los procesos y estándares del proyecto del conjunto de procesos estándar de la organización. El ejemplo en la tabla 11.7 incluye entregables obligatorios y opcionales para cada categoría.

Otro enfoque para asumir la adaptación del proceso es basarse en riesgos (Xu y Ramesh, 2008). De acuerdo con los posibles riesgos o desafíos que tiene un proyecto, se pueden identificar estrategias específicas de adaptación del proceso que son efectivas en la gestión de cada tipo de desafío (recursos, comunicación, gestión de requisitos, políticas y técnicas). En la tabla 11.8, se presenta un cuestionario con algunos ejemplos de estrategias sugeridas según el posible riesgo presentado en la organización. El cuestionario evalúa la importancia relativa de varios riesgos que deben abordarse adaptando el proceso de software.

Tabla 11.7. Plantilla de adaptación del proceso

Plantilla	Solicitud de cambio	Proyecto pequeño	Proyecto grande
Plan del proyecto	Obligatorio	Obligatorio	Opcional
Plan pequeño	Obligatorio	Opcional	
Plan de tareas	Opcional		
Requisitos de sistema	Opcional	Opcional	Opcional
Diseño	Obligatorio	Opcional	Opcional
<i>Logs</i> de problemas	Obligatorio	Obligatorio	Opcional
<i>Logs</i> de riesgos	Obligatorio	Obligatorio	Opcional
Lecciones aprendidas	Obligatorio	Obligatorio	Opcional
Plan de pruebas	Obligatorio	Opcional	Opcional
<i>Scripts</i> de pruebas	Obligatorio	Opcional	Opcional
Reportes de pruebas	Obligatorio	Opcional	Opcional

Fuente: elaboración propia con base en O'Regan (2017).

Tabla 11.8. Cuestionario de evaluación de riesgos

Categorías de desafíos	Preguntas de evaluación
Recurso	1. El proyecto tiene un presupuesto adecuado para lograr sus objetivos.
	2. El proyecto tiene el tiempo adecuado para lograr sus objetivos.
	3. El proyecto tiene suficiente personal con habilidades relevantes para lograr los objetivos del proyecto.
	4. El equipo de desarrollo tiene las herramientas necesarias.
	5. El equipo de desarrollo tiene el soporte técnico necesario.
Evaluación total del desafío: 5-11: alto, 12-18: moderado, 19-25: bajo	
Comunicación	1. Los miembros del equipo han trabajado juntos en otros proyectos.
	2. Los miembros del equipo trabajan juntos muy bien.
	3. El tamaño del equipo es pequeño.
	4. La tasa de rotación de personal es probable que sea baja.
	5. La duración del proyecto es corta.
	6. Los <i>stakeholders</i> en el proyecto tienen conocimientos similares.
Evaluación total del desafío: 6-13: alto, 14-22: moderado, 23-30: bajo	

Categorías de desafíos	Preguntas de evaluación
Gestión de requisitos	1. La propuesta inicial del proyecto es clara y concreta.
	2. Solo algunas partes interesadas diferentes están involucradas en este proyecto.
	3. Las partes interesadas clave son fácilmente accesibles.
	4. Los usuarios tienen experiencia relevante con el sistema de información.
	5. Existen sistemas o documentos heredados para que los equipos de desarrollo se familiaricen con el dominio de la aplicación.
	6. Los conductores del negocio del sistema son claros y todos los <i>stakeholders</i> los entienden.
	7. Los analistas tienen mucha experiencia en el dominio de negocio.
	8. Los requisitos para el sistema son estables y no volátiles.
Evaluación total del desafío: 8-18: alto, 19-29: moderado, 30-40: bajo	
Política	1. La alta gerencia está muy comprometida con el proyecto y sus objetivos.
	2. El proyecto tiene claros conductores de negocio.
	3. El proyecto tiene fácil acceso a la alta gerencia.
Evaluación total del desafío: 3-6: alto, 7-11: moderado, 12-15: bajo	
Técnico	1. El equipo de desarrollo está familiarizado con las herramientas utilizadas en el proyecto.
	2. El equipo de desarrollo está familiarizado con la tecnología utilizada en el proyecto.
	3. El equipo de desarrollo tiene experiencia en la metodología utilizada en el proyecto.
	4. El equipo de desarrollo tiene acceso al soporte técnico necesario (como consultores, expertos y sesiones de entrenamiento).
Evaluación total del desafío: 4-9: alto, 10-14: moderado, 15-20: bajo	
Escala 1: muy en desacuerdo; Escala 2: en desacuerdo; Escala 3: neutral; Escala 4: de acuerdo; Escala 5: muy de acuerdo.	

Fuente: elaboración propia con base en Xu y Ramesh (2008).

La generación de guías de adaptación apropiadas permitirá seleccionar un proceso adecuado para un proyecto, y así la organización abandonará malas prácticas, como la selección de procesos anteriores o la utilización de procesos que les son familiares.

En la tabla 11.9, se presenta un listado de criterios para realizar la adaptación del proceso donde cada criterio es enunciado junto con su descripción. Los criterios están categorizados de acuerdo con el equipo, ambiente interno, ambiente externo y los objetivos.

Tabla 11.9. Criterios para adaptación del proceso

Criterio	Descripción
Equipo	
Tamaño	La necesidad de formalización aumenta si el equipo crece.
Distribución	Los equipos pequeños se comunican directamente; los equipos distribuidos necesitan comunicación formalizada.
Rotación de personal	Si un integrante abandona, se perderá conocimiento. Si nuevos integrantes entran en el equipo, se producirán efectos en la dinámica de grupo.
Cooperación previa	Si el equipo trabajó antes en proyectos, la comunicación es menos formal.
Buena cooperación	Si el equipo trabaja de manera buena y colaborativa, la necesidad de comunicación/documentación formal puede disminuir.
Conocimiento del dominio	Poco o falta de conocimiento con respecto al dominio real es un riesgo.
Conocimiento de herramientas	Poco o falta de conocimiento con respecto a las herramientas reales es un riesgo.
Conocimiento de tecnologías	Poco o falta de conocimiento con respecto a la tecnología es un riesgo.
Conocimiento del proceso	Poco o falta de conocimiento con respecto al proceso a utilizar es un riesgo.
Ambiente interno	
Prototipos	Estrategia para la mitigación de riesgos que se aplica a proyectos que exploran un nuevo dominio o tecnología o si los requisitos son volátiles.
Propuesta clara de proyecto	Contiene objetivos y requisitos básicos esenciales para el éxito del proyecto.
Disponibilidad de la gestión	Se requiere la alta gerencia para resolver problemas y tomar decisiones de progreso del proyecto.
Apoyo de la gestión	La alta dirección debe apoyar activamente un proyecto.
Presupuesto del proyecto	Un pequeño presupuesto del proyecto generalmente implica un proceso no formalizado, pero también requiere un estricto control de costos.
Duración del proyecto	Factor que influye directamente en el proceso. Una duración larga puede causar riesgos y una duración corta es similar a un pequeño presupuesto.
Tipo de proyecto	Se deben enfatizar diferentes facetas de un proceso de software (tipo de obtención de requisitos, fases del ciclo de vida).
Rol del proyecto	Rol específico que caracteriza a un proyecto en relación con otros proyectos. La configuración típica es un cliente y un contratista.
Subcontratistas	Pueden realizar tareas en un proyecto. El proceso define artefactos que se intercambiarán e interfaces para la colaboración.

Criterio	Descripción
Control financiero	Un control financiero intensivo produce una documentación independiente y mayor participación en la planificación y toma de decisiones.
Medición	Proporciona información sobre el estado y mide el rendimiento del proyecto.
Soporte técnico	Un entorno técnico desconocido puede causar riesgos en el proyecto.
Lenguaje de programación	Nuevos lenguajes de programación pueden causar riesgos en el proyecto.
Productos reutilizables	Componentes reutilizables acortan el tiempo de desarrollo, pero consideran implicaciones legales y procedimientos de prueba e integración.
Sistema operativo	Un sistema operativo limita los lenguajes de programación, herramientas y componentes disponibles.
Bases de datos	El soporte para bases de datos es esencial en sistemas de información.
Infraestructura de herramientas	Deben definirse las herramientas que se utilizarán en un proyecto de software o en una fase específica.
Ambiente externo	
Aspectos legales	Críticos en aspectos como tiempos de entrega, funcionalidad, seguridad y protección.
Número de <i>stakeholders</i>	Cuanto mayor sea el número de partes interesadas involucradas, más tiempo se requiere para negociar todas las necesidades y los requisitos.
Disponibilidad de <i>stakeholders</i>	La falta de disponibilidad de las partes interesadas es un riesgo del proyecto, ya que puede causar demoras.
Conocimiento de los <i>stakeholders</i>	Si no son adecuados, el proceso de software debe proporcionar diferentes estrategias para la obtención de requisitos.
Estabilidad de requisitos	Influye directamente en el enfoque de un proyecto.
Proceso del cliente	Si un cliente tiene su propio proceso de software, requiere alinear los contratistas a través de interfaces de proceso o procedimientos.
Disponibilidad del cliente	Influye en la satisfacción del cliente, ya que el cliente puede monitorear continuamente el progreso.
Tipo de contrato	Influye directamente en el proceso de software. Por ejemplo, un modelo de precio fijo versus tiempo conduce a diferentes estrategias.
Disponibilidad de usuario	La disponibilidad del usuario final es importante durante la ingeniería de requisitos y las fases de integración y prueba.
Conocimiento del usuario	Para decidir sobre la capacitación requerida. Las estrategias de prueba deben seleccionarse de acuerdo con el conocimiento de los usuarios.
Entrenamiento	Requisitos relacionados con la capacitación. Por ejemplo, durante las pruebas de aceptación o el despliegue causan esfuerzos adicionales.
Objetivos	
Complejidad	A mayor complejidad, el proceso tiene una configuración más formalizada.

Criterio	Descripción
Grado de innovación	Puede causar un enfoque más exploratorio para mitigar los riesgos del proyecto.
Sistemas heredados	Considerarlos durante la ingeniería de requisitos. Limita el espacio de la solución, por ejemplo, requisitos de compatibilidad y migración de datos.
Documentación de sistemas heredados	Si no hay documentación para un sistema heredado disponible, se debe esperar un mayor esfuerzo en el análisis del sistema heredado.
Dominio	Implica estándares, normas, regulaciones y leyes.
Solución conceptual	Un proceso de software puede contener conocimientos específicos de dominio y mejores prácticas.
Solución técnica	Con contenidos específicos con respecto a desarrollo web, patrones, etc.
Seguridad y protección	Causan una documentación completa de un proyecto. El proceso debe proporcionar plantillas y sugerencias sobre la documentación.
Desarrollo de <i>hardware</i>	Si se incluye, el proceso debe proporcionar los artefactos correspondientes que se integran al proceso de software.
Interfaces con otros sistemas	Un sistema software puede ser parte de un ecosistema integrado. Las interfaces con este ecosistema deben definirse.
Interfaz de usuario	Si se tienen requisitos especiales de interfaz de usuario, el diseño, la implementación y las pruebas deben ser parte del proceso de software.
Pruebas de integración y de sistemas	Los requisitos relacionados con la integración del sistema deben ser reflejados por el proceso de software.

Fuente: elaboración propia con base en Kalus y Kuhrmann (2013).

► 11.7. Gestión del conocimiento para el reúso del proceso de software

Una característica común de la mayoría de los procesos de software es que requieren una gran cantidad de conocimiento para aplicarse efectivamente. El desarrollo de software es un proceso intensivo de conocimiento en el que la productividad puede disminuir con la pérdida y búsqueda del conocimiento previo, la repetición de errores y el exceso de información (Verma y Tiwari, 2009). Muchos conceptos de la gestión del conocimiento pueden aplicarse para apoyar y reutilizar conocimiento en los procesos de software. El conocimiento se puede clasificar en dos categorías (Rus *et al.*, 2002):

- *Conocimiento tácito*: Se considera en forma de ideas individuales, es altamente informal, personal, no verbalizado, intuitivo, no articulado y derivado de la experiencia y creencias.
- *Conocimiento explícito*: Es formal y sistemático. Se almacena en forma de artículos, textos, publicaciones y bases de datos.

Sin embargo, transformar el conocimiento tácito en conocimiento explícito es bastante difícil. Por tanto, requiere una gestión del conocimiento (Knowledge Management [KM]) apropiada. La gestión del conocimiento surge como área de interés que puede aplicarse para reutilizar el conocimiento de proyectos previos a fin de aplicarlo en nuevos proyectos. La gestión del conocimiento puede definirse como la disciplina que estudia la creación, preservación, aplicación y el reúso del conocimiento que está disponible en una organización; su objetivo es crear conocimiento compartido entre todos los usuarios (Alavi y Leidner, 2001).

El trabajo pionero en la aplicación de la reutilización del conocimiento en procesos de software fue la factoría de experiencia (Basili *et al.*, 1994), una propuesta de organización lógica o física para analizar y sintetizar toda clase de experiencia que actúa como un repositorio, empaquetando la experiencia por medio de una construcción informal, formal o esquematizada de modelos y medidas de varios productos y procesos de software y otras formas de conocimiento, a partir de documentos, personas o soporte automatizado (figura 11.19).

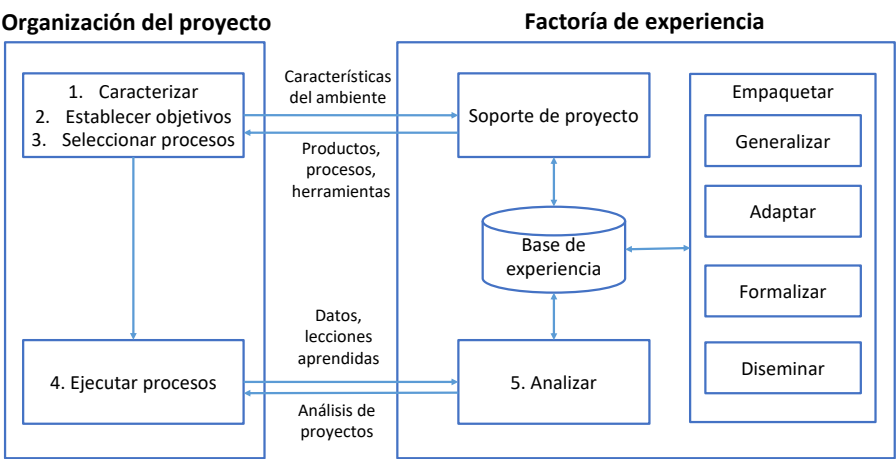


Figura 11.19. Factoría de experiencia

Fuente: elaboración propia con base en Basili *et al.* (1994).

Se pueden distinguir dos enfoques básicos para la gestión del conocimiento (Hansen *et al.*, 1999):

- ▶ *Codificación del conocimiento*: El conocimiento se documenta en forma de directrices y procedimientos. Para el caso del proceso de software, en repositorios y modelos de procesos de software. Sin embargo, algunos conocimientos son demasiado complejos como para documentarlos.
- ▶ *Personalización del conocimiento*: Los empleados acumulan la experiencia necesaria intercambiando experiencias entre ellos.

En los procesos de desarrollo de software, se pueden observar estos dos enfoques. Los modelos de procesos tradicionales (capítulo dos) se basan principalmente en la estrategia de codificación, definiendo los pasos relevantes de antemano, mientras el desarrollo ágil pone más énfasis en el conocimiento personalizado (Kneuper, 2018).

En la figura 11.20, se muestra una estructura común utilizada para gestión del conocimiento mediante los “componentes básicos de la gestión del conocimiento” (Probst, 1998).

- ▶ *Objetivos de conocimiento*: Para apoyar a los equipos de desarrollo en su tarea de producir resultados de alta calidad de manera eficiente.
- ▶ *Identificación del conocimiento*: Se refiere a la identificación del conocimiento necesario para los procesos de software, incluida una descripción de alto nivel del proceso y de nivel más detallada de los pasos del proceso, los artefactos producidos y los roles involucrados.
- ▶ *Adquisición de conocimiento*: Un enfoque para obtener el conocimiento necesario según lo identificado. Por ejemplo, utilizando un modelo de referencia estándar como **RUP** o *scrum*. Puede ser adquirido interna o externamente.
- ▶ *Desarrollo del conocimiento*: Si el conocimiento necesario aún no existe, debe desarrollarse.
- ▶ *Distribución del conocimiento*: El conocimiento desarrollado debe distribuirse para que sea útil mediante las estrategias de codificación o personalización.
- ▶ *Preservación del conocimiento*: El proceso se documenta y se realizan actividades, como capacitación continua, revisión periódica de los procesos y aseguramiento de la calidad para que el conocimiento no se pierda o quede desactualizado.

- *Uso del conocimiento*: Es el núcleo de todas las actividades porque solo al utilizarlo el conocimiento se vuelve relevante.
- *Medición del conocimiento*: Se mide el conocimiento del proceso y los resultados de las mediciones ayudan a adaptar los objetivos de este.

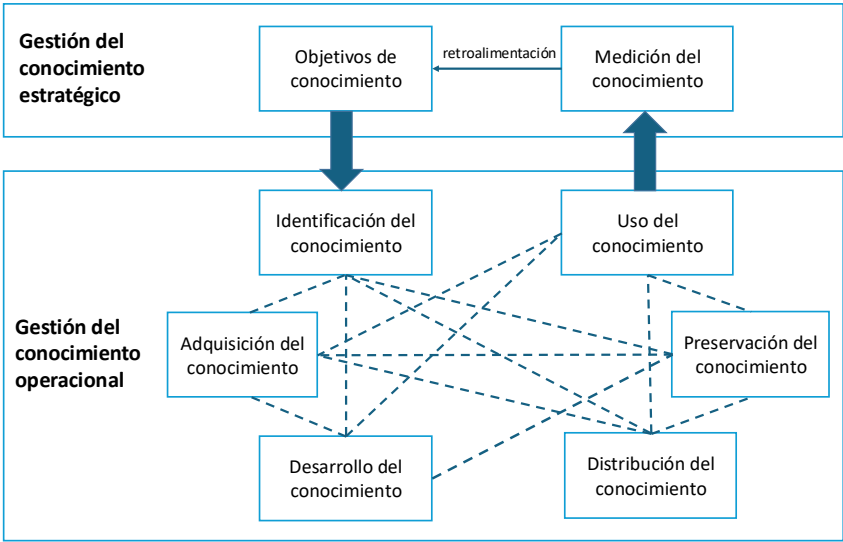


Figura 11.20. Principales procesos de gestión del conocimiento

Fuente: elaboración propia con base en Probst (1998).

Las técnicas de gestión del conocimiento pueden ayudar a crear y compartir conocimiento a nivel organizacional, departamental, proyectos e individuos, y adaptarlo a proyectos específicos. Cuando se desarrolla un proyecto de software, se debe contar con conocimiento apropiado para realizar con éxito cada tarea y a su vez ese conocimiento debe estar disponible cada vez que el proceso se ejecute, y así facilitar el trabajo de los equipos de desarrollo, como se observa en la figura 11.21.

La gestión del conocimiento permite registrar experiencias adquiridas durante el desarrollo de proyectos, de tal manera que se necesite menos esfuerzo y tiempo en la definición de procesos para proyectos futuros (Kneuper, 2002).

Algunos beneficios de institucionalizar programas de gestión del conocimiento para el proceso de software son (Rus *et al.*, 2001):

- Reúso eficiente de experiencias documentadas.
- Facilidad para solucionar problemas dentro de la organización.
- Identificación y almacenamiento de experiencias valiosas.
- Facilitar la definición de medidas para mejorar la ejecución del proceso e incrementar la calidad de los productos de software.

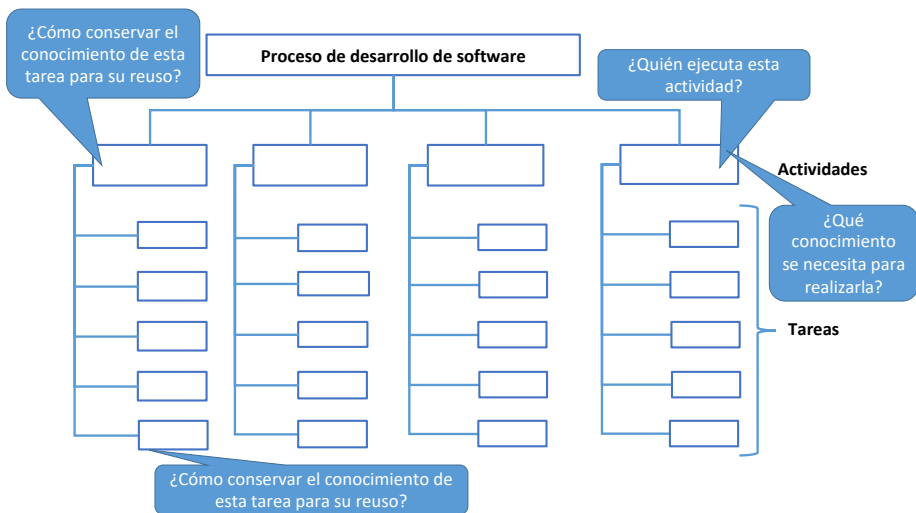


Figura 11.21. Proceso de software basado en conocimiento

Fuente: elaboración propia con base en Bermón Angarita (2010).

► Desafíos

El trabajo pionero de Osterweil (2011) estableció que los procesos de software también son software y, al igual que este, deben tener sus requisitos especificados, pueden modelarse, desarrollarse, probarse y reutilizarse. Para la ingeniería de procesos de software, las partes reutilizadas pueden ser operaciones, actividades, pasos, una solución o, incluso, un conjunto de posibles soluciones (Hassan, 2018).

Algunas cuestiones determinantes relacionadas con la adaptación del proceso de software que deben considerar las organizaciones desarrolladoras de software son el logro de una adaptación a nivel organizacional (adaptar un proceso estándar a una organización determinada) o en pro-

yecto (adaptar el proceso organizacional a un proyecto específico), con un enfoque formal (con pasos rigurosos y bien definidos) o informal (con guías y recomendaciones), la conformidad con estándares (por ejemplo, es una actividad obligatoria en modelos como **CMMI**) y la incorporación de herramientas de soporte (debido a la naturaleza del proceso de software como una actividad intensiva en conocimiento) (Pedreira *et al.*, 2007).

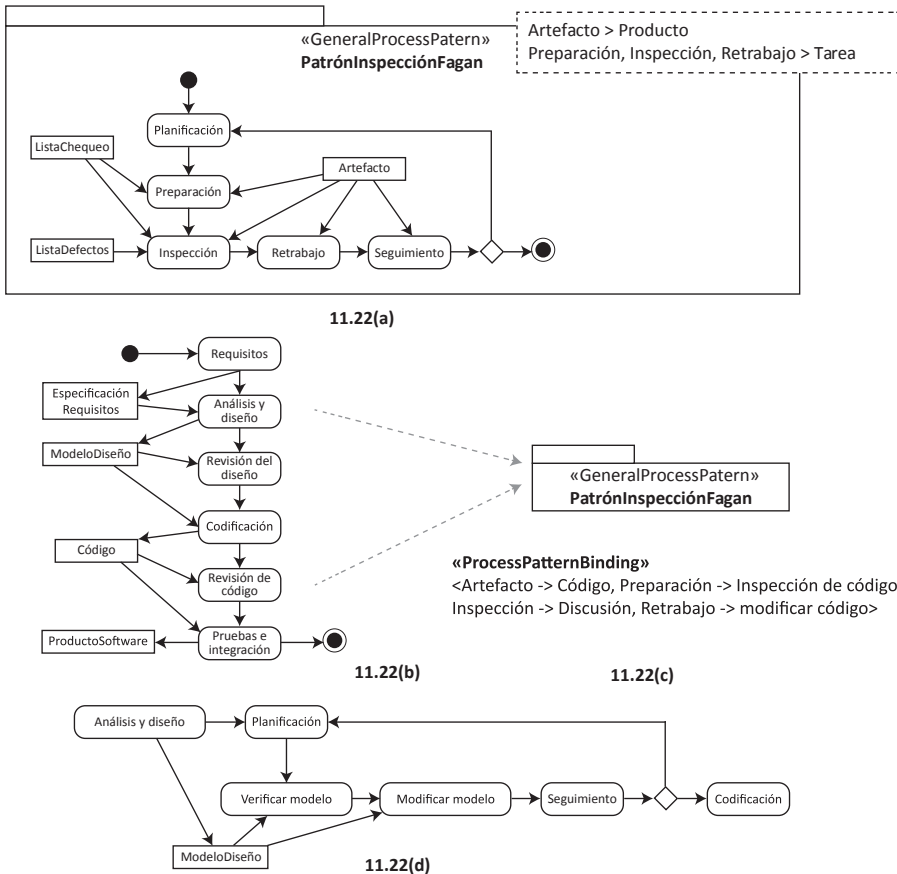
Principalmente, los errores al alinear el proceso de software estándar con el contexto de un proyecto puede tener consecuencias en el tiempo de desarrollo, la calidad y el costo del proyecto. Este desafío es fundamental cuando las empresas desarrollan proyectos muy diferentes entre sí. Las guías de adaptación del proceso (*tailoring*) son una buena opción para este tipo de organizaciones. Para pymes, la adaptación del proceso de software puede ser oneroso. Además, en este tipo de empresas, el proceso de software sigue un enfoque *ad hoc*, lo que hace que el proceso resultante sea dependiente de la capacidad del personal. La adaptación del proceso se recomienda si la empresa tiene varias líneas de producto lo suficientemente detalladas y definidas (Fitzgerald *et al.*, 2003; Nanda, 2001).

Otras dificultades esperadas de la adopción de un enfoque de reutilización de procesos son dificultad para identificar, recuperar y modificar elementos reutilizables, calidad insuficiente de los elementos reutilizables, existencia de barreras psicológicas, legales o económicas, necesidad de crear incentivos para la reutilización, alto costo de implementación y herramientas de soporte inadecuadas (Barreto *et al.*, 2011).

Con respecto a la aplicación de patrones de proceso, las principales dificultades están muy alineadas con las relacionadas con el modelado de procesos: cómo definir los detalles de un elemento de proceso (por ejemplo, tareas, productos de trabajo y roles) y cómo organizar un grupo de elementos de proceso para satisfacer una situación especial o ciertas restricciones (Tran *et al.*, 2007).

► Ejemplos

Se presentarán en esta sección algunos ejemplos para ilustrar cómo las aplicaciones de patrones de proceso se representan en los modelos de proceso. La figura 11.22 muestra un ejemplo del uso de patrones de proceso para generar contenidos de elementos de proceso.



Nota: 11.22(a): un patrón de proceso captura una definición reusable para una actividad de revisión. 11.22(b): modelo de proceso línea base con elementos de proceso no especificados. 11.22 (c): enlaces entre procesos que genera elementos fuente. 11.22(d): modelo resultante después de aplicar patrón de procesos.

Figura 11.22. Ejemplo de aplicación de patrón de proceso para generar elemento de proceso

Fuente: elaboración propia con base en Tran *et al.* (2011).

La figura 11.22(b) muestra el modelo del proceso de desarrollo de línea de base que contiene elementos de proceso no especificados. Al refinar este proceso de referencia, el diseñador de procesos decidió aplicar el proceso de Fagan para revisar diferentes artefactos de software. Así que reutiliza el patrón de inspección de Fagan (figura 11.22(a)) para generar el

contenido de las tareas de revisión del diseño y revisión de código. La figura 11.22(c) muestra el uso de patrón de inspección de Fagan para generar los contenidos de las tareas de revisión de diseño y revisión de código. Estas aplicaciones se representan a través de las relaciones con los parámetros reales que sustituyen los parámetros formales del patrón correspondiente. El modelo resultante se muestra en la figura 11.22(d).

► Resumen

El reúso de proceso de software asimila conceptos tomados del reúso de componentes software para adaptarlos al dominio de los procesos de software. Los conceptos de activos de software (que se convertirán en activos de proceso de software), repositorios de reúso (que se vuelven repositorios de conocimiento, o Process Asset Library [PAL]) y proceso de reúso serán adaptados para su uso efectivo en procesos de software.

Los elementos del reúso del proceso de software pueden ser entendidos por medio del modelo 3C: Concepto, Contenido y Contexto.

El estándar **IEEE 1517: 2010** Tecnología de la información - Proceso del ciclo de vida de software y sistemas - Proceso de reúso es una extensión de la norma **ISO/IEC 12207** que se enfoca en el reúso dentro del ciclo de vida del software. El estándar consta de cuatro categorías: desarrollo, operación y mantenimiento de productos de software utilizando activos reutilizables; desarrollo y mantenimiento de activos reutilizables; gestión del reúso, y gestión de activos reutilizables.

Un componente de software es una unidad de software independiente y portable que está completamente definida y al que se accede a través de un conjunto de interfaces. Los componentes de procesos tienen una interfaz requerida, una interfaz proporcionada, precondiciones, poscondiciones y una colección de atributos invariantes y variables.

Una línea de procesos de software es un conjunto de procesos similares dentro de un dominio particular o para un propósito particular, que tienen características comunes y variabilidad controlada. Se caracterizan por contener elementos de proceso que son similares y variables.

Una arquitectura de procesos de software es un marco conceptual para incorporar, relacionar y adaptar de manera consistente los elementos del proceso en procesos ejecutables. Una arquitectura está conformada por

una arquitectura de negocios, una arquitectura de aplicación y una arquitectura de infraestructura.

Un patrón de proceso captura un modelo de proceso que representa una solución que se puede aplicar en un contexto dado para resolver un problema recurrente de modelado de procesos. Existen tres tipos de patrones de proceso: abstractos, generales y concretos.

Se pueden aplicar métricas para evaluar los programas de reúso del proceso que tiene una organización. Las entidades que se pueden evaluar se pueden agrupar en productos, procesos y recursos orientados al reúso.

La adaptación del proceso es una actividad fundamental en la que los procesos estándares de la organización son adecuados a las características particulares de cada proyecto de desarrollo de software.

Por último, la gestión del conocimiento contiene principios, conceptos, recursos y tecnologías que ayudan al reúso del conocimiento del proceso. Los procesos de gestión del conocimiento: identificación, adquisición, desarrollo, distribución, preservación, uso y medición, ayudan a crear y compartir conocimiento a nivel organizacional, departamental, proyectos e individuos, y adaptarlo a proyectos específicos.

► Ejercicios

1. ¿Cuál es la diferencia de software con reutilización y software para reutilización?
2. Desarrollar un patrón de proceso de software para las siguientes actividades que incluya: nombre, autor, intención, problema, contexto, solución, consecuencias y usos conocidos:
 - Un evento del proceso *scrum*.
 - Una fase del proceso unificado.
3. ¿Cuál es la diferencia entre el proceso estándar de una organización y el proceso de software definido para un proyecto?
4. Describir cómo se puede adaptar un proceso estándar para producir un proceso definido para un proyecto.
5. ¿Qué es un componente de proceso de software?
6. ¿En qué se diferencian las líneas de proceso de las arquitecturas de proceso?

7. Seleccionar un proceso de software que esté utilizando actualmente en sus proyectos y aplicar los criterios de adaptación del proceso mostrados en la tabla 11.9.
8. ¿Qué es una factoría de experiencia y cuál es su relación con la gestión del conocimiento del proceso de software?

► Referencias

- Akbar, R., Hassan, M. F. y Abdullah, A. (2011). A review of prominent work on agile processes Software Process Improvement and process tailoring practices. En J. M. Zain, W. M. b. Wan Mohd y E. El-Qawasmeh (eds.), *Software Engineering and Computer Systems. ICSECS 2011. Communications in Computer and Information Science* (pp. 571-585). Springer. https://doi.org/10.1007/978-3-642-22203-0_49
- Alavi, M. y Leidner, D. E. (2001). Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly*, 25(1), 107-136. <https://doi.org/10.2307/3250961>
- Alexander, C. (1979). *The timeless way of building*. Oxford University Press.
- Anguswamy, R. y Frakes, W. B. (2012). A study of reusability, complexity, and reuse design principles. En *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 161-164). <https://doi.org/10.1145/2372251.2372280>
- Armbrust, O., Katahira, M., Miyamoto, Y., Münch, J., Nakao, H. y Ocampo, A. (2009). Scoping software process lines. *Software Process: Improvement and Practice*, 14(3), 181-197. <https://doi.org/10.1002/spip.412>
- Barreto, A. S., Murta, L. G. P. y Rocha, A. R. (2011). Software process definition: A reuse-based approach. *Journal of Universal Computer Science (JUCS)*, 17(13), 1765-1799. https://www.jucs.org/jucs_17_13/software_process_definition_a/jucs_17_13_1765_1799_barreto.pdf
- Basili, V., Caldiera, G. y Rombach, D. (1994). Experience factory. En *Encyclopedia of software engineering* (vol. 1, pp. 476-496). John Wiley & Sons.
- Bermón Angarita, L. (2010). *Librería de activos para la gestión del conocimiento sobre procesos de software: PAL-Wiki* [tesis doctoral, Universidad Carlos III de Madrid]. https://e-archivo.uc3m.es/bitstream/handle/10016/10231/Tesis_Leonardo_Bermon.pdf?sequence=2&isAllowed=y
- Bhuta, J., Boehm, B. y Meyers, S. (2006). Process elements: Components of software process architectures. En M. Li, B. Boehm y L. J. Osterweil (eds.), *Unifying the Software Process Spectrum. SPW 2005. Lecture Notes in Computer Science* (pp. 332-346). Springer. https://doi.org/10.1007/11608035_28
- Birk, A., Heller, G., John, I., Schmid, K., von der Maßen, T. y Muller, K. (2003). Product line engineering, the state of the practice. *IEEE Software*, 20(6), 52-60. <https://doi.org/10.1109/MS.2003.1241367>

- Chrissis, M. B., Konrad, M. y Shrum, S. (2006). *CMMI: Guidelines for process integration and product improvement*. Addison-Wesley.
- Ezran, M., Morisio, M. y Tully, C. (2002). *Practical software reuse*. Springer.
- Feiler, P. H. y Humphrey, W. S. (1993). Software process development and enactment: Concepts and definitions. En *Proceedings of the Second International Conference on the Software Process-Continuous Software Process Improvement* (pp. 28-40). **IEEE**. <https://doi.org/10.1109/SPCON.1993.236824>
- Fenton, N. y Bieman, J. (2014). *Software metrics: A rigorous and practical approach*. CRC Press.
- Fitzgerald, B., Russo, N. y O'Kane, T. (2003). Software development method tailoring at Motorola. *Communications of the ACM*, 46(4), 65-70. <https://doi.org/10.1145/641205.641206>
- Forrester, E. (ed.) (2006). *A process research framework: The international process research consortium*. Carnegie Mellon University, Software Engineering Institute.
- Fusaro, P., Tortorella, M. y Visaggio, G. (1998). **REP**-characterising and exploiting process components: Results of experimentation. En *Proceedings Fifth Working Conference on Reverse Engineering* (Cat. No. 98TB100261) (pp. 20-29). **IEEE**. <https://doi.org/10.1109/WCRE.1998.723172>
- Gallina, B., Kashiyyarandi, S., Martin, H. y Bramberger, R. (2014). Modeling a safety-and automotive-oriented process line to enable reuse and flexible process derivation. En *2014 IEEE 38th International Computer Software and Applications Conference Workshops* (pp. 504-509). **IEEE**. <https://doi.org/10.1109/COMPSACW.2014.84>
- Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison Wesley.
- Gary, K. A. y Lindquist, T. E. (1999). Cooperating process components. En *Proceedings. Twenty-Third Annual International Computer Software and Applications Conference* (Cat. No. 99CB37032) (pp. 218-223). **IEEE**. <https://doi.org/10.1109/CMP-SAC.1999.812704>
- Ginsberg, M. P. y Quinn, L. H. (1995). *Process tailoring and the software capability maturity model*. Carnegie Mellon University, Software Engineering Institute. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fe11de0ed0212b58fb9d-47152c94a34ab5b31974>
- Gnatz, M., Marschall, F., Popp, G., Rausch, A. y Schwerin, W. (2003). The living software development process. *Software Quality Professional*, 5(3), 4-16. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=72db6fd8ac2cc6ab2bacf7b03b-4c139a498f8c89>
- Hansen, M. T., Nohria, N. y Tierney, T. (2000). What's your strategy for managing knowledge? En J. A. Woods y J. Cortada (eds.), *The knowledge management yearbook 2000-2001* (pp. 55-69). Routledge. <https://doi.org/10.4324/9780080941042>
- Hassan, A. (2018). *Style and meta-style: Another way to reuse software architecture evolution* [tesis de doctorado, Université de Nantes]. <https://hal.science/tel-01917775/>
- Hollenbach, C. y Frakes, W. (1996). Software process reuse in an industrial setting. En *Proceedings of Fourth IEEE International Conference on Software Reuse* (pp. 22-30). **IEEE**. <https://doi.org/10.1109/ICSR.1996.496110>

- Hurtado Alegria, J. A., Bastarrica, M. C., Quispe, A. y Ochoa, S. F. (2014). MDE-based process tailoring strategy. *Journal of Software: Evolution and Process*, 26(4), 386-403. <https://doi.org/10.1002/smr.1576>
- Institute of Electrical and Electronics Engineers. (2010). *Std 1517-2010 IEEE Standard for Information Technology - Software Life Cycle Processes - Reuse Processes - Description*. IHS Standards.
- Kalus, G. y Kuhrmann, M. (2013). Criteria for software process tailoring: A systematic review. En *Proceedings of the 2013 International Conference on Software and System Process* (pp. 171-180). IEEE. <https://doi.org/10.1145/2486046.2486078>
- Karlsson, E. A. (ed.) (1995). *Software reuse: A holistic approach*. John Wiley & Sons.
- Kneuper, R. (2018). *Software processes and life cycle models: An introduction to modelling, using and managing agile, plan-driven and hybrid processes*. Springer. <https://doi.org/10.1007/978-3-319-98845-0>
- Kucza, T., Nättinen, M. y Parviainen, P. (2001). Improving knowledge management in software reuse process. En F. Bomarius y S. Komi-Sirviö (eds.), *Product Focused Software Process Improvement. PROFES 2001. Lecture Notes in Computer Science* (pp. 141-152). Springer. https://doi.org/10.1007/3-540-44813-6_15
- Latour, L., Wheeler, T. y Frakes, B. (1991). Descriptive and predictive aspects of the 3Cs model: SETA1 working group summary. En *Proceedings of the First International Symposium on Environments and Tools for Ada* (pp. 9-17). Syracuse University. <https://dl.acm.org/doi/pdf/10.1145/112629.112632>
- Li, T. (2008). Overview of software processes and software evolution. En *An approach to modelling software evolution processes* (pp. 8-33). Springer. https://doi.org/10.1007/978-3-540-79464-6_2
- Lim, W. C. (1998). *Managing software reuse: A comprehensive guide to strategically reengineering the organization for reusable components*. Prentice-Hall.
- Magdaleno, A. M., de Oliveira Barros, M., Werner, C. M. L., de Araujo, R. M. y Batista, C. F. A. (2015). Collaboration optimization in software process composition. *Journal of Systems and Software*, 103, 452-466. <https://doi.org/10.1016/j.jss.2014.11.036>
- McIlroy, M. D., Buxton, J., Naur, P. y Randell, B. (1968). Mass-produced software components. En *Proceedings of the 1st International Conference on Software Engineering, Garmisch Partenkirchen, Germany* (pp. 88-98). Petrocelli/Charter Publishers. <https://st.inf.tu-dresden.de/files/teaching/ss16/cbse/slides/50-cbse-transconsistent-composition.pdf>
- Medina Domínguez, F. (2010). *Marco metodológico para la mejora de la eficiencia de uso de los procesos de software* [tesis doctoral, Universidad Carlos III de Madrid]. <https://e-archivo.uc3m.es/bitstream/handle/10016/7433/Memoria%20Tesis-Fuensanta%20Medina%20Dominguez.pdf?sequence=1&isAllowed=y>
- Nanda, V. (2001). On tailoring an organizational standard software development process for specific projects. En *Proceedings of the 11th International Conference on Software Quality* (pp. 1-13). IEEE.
- O'Regan, G. (2017). *Concise guide to software engineering: From fundamentals to application methods*. Springer. <https://doi.org/10.1007/978-3-319-57750-0>

- Osterweil, L. (2011). *Software processes are software too*. En P. Tarr y A. Wolf (eds.), *Engineering of software* (pp 323-344). Springer. https://doi.org/10.1007/978-3-642-19823-6_17
- Pedreira, O., Piattini, M., Luaces, M. R. y Brisaboa, N. R. (2007). Una revisión sistemática de la adaptación del proceso de software. *REICIS: Revista Española de Innovación, Calidad e Ingeniería del Software*, 3(2), 21-39. <https://www.redalyc.org/pdf/922/92230204.pdf>
- Pesantes, M., Lemus, C., Mitre, H. A. y Mejía, J. (2012). Software process architecture: Roadmap. En 2012 *IEEE Ninth Electronics, Robotics and Automotive Mechanics Conference* (pp. 111-116). *IEEE*. <https://doi.org/10.1109/CERMA.2012.25>
- Pressman, R. S. (2005). *Software engineering: A practitioner's approach*. McGraw-Hill.
- Probst, G. J. B. (1998). Practical knowledge management: A model that works. En *Managing knowledge: Building blocks for success* (pp. 17-29). Wiley.
- Rombach, D. (2006). Integrated software process and product lines. En M. Li, B. Boehm y L. J. Osterweil (eds.) *Unifying the Software Process Spectrum. SPW 2005. Lecture Notes in Computer Science* (pp 83-90). Springer. https://doi.org/10.1007/11608035_9
- Rus, I., Lindvall, M. y Sinha, S. (2002). Knowledge management in software engineering. *IEEE Software*, 19(3), 26-38.
- Santos, V., Cortés, M. y Brasil, M. (2009). Dynamic management of the organizational knowledge using case-based reasoning. En L. A. Maciaszek, C. González-Pérez y S. Jablonski (eds.), *Evaluation of Novel Approaches to Software Engineering. ENASE ENASE 2009 2008. Communications in Computer and Information Science* (pp. 220-233). Springer. https://doi.org/10.1007/978-3-642-14819-4_16.
- Software Engineering Institute. (2010). *CMMI® para Desarrollo, Versión 1.3*. Software Engineering Institute. Carnegie-Mellon University, Pittsburg, Pennsylvania.
- Sommerville, I. (2015). *Software engineering: Always learning*. Pearson.
- Teixeira, E. N., Aleixo, F. A., de Sousa Amancio, F. D., Oliveira, E., Kulesza, U. y Werner, C. (2019). Software process line as an approach to support software process reuse: A systematic literature review. *Information and Software Technology*, 116, 106175. <https://doi.org/10.1016/j.infsof.2019.08.007>
- Tran, H. N., Coulette B. y Dong, T. B. T. (2005). A classification of process patterns. En *Proceedings of the International Conference on Software Development (SWDC-REK 2005)*, Reykjavik.
- Tran, H. N., Coulette, B. y Thuy, D. T. B. (2007). Broadening the use of process patterns for modeling processes. En *SEKE* (pp. 57-62). <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a094027803f6fc090c35caef958b33924789c960#page=77>
- Tran, H. N., Coulette, B., Tran, D. T. y Vu, M. H. (2011). Automatic reuse of process patterns in process modeling. En *Proceedings of the 2011 ACM Symposium on Applied Computing* (pp. 1431-1438). <https://doi.org/10.1145/1982185.1982494>
- Verma, A. y Tiwari, M. K. (2009). Role of corporate memory in the global supply chain environment. *International Journal of Production Research*, 47(19), 5311-5342. <https://doi.org/10.1080/00207540801918570>

- Washizaki, H. (2006). Deriving project-specific processes from process line architecture with commonality and variability. En *2006 4th IEEE International Conference on Industrial Informatics* (pp. 1301-1306). IEEE. <https://doi.org/10.1109/INDIN.2006.275847>
- Xu, P. y Ramesh, B. (2008). Using process tailoring to manage software development challenges. *IT Professional*, 10(4), 39-45. <https://doi.org/10.1109/MITP.2008.81>



► Parte IV. Aplicaciones

Esta última parte del libro tiene como objetivo mostrar que el área del conocimiento de la ingeniería de procesos de software, como todo campo científico y tecnológico, tiene implicaciones prácticas en desarrollos específicos y presenta muchos casos de éxito en los que la creación o modificación de procesos de software han contribuido al desarrollo organizacional, la innovación y la mejora continua de las organizaciones. Además, el área está en continuo crecimiento, planteándose nuevas preguntas por resolver, definiendo problemas concretos y considerando el devenir del desarrollo informático para trazar tendencias actuales y futuras de investigación en el área.

► Capítulo 12. Procesos de software para aplicaciones específicas

Este capítulo presenta procesos de software que han sido creados y propuestos, modificados o adaptados a las necesidades particulares de las aplicaciones a desarrollar. Cada aplicación está enmarcada en un área de conocimiento específico en la que un proceso genérico o tradicional debe adaptarse cumpliendo criterios o estándares propios del ámbito de la aplicación. El capítulo describe procesos propuestos para diversas aplicaciones informáticas actuales, como el reconocimiento de patrones, minería de datos, *machine learning*, internet de las cosas, entre otras.

► Capítulo 13. Casos de éxito

Este capítulo describe varios casos de éxito en empresas desarrolladoras de software reconocidas a nivel mundial. Cada caso de estudio presentado contiene un panorama

general de la situación de la empresa en un momento dado y cómo algún tipo de implementación relacionada con procesos de software permitió que la empresa evolucionara para resolver problemas concretos, cumpliera sus objetivos de negocio o innovara en la forma de desarrollar sus productos de software.

► **Capítulo 14. Tendencias actuales de investigación del proceso de software**

Este capítulo se ocupa de presentar las principales tendencias actuales de investigación en el área de la ingeniería de procesos de software. Para ello, se describirá el marco de trabajo International Process Research Consortium (**IPRC**) que sirve como punto de partida para formular iniciativas de investigación en procesos. El capítulo también presenta rutas de investigación basadas en la minería de procesos y en la incorporación de técnicas basadas en el aprendizaje automático y el aprendizaje profundo aplicadas a los procesos de software. El capítulo finaliza con una descripción de nuevos modelos de procesos surgidos debido a la globalización económica, la consolidación de internet con las aplicaciones móviles, la computación en la nube y los dispositivos electrónicos conectados a internet, la fusión de plataformas tecnológicas y los nuevos esquemas de desarrollo de software.

Procesos de software para aplicaciones específicas

► Objetivos

Los objetivos de este capítulo consisten en describir algunos procesos de software desarrollados para aplicaciones informáticas concretas. Al estudiar este capítulo, el lector:

- Obtendrá un panorama general de algunos procesos de software que han sido definidos para el desarrollo de aplicaciones informáticas actuales y ampliamente utilizadas en la industria.
- Comprenderá que los procesos tradicionales, ágiles o industriales han sido modificados o adaptados para cubrir las necesidades de proyectos de software en áreas de conocimiento particulares.

► 12.1. Desarrollo de aplicaciones de reconocimiento de patrones

El reconocimiento, la descripción, la clasificación y la agrupación automáticos de patrones son problemas importantes en una variedad de disciplinas científicas y de ingeniería, como la biología, la psicología, la medicina, el *marketing*, la visión por computadora, la inteligencia artificial (**IA**) y la detección remota.

Un patrón se puede definir como “lo opuesto al caos; es una entidad, vagamente definida, a la que se le podría dar un nombre” (Watanabe, 1985). Por ejemplo, un patrón podría ser la imagen de una huella dactilar, una palabra manuscrita, un rostro humano o una señal de voz. El reconocimiento/ clasificación de un patrón puede consistir en alguna de las siguientes tareas (Jain *et al.*, 2000):

- *Clasificación supervisada*: En la que el patrón de entrada se identifica como miembro de una clase predefinida.
- *Clasificación no supervisada*: En la que el patrón se asigna a una clase desconocida.

El interés en el reconocimiento de patrones se ha incrementado en las últimas décadas debido al surgimiento de aplicaciones emergentes que requieren un exigente procesamiento computacional (tabla 12.1).

Tabla 12.1. Ejemplos de aplicaciones de reconocimientos de patrones

Dominio del problema	Aplicación	Patrón de entrada	Clases patrón
Bioinformática	Análisis de secuencia	ADN/secuencia de proteínas	Tipos de genes/ patrones
Minería de datos	Búsqueda de patrones significativos	Puntos en el espacio multidimensional	Clústeres compactos y bien definidos
Clasificación de documentos	Búsqueda en internet	Documentos de texto	Categorías semánticas
Análisis de imágenes de documentos	Lectura para ciegos	Imagen de documentos	Caracteres alfanuméricos
Automatización industrial	Impresión de circuitos impresos	Intensidad o rango de imágenes	Defectos en productos
Recuperación en bases de datos multimedia	Búsqueda en internet	Clips de video	Géneros de video
Reconocimiento biométrico	Identificación personal	Rostros, iris, huellas digitales	Usuarios autorizados para control de acceso
Sensores remotos	Pronóstico de rendimiento de cultivos	Imagen de multiespectro	Categorías de uso de la tierra
Reconocimiento de voz	Solicitud telefónica sin operador	Forma de la onda de habla	Palabras habladas

Fuente: elaboración propia con base en Jain *et al.* (2000).

Un sistema de reconocimiento de patrones es un sistema que adquiere datos, extrae un conjunto de características destacadas de estos, compara este conjunto de características con los conjuntos de características almacenados en una base de datos y ejecuta una acción basada en el resultado de la comparación. El desarrollo de un sistema de reconocimiento de patrones típico se puede observar en la figura 12.1 con cinco fases principales: adquisición, preprocesamiento, extracción de características, clasificación y una de base de datos (Jain *et al.*, 2008).

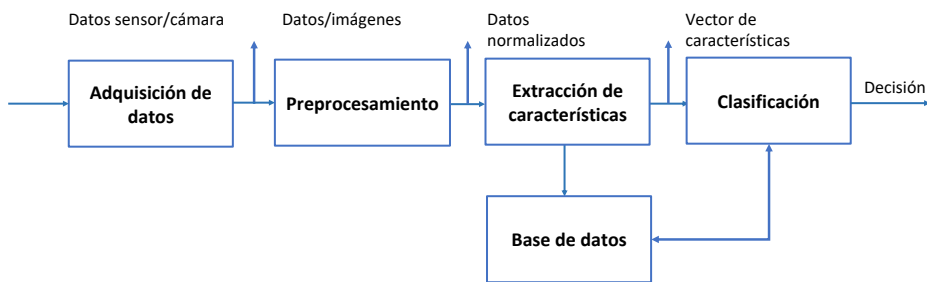


Figura 12.1. Fases típicas de desarrollo de un sistema de reconocimiento de patrones

Fuente: elaboración propia con base en Jain *et al.* (2008).

- **Adquisición:** Se requiere un lector, cámara o escáner adecuado para adquirir los datos sin procesar.
- **Preprocesamiento:** la calidad de los datos adquiridos puede no ser de buena calidad, por lo que se evalúa para determinar su idoneidad para un procesamiento posterior. Además, los datos adquiridos se someten a una mejora de la señal para incrementar su calidad. Algunos pasos de preprocesamiento que se pueden aplicar antes de la extracción de características son normalización del tamaño, normalización de iluminación (para reconocimiento de imágenes), filtrado (eliminación de ruido), filtrado de paso alto (elimina los gradientes de baja frecuencia en la imagen), eliminación de fondo, normalización traslacional y rotacional (cuando la imagen está desplazada o girada).
- **Extracción de características:** Se procesan los datos y se extrae un conjunto de características destacadas para representar la característica subyacente. Por ejemplo, la posición y las distancias entre los ojos, la nariz y la boca en una imagen facial. Durante el entre-

namiento, este conjunto de funciones se almacena en la base de datos y se denomina plantilla.

- *Clasificación:* Las características extraídas se comparan con las plantillas almacenadas para generar puntajes de coincidencia. Se determina la distancia mínima entre la entrada y los conjuntos de características de la plantilla y se informa una puntuación de coincidencia. También contiene un módulo de toma de decisiones, en el que las distancias se utilizan para aceptar o negar un reconocimiento determinado.
- *Base de datos:* Actúa como repositorio de información biométrica. Durante la etapa de entrenamiento, el conjunto de características extraído de la muestra sin procesar se almacena en la base de datos y el módulo de clasificación lo utiliza más tarde para identificar conjuntos de características de entrada.

► 12.2. Desarrollo de aplicaciones de internet de las cosas

Business Process Model and Notation (**BPMN**) es uno de los lenguajes estándares para el modelado de procesos de negocio, como se presentó en el apartado 5.3.2. A pesar de su amplio uso, cuando se trata de representar procesos comerciales ubicuos, este lenguaje de modelado de procesos comerciales no es adecuado (Abouzid y Saidi, 2019).

A continuación, se presentan algunos ejemplos de ubicuidad que BPMN no puede afrontar apropiadamente (Yousfi *et al.*, 2016):

- El peaje de una carretera se puede pagar sobre la marcha utilizando una etiqueta Radio Frequency Identification (**RFID**) en el parabrisas del automóvil. La identificación automática no se puede representar de manera adecuada mediante **BPMN**.
- Un taxi se asigna a un cliente en función de su ubicación actual. El mecanismo de identificar la ubicación actual del usuario y asignarle el taxi más apropiado no puede ser representado.

Para corregir estas falencias, se desarrolló uBMPN o **BPMN** ubicuo. uBPMN introduce tres nuevos elementos de tarea: tarea sensor, tarea lector, tarea recolector, tarea imagen y tarea audio (Yousfi *et al.*, 2019):

- *Sensor:* Incluye sensores inteligentes, alámbricos e inalámbricos e información de contexto del sensor en un ambiente de negocios.

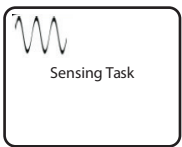
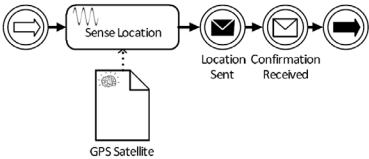
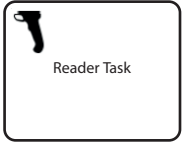
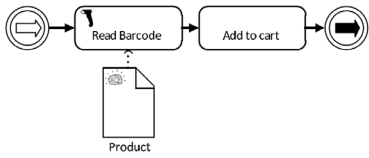
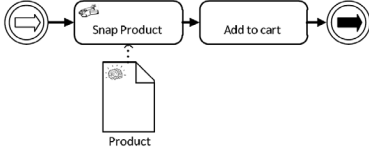
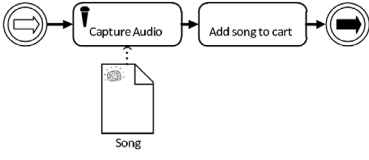
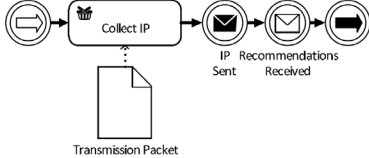
Los tipos de sensores pueden categorizarse en temperatura, velocidad, movimiento, Global Positioning System (**GPS**), etc.

- ▶ *Lector*: Utiliza un lector inteligente como barras de código, **RFID**, datos biométricos, etc.
- ▶ *Colector*: Recolecta cualquier contexto que sensores y lectores no son capaces de recolectar. La fuente de información de contexto es un archivo, una base de datos u otro proceso.
- ▶ *Imagen*: Tarea que utiliza una transmisión de video para recopilar una información particular en el entorno de negocios. Los datos recopilados son una captura de imagen.
- ▶ *Audio*: Tarea que utiliza un flujo de audio para recopilar información particular en el entorno de negocios. Los datos recopilados son un segmento de audio.

En la tabla 12.2, se presentan las notaciones gráficas de los elementos tarea de uBPMN y algunos ejemplos de aplicación.

Un ejemplo de un modelo uBPMN se presenta en la figura 12.2, cuyo objetivo de negocios es “Solicitar monturas para anteojos”. El proceso despliega tres tecnologías de computación ubicua de entrada (imagen, colector y sensor) y describe tres características de computación ubicua (sostenibilidad, realidad aumentada y conciencia del contexto). En resumen, uBPMN satisface los requisitos de la computación ubicua y sirve como un impulsor hacia una extensión oficial de **BPMN** v2.0 para modelar procesos de negocio ubicuos.

Tabla 12.2. Notaciones gráficas de elementos tarea de uBPMN

Elemento tarea	Ejemplo	Descripción
 Sensing Task		Un usuario solicita un taxi que se le asigna después que un sensor detecta su ubicación actual a partir de un satélite GPS. Luego, la ubicación se transfiere a la compañía de taxis.
 Reader Task		Un comprador se encuentra en una tienda virtual y puede agregar artículos a su carrito escaneando sus códigos de barras.
 Image Task		Un usuario que ve un letrero publicitario en la calle no tiene acceso a su código de barras. Pero puede usar un algoritmo de reconocimiento para agregar el producto al carrito de compras.
 Audio Task		Un usuario está escuchando una canción y quiere comprarla, pero no conoce al cantante ni el título. Puede agregar una canción a su carrito después de proporcionar un extracto de 10 segundos.
 Collector Task		Un usuario está de viaje y quiere ver las películas mejor calificadas en su ubicación actual. La lista de las películas mejor calificadas se muestra después de recopilar su ubicación.

Fuente: elaboración propia con base en Yousfi *et al.* (2016).

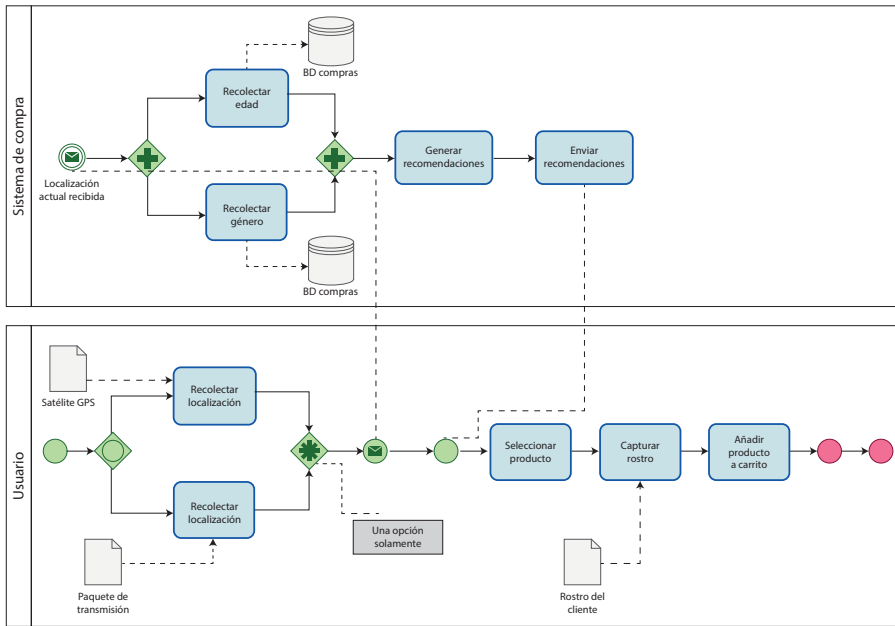


Figura 12.2. Ejemplo de modelo uBPMN

Fuente: elaboración propia con base en Yousfi *et al.* (2016) desarrollada con la herramienta Bizagi Modeler.

► 12.3. Desarrollo de aplicaciones *blockchain*

La tecnología *blockchain* permite un entorno descentralizado que almacena registros digitales dentro de bloques de confiabilidad mejorada con una seguridad única donde cada bloque se conecta con un código *hash* del bloque anterior, formando una cadena de bloques (Faruk *et al.*, 2022a; Shivers *et al.*, 2021).

El proceso de desarrollo para recopilar requisitos, analizar, diseñar, desarrollar, probar e implementar aplicaciones de cadena de bloques aún está emergiendo y se está investigando (Chakraborty *et al.*, 2018; Marchesi *et al.*, 2018).

Se han aplicado y adaptado modelos de procesos de software como cascada, ágil, DevOps y Rational Unified Process (**RUP**) al desarrollo orientado a *blockchain*. Sin embargo, como *blockchain* se ejecuta en una red de nodos *peer-to-peer* (P2P) donde los resultados y el estado deben ser los

mismos en todos los nodos, el proceso de desarrollo debe mejorarse en función de las especificidades de *blockchain*.

Un modelo iterativo e incremental basado en *scrum* para el desarrollo de software orientado a *blockchain* denominado **ABCDE** (Marchesi *et al.*, 2020) integra los componentes de la cadena de bloques, incluidos contratos inteligentes, bibliotecas, estructuras de datos y componentes de la cadena, que en conjunto constituyen un sistema de software descentralizado.

La figura 12.3 muestra el modelo **ABCDE** que se centró en la cadena de bloques de Ethereum y el lenguaje Solidity. El marco se inicia con el objetivo del sistema, encontrar actores e historias de usuarios, diseñar el contrato inteligente, codificar y probar tanto el contrato inteligente como el sistema de aplicaciones, y finalmente integrar, probar e implementar el sistema descentralizado (Faruk *et al.*, 2022b).

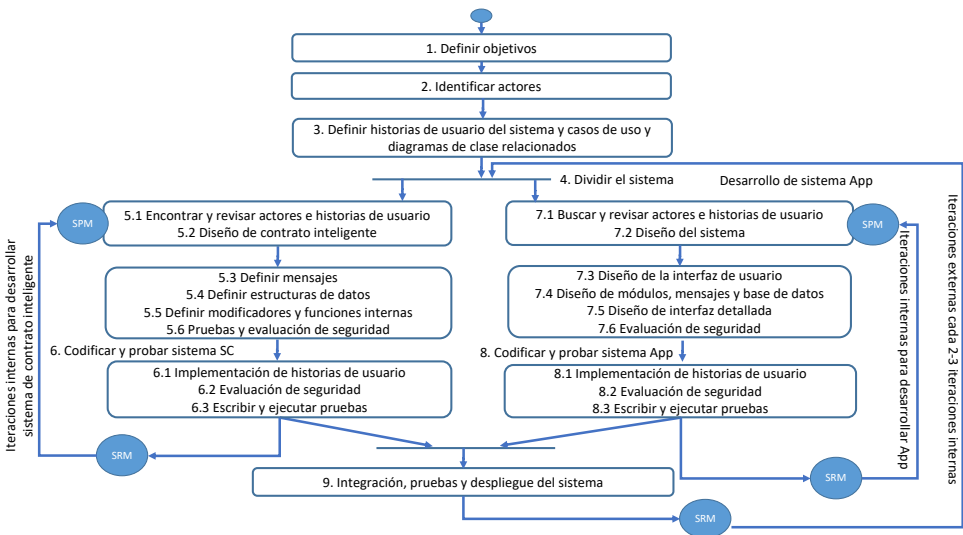


Figura 12.3. Modelo ABCDE para el desarrollo *blockchain* basado en Ethereum

Fuente: elaboración propia con base en Faruk *et al.* (2022a).

Ethereum como marco de trabajo y Solidity como lenguaje de programación es una de las cadenas de bloques más populares. Sin embargo, actualmente, la cadena de bloques Hyperledger está ganando bastante atención debido a su solución empresarial junto con la naturaleza de la cadena de bloques de permisos y consorcios.

► 12.4. Desarrollo de aplicaciones de software sostenibles

Los impactos causados por las tecnologías de la información y de la comunicación (**TIC**) en el medio ambiente se pueden clasificar en tres categorías (Berkhout y Hertin, 2001):

- Impactos de primer orden: Impactos obvios que resultan directamente de la producción y el uso de infraestructura y dispositivos **TIC**, como utilización de recursos, contaminación, consumo de electricidad y eliminación de desechos electrónicos.
- Impactos de segundo orden: Impactos indirectos del uso de las **TIC**, como la conservación de recursos y energía, causados por desmaterialización, desmovilización y sustitución de bienes de información por bienes tangibles.
- Impactos de tercer orden: Impactos indirectos del uso de las **TIC** que aparecen a largo plazo, como cambios en los estilos de vida y sistemas de valores. Estos impactos pueden compensar en exceso el ahorro de energía de las **TIC** (efectos de rebote).

Un modelo de referencia para software *green* y sostenible denominado **GREENSOFT** (Naumann *et al.*, 2011) realiza un refinamiento de los impactos ambientales de las **TIC** para cubrir los problemas de sostenibilidad social en lugar de limitarlos a los problemas ambientales. Los efectos fueron identificados como:

- Efectos de la oferta de **TIC** (representan efectos de primer orden).
- Efectos del uso de las **TIC** (representan efectos de segundo orden).
- Efectos sistémicos de las **TIC** (representan efectos de tercer orden).

El modelo **GREENSOFT** también apoya las diferentes *stakeholders* de un producto de software (desarrolladores, administradores, usuarios) en su desarrollo, mantenimiento y uso de manera sostenible. El modelo presentado en la figura 12.4 consiste en cuatro partes:

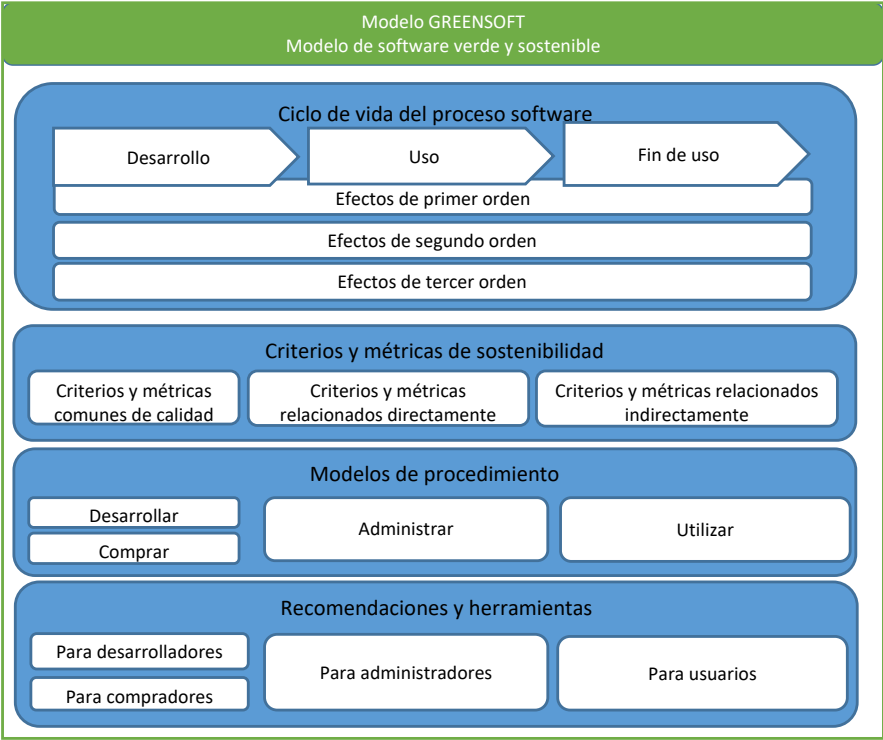


Figura 12.4. Modelo GREENSOFT

Fuente: elaboración propia con base en Naumann *et al.* (2011).

- *Ciclo de vida de los productos de software*: Basado en los tres niveles de impacto, dirige a los *stakeholders* a considerar los impactos en el desarrollo sostenible durante las fases de desarrollo, distribución, uso, desactivación y eliminación.
- *Criterios y métricas de sostenibilidad*: Define métricas y criterios útiles para la evaluación de la sostenibilidad de un producto de software.
- *Modelos de procedimiento*: Clasifica el ciclo de vida del software en cuatro modelos de subprocedimiento y propone un conjunto de actividades y procesos que están orientados hacia el desarrollo sostenible en cada modelo. Los cuatro submodelos son desarrollar, comprar, administrar y utilizar.

- *Recomendaciones y herramientas:* Esta parte guía a los *stakeholders* de un producto de software para que cumplan con las pautas y los procedimientos ecológicos al tratar con el producto de software.

Se ha destacado que el modelo **GREENSOFT** no incluye las etapas de requisitos y pruebas, además, recomienda agregar una etapa de análisis *green* (Berkhout y Hertin, 2001):

- *Etapas de requisitos:* Los ingenieros de requisitos definen las funcionalidades. Por tanto, se reducirá la tasa de cambios en el software y se conservarán los recursos para los cambios.
- *Etapas de prueba:* Ayuda a evaluar el cumplimiento de los requisitos del producto de software.
- *Etapas de análisis green:* Su objetivo es medir el “verdor” de los productos de otras etapas y permitir volver a etapas anteriores para aplicar cambios que ayuden al desarrollo sostenible.

► 12.5. Desarrollo de aplicaciones de *machine learning*

Una de las aplicaciones ampliamente difundidas y utilizadas en la industria del software es la integración de capacidades de **IA** basadas en los avances en el *machine learning* (aprendizaje automático). La **IA** incluye tecnologías para el razonamiento, la resolución de problemas, la planificación y el aprendizaje, entre otras. El aprendizaje automático se refiere a las técnicas de modelado estadístico que han impulsado el mercado de software.

Existen tres diferencias fundamentales en la creación de aplicaciones y plataformas para el entrenamiento y uso de modelos de aprendizaje automático (Amershi *et al.*, 2019):

- El aprendizaje automático tiene que ver con datos: La cantidad de esfuerzo y rigor necesarios para descubrir, generar y administrar datos son más complejos que hacer lo mismo con el código de software.
- La creación para adaptar y extender los modelos requiere que los equipos no solo tengan habilidades de ingeniería de software, sino que requieren un conocimiento suficiente de aprendizaje automático para construir, evaluar y ajustar modelos.
- Puede ser difícil mantener límites estrictos entre los componentes de aprendizaje automático: Los modelos de aprendizaje automáti-

co pueden ser complejos y los componentes se afectan unos a otros durante el entrenamiento y ajuste.

A pesar de estas diferencias, los procesos tienen en común la esencia de un enfoque centrado en datos y los bucles de retroalimentación entre las diferentes etapas. La figura 12.5 muestra una vista simplificada del flujo de trabajo que consta de nueve etapas.

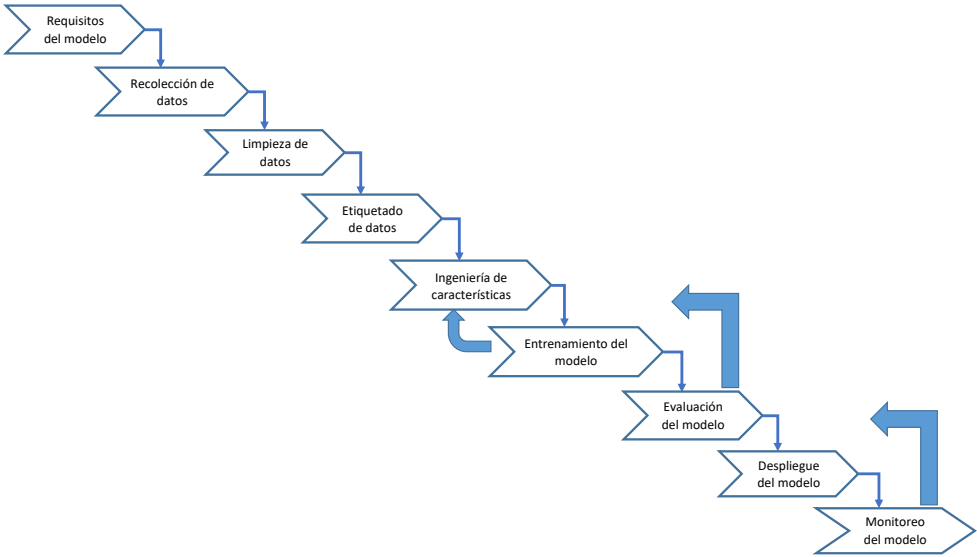


Figura 12.5. Proceso de *machine learning*

Fuente: elaboración propia con base en Amershi *et al.* (2019).

Algunas etapas están orientadas a datos (recopilación, limpieza y etiquetado) y otras están orientadas a modelos (requisitos, ingeniería de características, evaluación, implementación y monitoreo). Hay varios bucles de retroalimentación en el flujo de trabajo. Las flechas de retroalimentación más grandes indican que la evaluación y el monitoreo pueden retroceder a cualquiera de las etapas anteriores. La flecha de retroalimentación más pequeña ilustra que el entrenamiento del modelo puede regresar a la ingeniería de características.

En la etapa de requisitos del modelo, los diseñadores deciden qué características son factibles de implementar con el aprendizaje automático y cuáles son útiles para un producto existente o nuevo. También deciden

qué tipos de modelos son los más apropiados para el problema. Durante la recopilación de datos, los equipos buscan e integran conjuntos de datos disponibles o recopilan datos propios. La limpieza de datos implica la eliminación de registros inexactos o con ruido.

El etiquetado de datos asigna etiquetas reales a cada registro. La mayoría de las técnicas de aprendizaje supervisado requieren etiquetas para inducir un modelo. Las etiquetas pueden ser proporcionadas por los desarrolladores o expertos del dominio.

La ingeniería de características se refiere a las actividades que se realizan para extraer y seleccionar características para los modelos de aprendizaje automático. Para algunos modelos (por ejemplo, redes neuronales convolucionales), esta etapa es menos explícita y, a menudo, se combina con la siguiente etapa, el entrenamiento del modelo. Durante el entrenamiento, los modelos elegidos se entrenan y ajustan con datos limpios recopilados y sus respectivas etiquetas. Luego, en la evaluación del modelo, se evalúa la salida en conjuntos de datos probados o protegidos utilizando métricas predefinidas. Para dominios críticos, esta etapa podría implicar evaluación humana. Luego, el código de inferencia del modelo se implementa en los dispositivos y se monitorea para detectar posibles errores durante su ejecución.

Para simplificar, el proceso es lineal; sin embargo, los flujos de trabajo de aprendizaje automático son altamente no lineales y contienen varios bucles de retroalimentación. Por ejemplo, si los desarrolladores notan que hay un gran cambio de distribución entre los datos de entrenamiento y los datos reales, es posible que deseen regresar y recopilar datos más representativos y volver a ejecutar el flujo de trabajo.

De manera similar, pueden revisar sus elecciones de modelado realizadas en la primera etapa. Si bien los bucles de retroalimentación son típicos en los procesos ágiles, la peculiaridad del flujo de trabajo de aprendizaje automático está relacionada con la cantidad de experimentación necesaria para converger en un buen modelo para el problema. De hecho, el trabajo diario de un desarrollador que realiza aprendizaje automático implica iteraciones frecuentes sobre el modelo seleccionado, los hiperparámetros y el refinamiento del conjunto de datos.

► 12.6. Desarrollo de aplicaciones para automatización industrial

El concepto de automatización está asociado con la eliminación o disminución de la intervención de seres humanos en los diferentes procesos de producción al aplicar sistemas mecánicos, electrónicos y computarizados, para operar y controlar la producción con mayor eficiencia y eficacia (Dubey, 2011).

Existen varias clases de productos de automatización (ARC, 2012) y la mayoría de estos productos son intensivos en software, ya que su funcionalidad está determinada en gran medida por el software ejecutado en los computadores integrados (Vyatkin, 2013).

Un proyecto de aplicación de automatización implica la cooperación de personas involucradas en diversos diseños: procesos, instalaciones, eléctrico, instrumentación y automatización. Estos proyectos construyen aplicaciones de control para monitorear y controlar sistemas complejos, como centrales eléctricas, refinerías, etc. Las fases del ciclo de vida del desarrollo de aplicaciones de automatización se muestran en la figura 12.6.

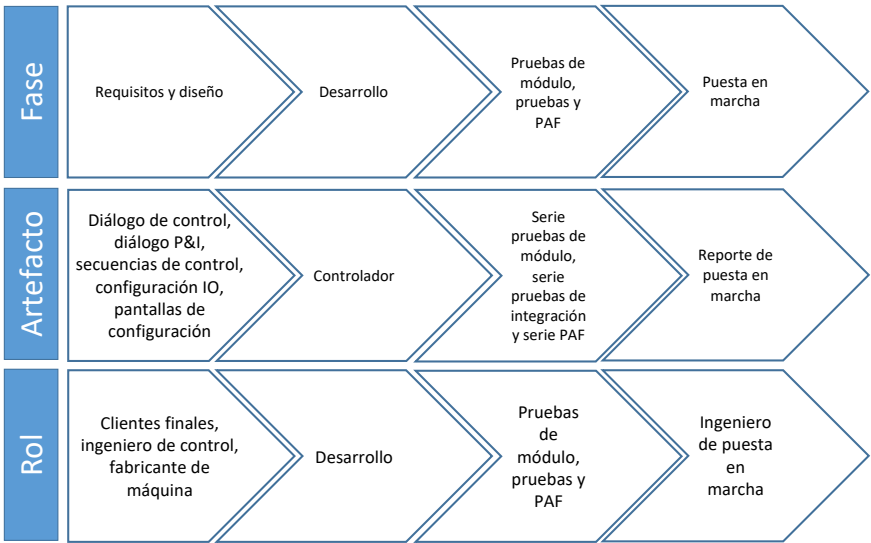


Figura 12.6. Ciclo de vida de desarrollo de aplicaciones de automatización

Fuente: elaboración propia con base en Dubey (2011).

El proceso contiene cuatro fases principales: a) requisitos y diseño, b) desarrollo, c) prueba y d) despliegue y puesta en servicio. El modelo también incluye los artefactos (involucrados o generados) en cada fase y los roles responsables de cada fase. También hay una larga fase de mantenimiento después de la fase de puesta en marcha (que no se muestra en la figura 12.6):

- ▶ *Requisitos y diseño*: Los aspectos operativos se recogen durante esta fase. Esta fase ayuda a tomar decisiones de inversión en términos de selección y compra de *hardware*. La recopilación de requisitos y el diseño a menudo parece borrosa y pueden combinarse. Se desarrolla un diseño de alto nivel y un diseño más detallado denominado diagramas de proceso e instrumentación (**P&ID**). Esta fase es iterativa hasta que se finaliza el diseño entre los clientes y el equipo de desarrollo.
- ▶ *Desarrollo*: Implica el desarrollo de una aplicación de control y la interfaz hombre-máquina (**IHM**). Un desarrollador de control se enfoca en el aspecto de la lógica de control, mientras un desarrollador de **IHM** se enfoca en la presentación gráfica de varias conexiones, flujos, dispositivos y los datos en línea recopilados por estos dispositivos. Los diagramas de diseño y la lista de entrada/salida (**E/S**) son los artefactos de entrada a la fase de desarrollo.
- ▶ *Pruebas*: La solución desarrollada pasa por varios niveles de prueba antes de su puesta en marcha, como a) pruebas de módulo, b) pruebas de integración, c) pruebas de aceptación en fábrica (**PAF**) y d) pruebas de aceptación en sitio (**PAS**). Las pruebas de módulos son equivalentes a las pruebas unitarias en las que se escriben casos de prueba para validar la funcionalidad de los programas de control individuales. La solución integral se prueba con los requisitos **PAF**. La configuración de **PAF** es una combinación de simuladores de software y *hardware* real. Los clientes participan de las ejecuciones de **PAF** en las que validan el conjunto de secuencias de operaciones y la apariencia de varias pantallas **IHM** del operador. El cliente debe aprobar una ejecución de **PAF** para que la solución pase a la siguiente fase. Después de **PAF**, el *hardware* se envía al sitio de la planta y comienza la fase de puesta en marcha. **PAS** tiene lugar durante la fase de puesta en marcha.
- ▶ *Puesta en marcha*: Consiste en configurar el *hardware*, los canales de comunicación y las estaciones de trabajo en la planta. Continúa hasta que el sistema implementado se estabiliza y los clientes

están satisfechos con la aplicación. La parte inicial de la puesta en marcha está dedicada a la configuración y conexión, y la última parte está dedicada a ajustar la aplicación con datos en tiempo real. Implica la interacción entre el cliente, los ingenieros de puesta en marcha y el representante de dispositivos de terceros. En la puesta en marcha, los dispositivos y controladores están conectados y la aplicación se inicia con señales/datos reales. Las aplicaciones de automatización pasan por una fase de puesta en marcha virtual antes de que pueda comenzar la actividad real.

► 12.7. Desarrollo de aplicaciones para minería de datos

Un modelo de procesos utilizado para el desarrollo de aplicaciones orientadas a la minería de datos es Cross-Industry Standard Process for Data Mining (**CRISP-DM**), el cual es independiente de la industria, las herramientas y aplicaciones. Consta de seis fases iterativas de entendimiento del negocio hasta el despliegue. La tabla 12.3 describe brevemente la idea principal, las tareas y el resultado de estas fases, según la guía del usuario de **CRISP-DM** (Chapman *et al.*, 2000; Wirth y Hipp, 2000).

Tabla 12.3. Descripción del modelo de procesos CRIPS-DM

Fase	Aplicación
Entendimiento del negocio	La situación del negocio debe evaluarse para obtener una visión general de los recursos disponibles y necesarios. En esta fase, se determinan los objetivos, tipos y criterios de éxito de la minería de datos. También se crea un plan de proyecto.
Entendimiento de datos	La recopilación de datos, su exploración y descripción y la verificación de la calidad de los datos son tareas de esta fase.
Preparación de datos	La selección de datos debe realizarse mediante la definición de criterios de inclusión y exclusión. Se aplica limpieza de datos para mejorar su calidad.
Modelado	Consiste en seleccionar la técnica de modelado, construir el caso de prueba y el modelo. La selección depende del problema y de los datos. Para construir el modelo, se deben establecer parámetros específicos.
Evaluación	Los resultados se comparan con los objetivos de negocio definidos. Los resultados deben interpretarse y deben definirse acciones adicionales.
Despliegue	Describe de forma general la guía del usuario por medio de un informe final o un componente de software. Además, en esta fase se planifica la implementación, la supervisión y el mantenimiento.

Fuente: elaboración propia con base en Chapman *et al.* (2000).

El proceso **CRISP-DM** se describe en términos de un modelo de procesos jerárquico, que consta de conjuntos de tareas descritas en cuatro niveles de abstracción (de general a específico): fase, tarea genérica, tarea especializada e instancia de proceso (figura 12.7).

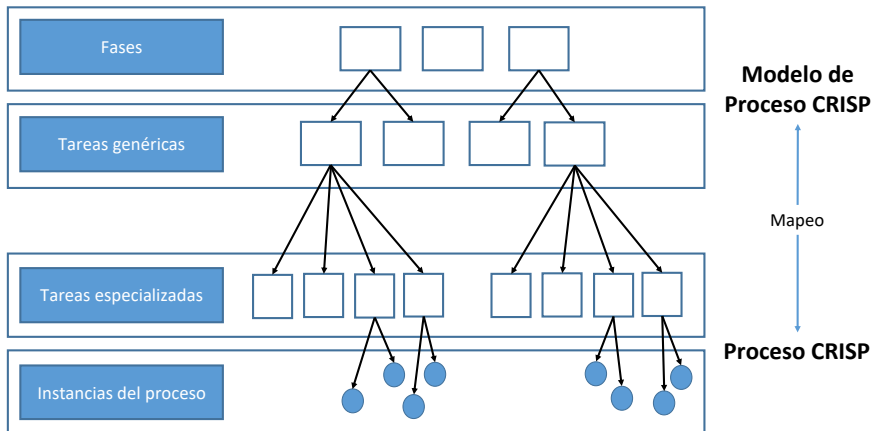


Figura 12.7. Ciclo de vida de desarrollo de aplicaciones de automatización

Fuente: elaboración propia con base en Dubey (2011).

En el nivel superior, el proceso de minería de datos se organiza en varias fases. Cada fase consta de varias tareas genéricas de segundo nivel. Este segundo nivel se llama genérico, por ser lo suficientemente general para cubrir las situaciones posibles de minería de datos. Las tareas genéricas pretenden ser completas (cubren tanto el proceso completo de minería de datos como las aplicaciones de minería de datos) y estables (el modelo debe ser válido para desarrollos aún imprevistos, como nuevas técnicas de modelado).

El tercer nivel de tareas especializadas describe cómo se deben llevar a cabo las acciones de las tareas genéricas en situaciones específicas. Por ejemplo, puede haber una tarea genérica llamada limpieza de datos y el tercer nivel describe cómo esta difiere en diferentes situaciones, como la limpieza de valores numéricos frente a la limpieza de valores categóricos. La descripción de fases y tareas como pasos discretos realizados en un orden específico representa una secuencia idealizada de eventos. En la práctica, muchas de las tareas se pueden realizar en un orden diferente y, a menudo, será necesario retroceder repetidamente a tareas anteriores y repetir ciertas acciones.

El cuarto nivel, la instancia de proceso, es un registro de las acciones, las decisiones y los resultados de un compromiso real de minería de datos. Una instancia de proceso se organiza de acuerdo con las tareas definidas en los niveles superiores, pero representa lo que realmente sucedió en un compromiso particular, en lugar de lo que sucede en general.

► Desafíos

El desarrollo de software es una actividad crítica en todos los ámbitos de la sociedad en los que cada vez se depende más de software en los productos y servicios modernos. Cada producto o servicio tiene un conjunto específico de requisitos que no necesariamente se pueden desarrollar aplicando procesos tradicionales, procesos ágiles o estándares de la industria. Para ello, los modelos de procesos de software específicos a las aplicaciones informáticas a desarrollar han sido propuestos o adaptados.

En general, los investigadores y profesionales de los procesos de software deberían reutilizar las experiencias y los logros de otras áreas y disciplinas. El desarrollo de estas aplicaciones específicas tiene algunos de los desafíos que ya se han mencionado en capítulos anteriores como la incorporación de métricas del proceso, la falta de herramientas de soporte para el proceso y la carencia de estudios empíricos sobre el impacto de los procesos definidos en el desarrollo de las aplicaciones específicas. Además, la mayoría de las tecnologías desarrolladas por la comunidad de procesos de software no se han transferido al uso industrial (Fuggetta, 2000).

► Ejemplos

Deep Intelligence (2022) es una plataforma diseñada para ayudar a seleccionar algoritmos óptimos de IA y aprendizaje automático para el análisis de conjuntos de datos. Se puede personalizar para leer cualquier tipo de datos provenientes de diversas fuentes, como web, archivos, bases de datos, sensores, etc. Es un entorno de visualización fácil de usar e intuitivo que sirve de guía en la creación y configuración de algoritmos que analizan datos. Además, la plataforma permite crear cuadros de mando para mejorar la experiencia de visualización y se pueden integrar con cualquier otra aplicación en línea.

Entre sus principales características se encuentran:

- ▶ Facilita la adquisición de datos (admite múltiples fuentes).
- ▶ Reduce el tiempo de programación y pruebas (utilizando asistentes).
- ▶ Facilita la difusión de los resultados (se pueden compartir libremente).
- ▶ Mantiene actualizados tanto los resultados del modelo como las visualizaciones cuando la fuente cambia.
- ▶ Monitorización en tiempo real (crea cuadros de mando).
- ▶ No necesita personal experto en programación o IA (solo necesita conocer la lógica del negocio).

La plataforma incluye algoritmos supervisados, como clasificación, regresión, series temporales, clasificación de textos y de imágenes; y algoritmos no supervisados, como agrupamiento, reducción de dimensionalidad, reglas de asociación y similitud.

En la figura 12.8, se presenta la página principal de Deep Intelligence.

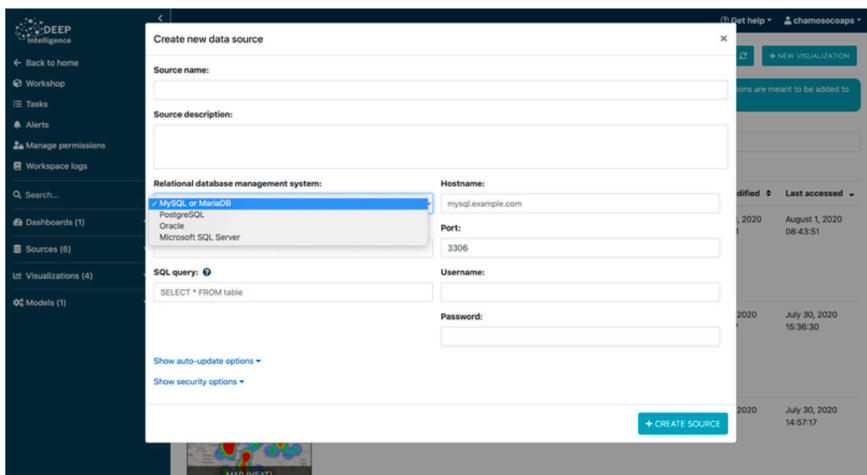


Figura 12.8. Página principal de Deep Intelligence

Fuente: Deep Intelligence (2022).

En la figura 12.9, se presenta un ejemplo con datos de la pandemia de covid-19.

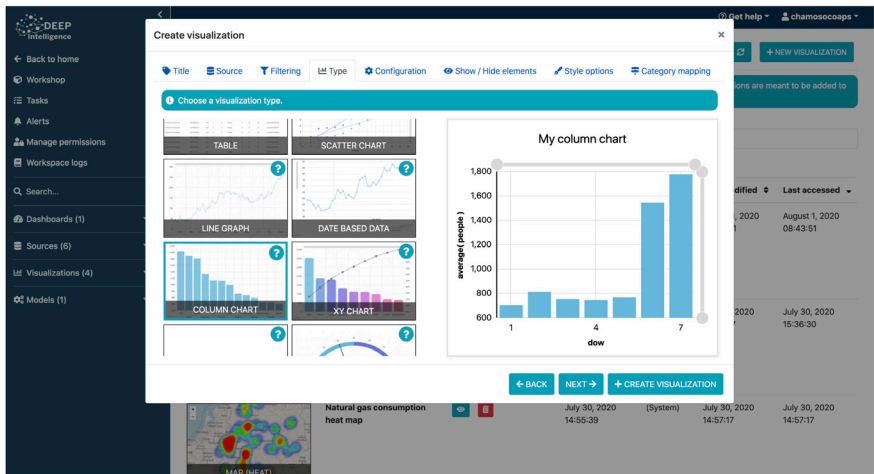


Figura 12.9. Deep Intelligence con datos de la pandemia de covid-19

Fuente: Deep Intelligence (2022).

► Resumen

Existen numerosas aplicaciones informáticas que implementan procesos específicos de desarrollo de software. Los procesos han sido adaptados a las necesidades particulares de cada tipo de aplicación.

El desarrollo de aplicaciones de reconocimiento de patrones generalmente tiene las fases de adquisición de datos, preprocesamiento, extracción de características, clasificación y cuentan con una base de datos.

El modelado de aplicaciones orientadas al internet de las cosas ha modificado el estándar **BPMN** para añadirle notaciones gráficas ubicuas generando un nuevo lenguaje denominado uBPMN (**BPMN** ubicuo). Este lenguaje incluye nuevos elementos gráficos, como tareas sensor, lector, recolector, imagen y audio.

El desarrollo de aplicaciones *blockchain* utiliza un modelo denominado **ABCDE** que integra los componentes de la cadena de bloques, incluidos contratos inteligentes, bibliotecas, estructuras de datos y componentes de la cadena, que en conjunto constituyen un sistema de software descentralizado.

El desarrollo de aplicaciones de software sostenibles utiliza el modelo **GREENSOFT** que incluye un ciclo de vida de los productos, criterios y mé-

tricas de sostenibilidad, modelos de procedimiento y recomendaciones y herramientas.

El desarrollo de aplicaciones de *machine learning* se basa en un proceso que contiene las fases de requerimientos del modelo, recolección de datos, limpieza de datos, ingeniería de características, entrenamiento, evaluación, despliegue y monitoreo del modelo.

El desarrollo de aplicaciones de automatización industrial se realiza a partir de un proceso que contiene cuatro fases principales: requisitos y diseño, desarrollo, prueba y despliegue y puesta en servicio.

El desarrollo de aplicaciones para minería de datos puede utilizar el modelo de procesos jerárquico **CRISP-DM**, el cual consta de conjuntos de tareas descritas en cuatro niveles de abstracción: fases, tareas genéricas, tareas especializadas e instancias de procesos.

► Ejercicios

1. Describa las propiedades de la tecnología *blockchain*.
2. ¿En qué se diferencia el desarrollo de aplicaciones *blockchain* de otras tecnologías?
3. ¿En qué consiste el desarrollo de software sostenible?
4. Describa los niveles jerárquicos del modelo de procesos **CRISP-DM** para minería de datos.
5. Describa las fases del proceso de reconocimiento de patrones.
6. Desarrolle un diagrama uBPMN para modelar un dispositivo médico que puede ayudar en el monitoreo en tiempo real de pacientes de forma remota. El dispositivo informa de una emergencia, como un ataque de asma, insuficiencia cardíaca, etc., a personal médico. Por consiguiente, los médicos cuentan con información para trasladarse al lugar donde se encuentra la emergencia con base en la información GPS.
7. ¿Cuáles son las ventajas de la automatización industrial?
8. ¿Por qué existe un bucle de retroalimentación entre las fases modelo de entrenamiento e ingeniería de características en el modelo de procesos para *machine learning*?

► Referencias

- Abouzid, I. y Saidi, R. (2019). Proposal of **BPMN** extensions for modelling manufacturing processes. En *2019 5th International Conference on Optimization and Applications (ICOA)* (pp. 1-6). **IEEE**. <https://doi.org/10.1109/ICOA.2019.8727651>
- Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E. ... y Zimmermann, T. (2019). Software engineering for machine learning: A case study. En *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (pp. 291-300). **IEEE**. <https://doi.org/10.1109/ICSE-SEIP.2019.00042>
- ARC**. (2012). *Automation expenditures for discrete industries*. <https://www.arcweb.com/market-studies/automation-software-expenditures-discrete-industries>
- Berkhout, F. y Hertin, J. (2001). *Impacts of information and communication technologies on environmental sustainability: Speculations and evidence*. **OECD**. <https://www.oecd.org/science/inno/1897156.pdf>
- Chakraborty, P., Shahriyar, R., Iqbal, A. y Bosu, A. (2018). Understanding the software development practices of blockchain projects: A survey. En *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1-10). <https://doi.org/10.1145/3239235.3240298>
- Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C. y Wirth, R. (2000). **CRISP-DM** 1.0: Step-by-step data mining guide. *SPSS Inc*, 9(13), 1-73.
- Deep Intelligence. (2022). <https://app.deepint.net/shared/c9d55a1f-6ec9daf4-0dd31924-17429a774d9/dashboards/000001768ad911aa-d0711434-b94233c3-e0e-32b99?lang=en#1>
- Dubey, A. (2011). Evaluating software engineering methods in the context of automation applications. En *2011 9th IEEE International Conference on Industrial Informatics* (pp. 585-590). **IEEE**. <https://doi.org/10.1109/INDIN.2011.6034944>
- Faruk, M. J. H., Islam, M., Alam, F., Shahriar, H. y Rahman, A. (2022a). Bie Vote: A Biometric Identification Enabled Blockchain-Based Secure and Transparent Voting Framework. En *2022 Fourth International Conference on Blockchain Computing and Applications (BCCA)* (pp. 253-258). **IEEE**. <https://doi.org/10.1109/BCCA55292.2022.9922588>
- Faruk, M. J. H., Subramanian, S., Shahriar, H., Valero, M., Li, X. y Tasnim, M. (2022b). Software engineering process and methodology in blockchain-oriented software development: A systematic study. En *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA)* (pp. 120-127). **IEEE**. <https://doi.org/10.1109/SERA54885.2022.9806817>
- Fuggetta, A. (2000). Software process: A roadmap. En *Proceedings of the Conference on the Future of Software Engineering* (pp. 25-34). <https://dl.acm.org/doi/pdf/10.1145/336512.336521>
- Jain, A. K., Duin, R. P. W. y Mao, J. (2000). Statistical pattern recognition: A review. **IEEE Transactions on pattern analysis and machine intelligence**, 22(1), 4-37. <https://doi.org/10.1109/34.824819>
- Jain, A. K., Flynn, P. y Ross, A. A. (eds.) (2008). *Handbook of biometrics*. Springer.
- Marchesi, L., Marchesi, M. y Tonelli, R. (2020). **ABCDE**: Agile block chain DApp

- engineering. *Blockchain: Research and Applications*, 1(1-2), 100002. <https://doi.org/10.1016/j.bcr.2020.100002>
- Marchesi, M., Marchesi, L. y Tonelli, R. (2018). An agile software engineering method to design blockchain applications. En *Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia* (pp. 1-8). <https://doi.org/10.1145/3290621.3290627>
- Naumann, S., Dick, M., Kern, E. y Johann, T. (2011). The **GREENSOFT** Model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1(4), 294-304. <https://doi.org/10.1016/j.suscom.2011.06.004>
- Shivers, R., Rahman, M. A., Faruk, M. J. H., Shahriar, H., Cuzzocrea, A. y Clincy, V. (2021). Ride-hailing for autonomous vehicles: Hyperledger fabric-based secure and decentralize blockchain platform. En *2021 IEEE International Conference on Big Data (Big Data)* (pp. 5450-5459). **IEEE**. <https://doi.org/10.1109/BigData52589.2021.9671379>
- Vyatkin, V. (2013). Software engineering in industrial automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics*, 9(3), 1234-1249. <https://doi.org/10.1109/TII.2013.2258165>
- Watanabe, S. (1985). *Pattern recognition: Human and mechanical*. John Wiley & Sons.
- Wirth, R. y Hipp, J. (2000). **CRISP-DM**: Towards a standard process model for data mining. En *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining* (vol. 1, pp. 29-39). <https://www.cs.unibo.it/~danilo.montesi/CBD/Beatriz/10.1.1.198.5133.pdf>
- Yousfi, A., Batoulis, K. y Weske, M. (2019). Achieving business process improvement via ubiquitous decision-aware business processes. *ACM Transactions on internet Technology (TOIT)*, 19(1), 1-19. <https://doi.org/10.1145/3298986>
- Yousfi, A., Bauer, C., Saidi, R. y Dey, A. K. (2016). uBPMN: A **BPMN** extension for modeling ubiquitous business processes. *Information and Software Technology*, 74, 55-68. <https://doi.org/10.1016/j.infsof.2016.02.002>



Capítulo 13

Casos de éxito

► Objetivos

Los objetivos de este capítulo consisten en describir algunos casos de éxito de implementación de nuevos procesos de software en empresas reconocidas del sector de software a nivel mundial. Al estudiar este capítulo, el lector:

- Comprenderá los problemas y desafíos que tuvieron las empresas al abordar la definición e implantación de nuevos procesos de software.
- Entenderá cómo algunas empresas implementaron prácticas ágiles en sus entornos operativos de producción de software.
- Asimilará cómo las empresas aplican métricas y conceptos de control estadístico del proceso de software en sus proyectos.

El grupo Standish ha realizado investigaciones sobre el alcance de los problemas con los proyectos de software desde mediados de la década de 1990. El reporte **CHAOS** del Standish Group (2020) es uno de los informes más famosos sobre el estado actual de la industria del software. Este estudio se realizó en Estados Unidos, pero sus resultados se podrían aplicar a diversas empresas de otras partes del mundo. Los resultados indican serios problemas con la entrega a tiempo de los proyectos o la

cancelación de proyectos. En la tabla 13.1, se presenta un extracto de ese informe que indica que las tasas de éxito de los proyectos ágiles son mejores que los proyectos basados en el modelo en cascada. A su vez, la cantidad de proyectos que falló fue menor utilizando enfoques ágiles.

Tabla 13.1. Resumen de proyectos de éxito con desafíos y con fallos del reporte CHAOS

Método	Exitoso (%)	Con desafíos (%)	Con fallos (%)
Ágil	42	47	11
Cascada	13	59	28

Fuente: elaboración propia con base en Standish Group (2020).

Sin embargo, es bueno que los interesados en los procesos de software conozcan casos de éxito de reconocidas empresas de la industria del software. Para ello, en este capítulo se presentará un listado de casos de estudio exitosos relacionados con los conceptos de procesos de software descritos.

Un caso de estudio orientado a la ingeniería de software “recopila datos detallados del proyecto para determinar si el producto desarrollado es más fácil de producir que proyectos similares en el pasado” (Zelkowitz, 2009). En un caso de estudio, los proyectos se monitorean y se utilizan métodos de “observación” para recopilar datos relevantes durante el ciclo de vida del estudio (Zelkowitz y Wallace, 1998). Estas experiencias reales se pueden utilizar para mejorar las prácticas de ingeniería de software (Garzás y Paulk, 2013).

► **13.1. Caso de estudio: proyecto Sentinel del FBI**

Cada investigación desarrollada por el Federal Bureau of Investigation (**FBI**) tiene un archivo con los registros que se crean como parte de las investigaciones realizadas en cada caso. En 2003, el **FBI** decidió digitalizar los casos y automatizar los procesos relacionados. Para ello, se desarrolló un proyecto denominado Sentinel (Olszewski y Wingreen, 2011).

El proyecto inició en marzo de 2006 con una base de usuarios finales de más de 30.000 agentes, analistas y empleados administrativos. Las estimaciones originales del proyecto fueron US\$451 millones en costos y una fecha de finalización en diciembre de 2009. El proyecto se desarrollaría en cuatro fases y el **FBI** contrató a la empresa Lockheed Martin, que propuso utilizar un proceso de desarrollo de software tradicional (cascada).

Para agosto de 2010, el **FBI** había gastado US\$405 millones del presupuesto; sin embargo, la funcionalidad entregada solo correspondía a dos de las cuatro fases de Sentinel. Debido a los excesos de costo y cronograma, el **FBI** emitió una orden de suspensión de trabajos en julio de 2010.

Para concluir con éxito el proyecto, el **FBI** optó por adoptar un nuevo enfoque para obtener mejores resultados: aplicar *scrum*. Una auditoría había concluido previamente que el **FBI** necesitaría US\$35 millones adicionales y seis años más si hubiera continuado con su enfoque tradicional. El **FBI** le dijo al inspector general del Departamento de Justicia que creía que podría completar Sentinel con el presupuesto restante y dentro de los 12 meses posteriores a la reanudación.

El **FBI** redujo el personal de Sentinel de 400 a 45 personas, 15 de las cuales eran programadores. Un nuevo Chief Technology Officer (**CTO**) del **FBI** dirigió el proyecto consiguiendo el objetivo de ofrecer algunas de las funciones de Sentinel cada 30 días. Para noviembre de 2011, un año después de reiniciar con el nuevo enfoque, se habían completado todas las fases de Sentinel. El **FBI** pudo terminar el proyecto por US\$30 millones en 12 meses y un ahorro de costos de más del 90%. En la tabla 13.2, se presenta un resumen de los diferentes procesos aplicados en el proyecto.

Tabla 13.2. Procesos de software aplicados en el proyecto Sentinel

Atributo	Cascada	Ágil
Duración	4 años	12 meses
Cantidad de personas	400	45
Presupuesto (en millones de dólares)	US\$405	US\$30
Resultados	2 fases de 4	4 fases de 4 (terminado)

Fuente: elaboración propia con base en Olszewski y Wingreen (2011).

► 13.2. Caso de estudio: Intel

En Intel, los equipos se organizaban en “silos” funcionales con entregas programadas regulares de entregables a otros equipos. El resultado fue que algunos equipos llevaban una carga inusual de trabajo al final de las fases del ciclo de vida y tenía una rotación alta al final de un proyecto. Cada equipo estaba compuesto por expertos en el dominio cuyas habilidades rara vez se superponían con las de los miembros de su equipo. Por tanto, era difícil o imposible emparejar tareas o compartir responsabilidades en un equipo.

Para superar estos problemas, Intel decidió que el trabajo se organizara en varias fases. La primera fue de preparación, en la que el grupo de transición inicial incluía seis equipos y numerosos subequipos. Aproximadamente 20 líderes de grupo y líderes técnicos asistieron a un curso de capacitación sobre *scrum master*. Luego, los líderes del equipo acordaron comprometerse durante tres meses a implementar los principios y las prácticas de *scrum* antes de intentar adaptarlo a las necesidades de Intel (Elwer, 2008).

Los líderes de grupo y equipo funcionaron como propietarios de productos para los siete equipos. Al final de los tres meses, había tres *scrum masters* adicionales para administrar siete equipos. Además, un octavo equipo se ofreció como voluntario para comenzar a usar *scrum*.

Después de aproximadamente cinco meses, escalar el trabajo en los equipos de *scrum* se convirtió en uno de los mayores desafíos. Antes de agregar los cinco equipos adicionales, la organización necesitaba más conocimiento sobre cómo administrar las dependencias entre múltiples equipos y facilitar una mejor comunicación entre estos. Por tanto, se realizó una nueva capacitación sobre escalado de *scrum*. El grupo pudo escalar a 12 equipos *scrum*, cada uno con aproximadamente cinco a nueve desarrolladores en un año. Durante la fase de preparación, se establecieron diferentes roles que se muestran en la tabla 13.3.

Durante la segunda fase denominada de supervivencia, los *sprints* de dos semanas eran imposibles de mantener y la mayoría de los equipos de *scrum* optaron por *sprints* de un día. Se reunirían durante una hora todos los días para planificar las próximas 24 horas y revisar y reflexionar sobre lo realizado el día anterior. Sin embargo, los comportamientos centrales de *scrum* se mantenían como la priorización basada en el valor comercial, el tamaño del equipo, no trabajar fuera del *backlog*, la implementación de mejoras de procesos y la revisión del producto del trabajo. Todo en una escala

mucho más pequeña y más rápida a medida que crecía el conocimiento del producto. Este periodo de intensa depuración y desarrollo se prolongó durante algunas semanas. Al final, los *scrums* sobrevivientes emergieron intactos y expandieron sus *sprints* a dos semanas y siguen vigentes hoy (Elwer, 2008).

Tabla 13.3. Roles definidos en el proyecto de Intel

Rol	Descripción
Propietario de empresas	Gerentes sénior o ingenieros principales encargados de la supervisión de varios equipos o de cuestiones técnicas generales para todos los equipos.
Propietario de productos	Generalmente gerentes de grupos funcionales.
Propietario técnico	Líderes técnicos de las áreas funcionales que podrían colaborar en cuestiones de integración, dependencia y arquitectura para garantizar la congruencia entre los equipos con resultados dependientes.
<i>Scrum master</i>	Ingeniero de equipos cruzados que ayudó a frenar la necesidad del <i>scrum master</i> de entrometerse en soluciones técnicas.
Equipo	Equipo encargado de un resultado particular del conjunto de pruebas.
Transitorio	Miembros del grupo con habilidades altamente especializadas que necesitan varios equipos solo para un <i>sprint</i> o dos a la vez.
Conductor	Miembro del equipo que representa a más de una persona, incluidos los supervisores del contratista o los miembros locales de un equipo remoto.
Propietario de la historia	Experto técnico con un conocimiento sobre cómo completar una historia y que puede desarrollar tareas.

Fuente: elaboración propia con base en Elwer (2008).

Durante la tercera fase, preparación para la fabricación, se formaron varios grupos de trabajo para tratar los problemas del producto. Estos grupos se convirtieron en equipos multifuncionales, los cuales se consideraron estructuras organizativas útiles, ya que el conocimiento y la experiencia técnica profunda viven en ellos. También brindaron a los miembros del equipo *scrum* un lugar para “volver a casa” entre proyectos. Se crearon *scrums* de “características” multifuncionales en los que los miembros del equipo multifuncional estaban cien por ciento dedicados y no estaban influenciados por sus gerentes funcionales durante el *sprint*. El éxito de los

equipos multifuncionales llevó a que se agregaran cinco nuevos *scrums* al proceso, elevando el impacto total a 18 *scrums* en dos años.

Se destacó la realización de *sprints* de nueve días y se desarrollaron reuniones de revisión, retrospectivas y de planificación cada dos viernes. De esta manera, el equipo siempre estaba fuera de un *sprint* cada dos fines de semana. Esto ayudó a mejorar la calidad de vida y la moral de los equipos (Elwer, 2008).

La cadencia de *sprint* de nueve días permitió a los propietarios de producto y equipos a cambiar de dirección según fuera necesario, a intervalos frecuentes. Esta cadencia también ayudó a reducir la agitación de los requisitos de proyectos anteriores y los gerentes de alto nivel comenzaron a ver que el equipo podía producir un producto de trabajo real cada dos viernes.

Intel se benefició del cambio al proceso ágil en aspectos como:

- *Tiempo de ciclo reducido*: Se obtuvo una reducción del 66% en el tiempo de ciclo.
- *Desempeño según el cronograma*: Se estableció y se mantuvo una planificación basada en la capacidad y una cadencia de dos semanas durante más de un año.
- *Moral mejorada*: Se mejoró la comunicación y la satisfacción laboral.
- *Mayor transparencia*: La aplicación de *scrum* descubrió errores, impedimentos, herramientas débiles y hábitos de ingeniería deficiente.

► 13.3. Caso de estudio: Google

Google tiene muchos productos exitosos orientados al cliente, como su motor de búsqueda y Gmail, desarrollados en el entorno de las culturas de *start-ups*, en las que la mayoría de las decisiones las toman los propios equipos sin mucha interferencia de la administración. Sus equipos de proyecto no tienen un director de proyecto, sino que se organizan y se comunican directamente con todos sus *stakeholders*. Hoy día, Google tiene más de 6.000 empleados en oficinas en todo el mundo.

La mentalidad general en Google es tener pocos procesos estándar. La razón es que los equipos de ingeniería individuales saben mejor lo que es correcto para ellos. Por otro lado, la alta dirección tiene confianza en sus

ingenieros, quienes no abusan de esta autonomía, sino que hacen lo que es mejor para su proyecto y la empresa.

Sin embargo, surgieron muchos problemas cuando el proyecto AdWords (su software de servicio de publicidad en línea) creció hasta volverse grande con una tasa alta de cambios. AdWords es una aplicación *business-to-business* (B2B), que requiere mucha más dirección comercial que otros productos orientados al consumidor. Además, actualizar AdWords requiere que todas las funciones deben traducirse a muchos idiomas, el material de ventas debe actualizarse, el personal de soporte debe capacitarse y la comunicación externa sobre las principales funciones debe prepararse (foros, blogs y correos).

Además, el código de la aplicación consta de más de 500 KLOC y el equipo de ingeniería está distribuido en cinco oficinas en todo el mundo: constantemente hay entre 15 y 20 proyectos importantes en curso, además de mantenimiento y pequeñas mejoras. Para abordar estos problemas, se introdujeron prácticas ágiles en varios equipos de proyecto en Google. Algunas de las técnicas ágiles introducidas fueron (Striebeck, 2006):

Backlog de lanzamientos y *burndown charts*: Estas dos herramientas brindaron una alta visibilidad del progreso del desarrollo para el equipo del proyecto. Además, el uso de wikis simples para almacenar los trabajos pendientes permitió a los ingenieros actualizar su progreso en poco tiempo. También se midió la tasa de trabajo pendiente por tareas completadas, no por funciones completadas.

Una reunión semanal de punto de control de desarrollo para planificar la próxima semana y trabajar en los cambios de alcance. A estas reuniones de puntos de control asistieron los ingenieros y los directivos de aseguramiento de la calidad, gestión de proyectos e interfaz gráfica.

Al finalizar los proyectos, los equipos seleccionaron los peores problemas a corregir y las mejores prácticas para mantener, como se observa en la tabla 13.4.

El enfoque ágil tuvo resistencia inicialmente debido a la forma de Google de desarrollar software que muchos ingenieros no creían que ningún proceso formal pudiera ser útil. Después de “probar” el enfoque ágil, los equipos del proyecto lo adoptaron y lo pusieron en práctica diariamente. Se mantuvo la cultura organizacional de Google y la autoorganización de los equipos, lográndose gestionar fácilmente varios proyectos en paralelo.

Tabla 13.4. Aspectos positivos y negativos de los proyectos en Google

Aspectos	Descripción
Positivos	Gestión de proyectos y herramientas (<i>burndown charts</i> y <i>backlogs</i>).
	Control temprano de calidad y disponibilidad de un servidor de ensayo.
	Trabajo en equipo y colaboración.
Negativos	Priorización poco clara o inexistente.
	Sensación de que el equipo hubiera perdido la fecha de lanzamiento varias veces.
	Demasiado riesgo al final debido a la acumulación de errores.

Fuente: elaboración propia con base en Striebeck (2006).

► 13.4. Caso de estudio: BBC

Un caso de estudio se desarrolló sobre la aplicación de ideas *lean* para gestionar el desarrollo de software de los procesos de software del departamento de webmedia de BBC Worldwide por parte de un equipo de desarrollo de nueve personas.

La base de *lean* es la eliminación continua de residuos, la cual requiere un enfoque en el flujo de trabajo a través del sistema para garantizar que el material se produzca solo cuando se necesita y en las cantidades exactas requeridas. Permite acercarse a niveles de inventario cercanos a cero, lo que hace que la producción sea más flexible, y también identificar rápidamente las fuentes de defectos (Middleton y Joyce, 2011).

Los datos recopilados en 2009 incluyeron observaciones directas del equipo de desarrollo, tableros *kanban*, reuniones diarias, entrevistas semiestructuradas con una amplia variedad de personal y análisis estadístico.

El equipo supervisó el tiempo que tardaba el trabajo en fluir a través de varias partes del flujo de valor. También realizaron un seguimiento de las medidas de calidad y rendimiento utilizando gráficos de control de procesos estadísticos de series temporales. Estos gráficos tenían dos elementos importantes. Primero, las líneas del límite de control superior horizontal muestran la varianza. Cuanto más alta es la línea, más variación hay en el

proceso, lo que daña gravemente la productividad. En segundo lugar, el promedio se muestra como una línea horizontal a lo largo del tiempo.

La evidencia mostró que durante un periodo de 12 meses el tiempo de entrega del software mejoró en un 37%, su consistencia aumentó en un 47% y los defectos informados por los clientes se redujeron en un 24%. Los datos recopilados contenían 84 características (el 52% eran pequeñas) en los primeros 5 meses y 64 características (el 75% eran pequeñas) en los últimos 5 meses (Middleton y Joyce, 2011).

Los resultados basados en datos cuantitativos y cualitativos mostraron reducciones en el tiempo de entrega, tasas de error y variabilidad, combinado con evidencia de mejora continua. La forma en que se manejaba el trabajo cambió significativamente. El volumen de trabajo que se permitió ingresar al proceso se redujo para garantizar que la carga de trabajo no superara la capacidad del sistema. El tiempo de ciclo reducido significó que el cliente recibió rápidamente entregas incrementales pequeñas y de alto valor. Esto redujo tanto el riesgo técnico como el de mercado. El control estadístico del proceso funcionó bien y proporcionó datos útiles sobre tendencias y variaciones.

El proceso *lean* también proporcionó un marco que fomentó otras mejoras beneficiosas, como reescribir partes del código heredado, desarrollar habilidades de equipo y reducir la rotación de personal.

► 13.5. Caso de estudio: BBVA

En 2014, **BBVA** como producto de un análisis de su situación en el mercado decidió diseñar nuevos esquemas de trabajo, ya que nuevos competidores basados en tecnologías *fintech* estaban penetrando en el sector financiero. Por tanto, **BBVA** decidió implementar técnicas ágiles y crear equipos *scrum* en España, México y en algunos países de Suramérica. Estos equipos se caracterizaban por (Forcano, 2018a):

- Combinar personal del producto e ingeniería formando equipos multidisciplinares.
- Los equipos conformados estaban coubicados, es decir, compartiendo los mismos espacios de trabajo.
- Los equipos estaban dedicados tiempo completo a los proyectos.
- Los equipos laboraban en ciclos de 2 a 3 semanas de duración.

Los resultados se observaron al completar un año de trabajo, al mejorar la calidad del producto desarrollado, tiempos de comercialización más rápidos y un aumento en la productividad de los empleados (con tasas triplicadas de entregas de funciones).

Para proyectos muchos más grandes, se implementaron las técnicas propuestas por el modelo Scaled Agile Framework (SAFe). En total, a finales de 2016, la empresa tenía 3.600 personas implementando proyectos basados en *scrum*.

La aplicación de *scrum* inició con las unidades relacionadas con soluciones para clientes e ingeniería. Sin embargo, el resto de áreas como recursos humanos, finanzas, riesgos, legal, etc., seguían trabajando con los procesos tradicionales, creando fricciones entre los equipos ágiles y no ágiles. Por consiguiente, se decidió continuar con recursos humanos y en 2016 todos los países donde estaba **BBVA** formaron grupos específicos de equipos ágiles en todas las unidades denominadas talento y cultura. En 2017, se observó que tener el área de recursos humanos en un esquema dual (ágil y no ágil) no era lo adecuado. Por tanto, todos los equipos se convirtieron en una unidad u organización ágil. Actualmente, la unidad de talento y cultura cuenta con más de 2000 personas en diez países que trabajan el enfoque ágil (Forcano, 2018a).

La organización de talento y cultura en **BBVA** se basa en cuatro grupos, como se presenta en la tabla 13.5.

Tabla 13.5. Grupos de trabajo establecidos en BBVA

Equipo	Definición	Funciones	Tamaño
Equipos <i>front</i>	Equipo de socios comerciales que ofrece asesoramiento estratégico y apoyo a áreas, gerentes y empleados.	Asesora y entrena a gerentes y apoya a los empleados a lo largo de su ciclo de vida en la organización.	0-15% del equipo del equipo total.
Equipos disciplinarios	Equipos expertos que desarrollan modelos, políticas, herramientas y plataformas para sus áreas.	Aseguran la conectividad de las personas a través de comunidades de práctica globales que comparten mejores prácticas y crean nuevos modelos y plataformas.	Suelen ser sénior pero pequeños (2-3 personas por disciplina), y representan no más del 10-15% del equipo total.

Equipo	Definición	Funciones	Tamaño
Equipo de desarrollo de soluciones	Grupo de profesionales dedicados a ejecutar proyectos o construir soluciones siguiendo los principios de <i>scrum</i> .	Trabajan en <i>sprints</i> de 2 a 3 semanas. Se asignan de forma dinámica y trimestralmente.	Representan al menos el 25-30% del equipo total.
Equipo de experiencia del empleado	Grupo de equipos para ejecutar procesos y entregar valor a los clientes internos utilizando <i>kanban</i> .	Desarrollan tareas con valor significativo rediseñando procesos y aplicando técnicas de automatización, robótica y aprendizaje automático.	Representan el 40-50% del área.

Fuente: elaboración propia con base en Forcano (2018a).

Este nuevo diseño organizativo constituyó la columna vertebral de un nuevo modelo de gobierno que exige nuevos roles y responsabilidades, nuevos eventos de equipo, nuevos modelos de gestión de personas, nuevas herramientas de comunicación, etc.

► 13.6. Caso de estudio: Spotify

Spotify es una compañía de software que proporciona servicios de música en *streaming* fundada en 2006 y con cerca de 2.900 empleados en 2018. Spotify cuenta con seis oficinas de desarrollo e investigación en tres países. Sus formas de trabajo y estructuras organizacionales están diseñadas para promover la innovación, la colaboración y la autonomía (Smite, 2020).

Spotify está organizado en forma de una matriz multidimensional (figura 13.1), con el escuadrón como el principal productor de valor, y los escuadrones agrupados en tribus. Estos tienen la habilidad de implementar el software desarrollado al final de cada *sprint*, cuentan con el apoyo de un propietario del producto y tienen acceso a un entrenador ágil. Una característica curiosa del funcionamiento de Spotify es que cada uno de estos pequeños grupos tiene una parte del producto que es totalmente suyo (Bäcklander, 2019).

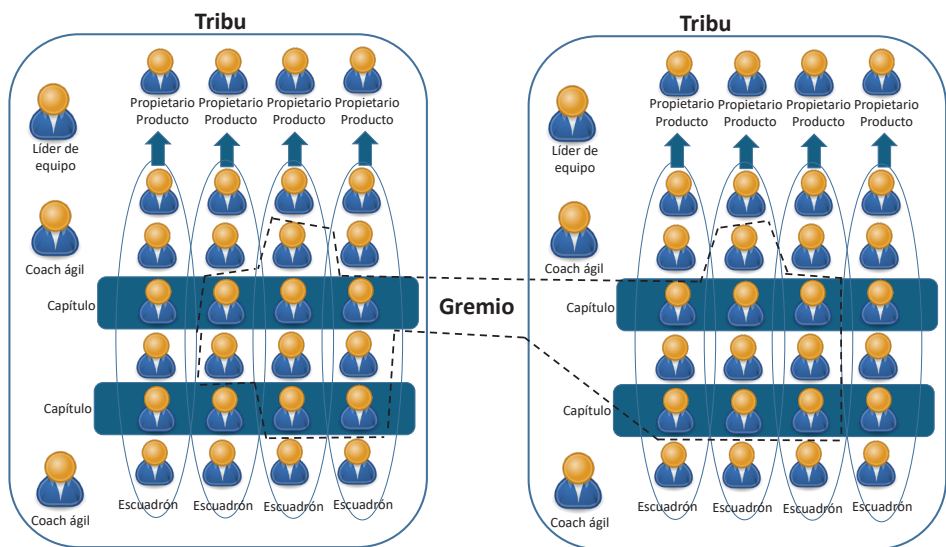


Figura 13.1. Organización matricial del departamento de tecnología de Spotify

Fuente: elaboración propia con base en Bäcklander (2019).

Los miembros de los escuadrones tienen líderes de capítulo, que son sus gerentes de primera línea. Un capítulo es el grupo de empleados dentro de una tribu que son del “mismo tipo”, por ejemplo, programadores de Java. Dado que los escuadrones a menudo son una combinación de competencias, los miembros de un escuadrón tendrán diferentes líderes de capítulo, que pueden estar presentes en ese escuadrón o residir en otro escuadrón dentro de la misma tribu. Un líder de capítulo generalmente divide su tiempo entre ser un “miembro ordinario del equipo” y deberes gerenciales. Tanto en los escuadrones como en las tribus hay comunidades conocidas como gremios, similares a los “grupos de interés” o “comunidades de práctica” a los que cualquiera puede unirse (Bäcklander, 2019). Un resumen de los diferentes tipos de miembros en Spotify se muestra en la tabla 13.6.

Los escuadrones relacionados forman tribus, encabezados por un líder de tribu. Algunos ejemplos de tribus en Spotify son tecnología e infraestructura y operación (que trabajan en sistemas y servicios internos que utilizan el resto de la empresa) y *more than music* (que trabaja en diferentes tipos de colaboraciones con otras empresas).

Tabla 13.6. Tipos de miembros en Spotify

Elemento	Descripción
Escuadrones	Los pequeños equipos <i>scrum</i> en Spotify se denominan escuadrones y deben sentirse como pequeñas empresas emergentes, ser autoorganizados y multifuncionales, e idealmente constar de cinco a siete personas.
Tribus	Todos los escuadrones están organizados en tribus que contienen entre 30 y 200 personas. Las tribus tienen una misión clara, un conjunto de principios, un líder sénior experimentado y las habilidades necesarias para diseñar funciones de software.
Capítulos	Un capítulo es un grupo de ingenieros que tiene el mismo gerente (líder del capítulo) y se centra en el crecimiento personal y el desarrollo de habilidades. Los ingenieros de los capítulos comparten conocimientos, aprenden unos de otros y discuten desafíos comunes.
Gremios	Un gremio es un grupo de personas con habilidades e intereses similares que comparten conocimientos, herramientas o código en Spotify.

Fuente: elaboración propia con base en Bäcklander (2019).

► 13.7. Caso de estudio: NASA

El desarrollo de sistemas aeroespaciales es una tarea compleja, la cual debe incluir estándares y requisitos de seguridad para garantizar la confiabilidad de *hardware* y *software* sofisticados. Las misiones futuras de la National Aeronautics and Space Administration (**NASA**) poseen un enfoque para desarrollar sistemas de naves espaciales basadas en enjambres que comprenden múltiples naves espaciales autoorganizadas y autónomas (Vassev *et al.*, 2012).

Los enfoques inspirados en la biología apuntan a nuevas clases de misiones de exploración espacial que utilizan la inteligencia y cooperación de enjambre para lograr sistemas robustos. Estos sistemas comprenden miles de pequeñas naves espaciales que trabajan juntas para explorar lugares en el espacio profundo donde una sola nave espacial monolítica no es práctica.

Sin embargo, el desarrollo de estos sistemas, desde la conceptualización hasta la validación, es una actividad multidisciplinaria compleja, y la confiabilidad y la seguridad son objetivos clave. Los sistemas no pueden exhibir fallas posteriores al lanzamiento o las fallas que puedan poner en

peligro la misión o causar la pérdida de vidas. Por tanto, requieren un diseño cuidadoso y pruebas exhaustivas para garantizar una confiabilidad adecuada.

Además, los sistemas aeroespaciales tienen requisitos estrictos de confiabilidad y tiempo real, necesitan una reasignación flexible de recursos y deben estar limitados en tamaño, peso y consumo de energía. Por consiguiente, los desarrolladores deben optimizar sus diseños en función de tres factores: rendimiento, confiabilidad y costo. Como resultado, el proceso de desarrollo, caracterizado por numerosas actividades iterativas de diseño y análisis, es largo y costoso. Por otro lado, los sistemas que requieren certificación antes de la operación deben pasar por una rigurosa verificación y validación (Vassev *et al.*, 2012).

Para cumplir estos requerimientos, el proceso de software para tales sistemas debe enfatizar la verificación, validación, certificación y prueba, como se observa en la figura 13.2.

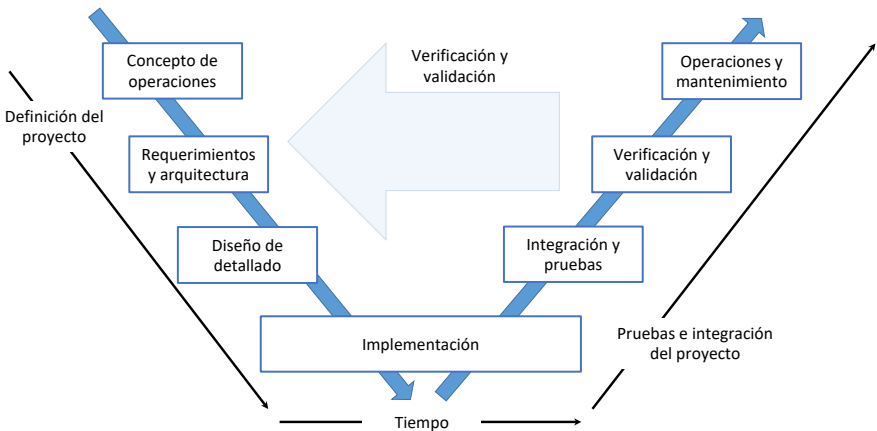


Figura 13.2. Proceso común de desarrollo de software de la NASA

Fuente: elaboración propia con base en Vassev *et al.* (2012).

La mayoría de los proyectos de desarrollo de software aeroespacial modernos utilizan algún tipo de metodología basada en espiral en lugar de un proceso en cascada porque enfatiza mejor la verificación.

El proceso de software debería ayudar a los desarrolladores de software a especificar el nivel de seguridad requerido para garantizar que diseñen e implementen sistemas seguros. Los ingenieros de la NASA pueden

expresar la seguridad del software como un conjunto de características y procedimientos que garantizan un rendimiento predecible del sistema en condiciones normales y anormales. La NASA utiliza dos estándares de seguridad de software (NASA, 1997) que definen cuatro niveles cualitativos de gravedad de los peligros (catastrófico, crítico, marginal e insignificante) y niveles de probabilidad (probable, ocasional, remoto e improbable). La gravedad y la probabilidad del peligro se correlacionan para derivar un índice de riesgo y priorizar la resolución del riesgo (tabla 13.7).

Tabla 13.7. Determinación del índice de riesgo de la NASA

Severidad del riesgo	Probabilidad del riesgo			
	Probable	Ocasional	Remoto	Improbable
Catastrófico	1	1	2	3
Crítico	1	2	4	4
Marginal	2	3	4	5
Negligente	3	4	5	5

Fuente: elaboración propia con base en Vashev *et al.* (2012).

Los sistemas aeroespaciales contemporáneos están diseñados e implementados como sistemas de múltiples componentes, en los que estos son autónomos y reutilizables, lo que requiere una alta independencia y una sincronización compleja. Los componentes de sistemas más sofisticados son considerados sistemas multiagente que incorporan algún grado de inteligencia.

Los agentes inteligentes son clave para realizar sistemas de autogestión. El desarrollo de sistemas aeroespaciales suele implicar sistemas de componentes múltiples; elementos de inteligencia artificial (IA); sistemas autónomos; sistemas en evolución; sistemas de alto riesgo y alto costo; restricciones de diseño rígidas; espacio de diseño extremadamente reducido, y sistemas altamente impulsados por el riesgo.

► 13.8. Caso de estudio: Apple

Apple desarrolla, vende y brinda soporte a una serie de computadores personales, reproductores multimedia portátiles, teléfonos móviles, tele-

visores inteligentes y accesorios de *hardware* informático. Sus productos de *hardware* más conocidos incluyen la línea Macintosh de computadores personales, la línea iPod de reproductores multimedia portátiles y el teléfono iPhone. El producto de software más conocido de Apple es el sistema operativo macOS.

Apple es conocida por los consumidores como una empresa líder en innovación que genera productos de alta calidad. Por otro lado, una empresa para ser ágil por naturaleza debe responder a un cambio repentino en el entorno empresarial no solo de forma rápida, sino también rentable. Para medir el grado de agilidad, se utilizaron tres métricas: cuota de mercado, capacidad de respuesta y rentabilidad (Ganguly *et al.*, 2009).

Para medir la cuota de mercado, se realizó una comparación entre el número de iPods vendidos en el último trimestre en 2003 (733.000) (Apple, 2004) contra el número de iPods vendidos en el tercer trimestre de 2007 (9.815.000) (Apple, 2007). La ecuación resultante fue:

$$AG(Ms) = 9.815.000 / 733.000 = 13,39$$

Lo que indica un gran aumento en la cuota de mercado y, por consiguiente, el comportamiento ágil de Apple con respecto a esta métrica.

Por otro lado, se puede comparar el tiempo de lanzamiento entre la primera y la segunda generación de iPod, y entre la primera y segunda generación de un competidor como Microsoft Zune. Aunque tomó ocho meses (octubre de 2001-julio de 2002) para que Apple transite de la primera a la segunda generación de iPod, a Microsoft le tomó 12 meses (noviembre de 2006-noviembre de 2007). La ecuación que indica la capacidad de respuesta es:

$$Ag(Re) = 12 \text{ meses} / 8 \text{ meses} = 1,5$$

Lo que indica que la tasa de capacidad de respuesta de Apple tenía un ritmo más rápido que su competidor.

Por último, una definición de métricas de costos establece que, para que una empresa se considere ágil, su costo de desarrollar un nuevo producto debe ser menor que el costo de las industrias de desarrollo de nuevos productos. Por ejemplo, el costo de desarrollar el iPod debe ser menor que el costo de desarrollar un reproductor de audio digital portátil del mismo tipo y con la misma especificación. El costo de fabricación de un iPod *shuffle* se estableció en alrededor de US\$45,00, mientras el mismo producto (lanzado por Rio Forge Sport) es de US\$53,00, que es marginalmente mayor. Por tanto, la métrica de rentabilidad es:

$AG(Ce) = US\$53,0 / US\$45,0 = 1,18$

Se concluye que el valor de la métrica de rentabilidad en el caso de Apple en el sector de reproductores de medios digitales es mayor que uno (valor preferido de la métrica). Por tanto, con base en las métricas desarrolladas en la tabla 13.8, se concluyó que Apple mostró suficientes características de agilidad para lograr la posición de líder del mercado en relación con sus reproductores multimedia digitales portátiles.

Tabla 13.8. Comportamiento de Apple en el sector de medios digitales

Métrica	Qué mide	Valor cuantitativo (valor preferido >1)	Apple
Cuota de mercado	• Calidad • Rentabilidad/beneficio	$AG(Ms) = \frac{9.815.000}{733.000} = 13,39$	• Un aumento constante en la cuota de mercado desde el periodo 1 hasta el periodo n • Tasa de crecimiento superior a la de sus competidores
Capacidad de respuesta	• Velocidad	$AG(Re) = \frac{12 \text{ meses}}{8 \text{ meses}} = 1,5$	• Rápida respuesta a las necesidades del consumidor. • Nuevo producto desarrollado y lanzado con intervalos cortos.
Rentabilidad	• Costo	$AG(Ce) = \frac{US\$53,00}{US\$45,00} = 1,18$	• Más barato de fabricar que sus homólogos de la competencia.

Fuente: elaboración propia con base en Ganguly *et al.* (2009).

► 13.9. Caso de estudio: Amazon

El esquema de trabajo en Amazon se caracteriza por un enfoque basado en la generación de valor para los clientes. La obsesión por el cliente no es solo para la alta dirección, las ventas y el *marketing*. Se espera que el personal esté absorto en conocer y mejorar el impacto que tiene todo lo que hacen en el cliente (Denning, 2019).

La obsesión por el cliente en Amazon va más allá de su satisfacción. Significa un impulso para continuar mejorando su experiencia. Estas iniciativas han llevado a la creación de nuevos negocios y nuevos grupos de clientes.

El enfoque basado en el cliente en Amazon está habilitado e impulsado por métricas. Por el contrario, en la mayoría de las grandes organizaciones, los datos de los clientes, donde existen, son irregulares y débiles, mientras los datos financieros son duros y ubicuos. En efecto, Amazon no inicia una actividad ni desarrolla una capacidad a menos que el equipo haya descubierto cómo medirá la respuesta de los clientes en tiempo real. Amazon se basa en las métricas de los clientes desde el principio. Para ello, los equipos pueden pasar semanas analizando estas métricas.

En Amazon, el punto de vista del cliente externo está integrado en las actividades y capacidades desde el principio. No se puede emprender ninguna actividad nueva significativa hasta que haya una revisión exhaustiva por parte de la gerencia de un documento de seis páginas que explique la actividad como una descripción.

La narrativa de seis páginas está respaldada por otro documento conocido como **PR/FAQ**, que contiene un comunicado de prensa futuro imaginando que describe los beneficios que obtienen los clientes y responde a las “preguntas frecuentes” sobre cómo se desarrolló la actividad. También hay un conjunto de métricas mediante las cuales se medirán los beneficios de la actividad para el cliente en tiempo real. Estos documentos son revisados rigurosamente por un grupo de altos directivos antes de comenzar a trabajar en la actividad. Si se aprueba, la actividad se financia y se incorpora al proceso de planificación anual conocido como **OP1**, en el que se evalúan los méritos relativos de cada actividad en términos de su contribución al valor para los clientes (Denning, 2019).

Una vez se aprueba y financia una actividad, comienza el trabajo y los documentos de planificación se actualizan constantemente a medida que se dispone de más información. A través de sus métricas de clientes en tiempo real, cada equipo informa a la alta dirección, no solo a un jefe directo. Al eliminar a los mandos intermedios como actor importante del proceso con sus propias metas y prioridades separadas, Amazon elimina en gran medida la competencia entre unidades. Las métricas que agregan valor a los clientes están disponibles para todos en tiempo real.

Como en los procesos ágiles, el trabajo se realiza en equipos multidisciplinarios autónomos integrados, conocidos en Amazon como “equipos

de dos *pizzas*”, un grupo lo suficientemente pequeño como para que pueda alimentarse con solo dos *pizzas*. Los equipos de Amazon son notablemente independientes y autónomos una vez que se aprueban sus documentos de planificación (Denning, 2019).

► Desafíos

En este capítulo, se han presentado algunos casos de estudio sobre procesos de software. Estos casos son reportes divulgados en algunas publicaciones científicas internacionales o publicados en internet. Sin embargo, encontrar casos de estudio sobre procesos de software de las principales empresas desarrolladoras de software a nivel mundial es difícil. Por un lado, el número de casos de estudio es escaso. Por otro lado, si se encuentran de estas empresas, sus procesos de software se presentan con una descripción general sin profundizar en detalles sobre actividades, roles y artefactos que se producen.

Otro aspecto a resaltar es que los casos de estudio identificados están orientados a describir casos de éxito y se conoce que hay muchas cuestiones de definición, mejora e implementación del proceso de software que presentan fracasos o lecciones aprendidas que no se publican y sería necesario que los investigadores e interesados en procesos de software pudieran conocer para asimilar como lecciones aprendidas y no repetir en proyectos futuros. Por ejemplo, Gregory *et al.* (2015) señalan que los informes sobre la aplicación de prácticas ágiles suelen describir historias de éxito de problemas resueltos y casi nunca dificultades persistentes, situaciones de deterioro o, incluso, fracaso total.

► Ejemplos

En Niazi (2006), se realizó una investigación sobre el impacto de iniciativas de Software Process Improvement (**SPI**) en las capacidades de varias organizaciones desde el punto de vista del costo, el tiempo, la productividad y la satisfacción del cliente.

Con respecto al costo, los resultados de iniciativas de mejora del proceso fueron positivos y muestran que los esfuerzos pueden reducir el costo general de desarrollo:

- Raytheon (Haley, 1996): Antes de implementar Capability Maturity Model (**CMM**), los costos promedio de reelaboración eran cercanos al 41% de los costos del proyecto; después de la iniciativa **CMM**, estos se redujeron cerca del 20%. La solución de problemas de código fuente se redujo en un 80% y el costo de repetir pruebas en un 50%.
- Hughes Aircraft (Humphrey *et al.*, 1991): La mejora del proceso resultó en ahorros cercanos a los US\$2 millones anuales.
- Motorola (Diaz y Sligo, 1997): El retorno de la inversión para las iniciativas **SPI** fue de US\$611.200 para una inversión de US\$90.180 (con un retorno total del 677%).

Los resultados de las investigaciones muestran que el impacto en el tiempo de las iniciativas **SPI** son un enfoque eficaz que puede ayudar a las organizaciones a reducir el tiempo de comercialización.

- Raytheon (Dion, 1993): Antes de la iniciativa **SPI**, la mayoría de los proyectos se completaban con retraso y luego cumplen las fechas de entrega.
- Herbsleb *et al.* (1994) identificaron una reducción anual en el tiempo de comercialización en el rango del 15% al 23%.
- Herbsleb y Goldenson (1996) encontraron una correlación entre una mayor madurez y los horarios de las reuniones. La capacidad para cumplir con el cronograma aumentó del 40% al 80% (en empresas del nivel 1 al nivel 3 **CMM**).

El impacto en la productividad también ha sido destacado en la implementación de iniciativas **SPI**:

- Herbsleb y Goldenson (1996) mostraron que la productividad de desarrollo de las empresas aumentó del 50% al 90% al pasar del nivel 1 al nivel 3 de **CMM**.
- Herbsleb *et al.* (1994) identificaron una ganancia anual en productividad en el rango del 6% al 67%.
- Raytheon (Haley, 1996): La iniciativa **SPI** obtuvo un aumento promedio de la productividad cercano al 130%.
- Butler (1995) describió un proyecto de desarrollo 10 veces más productivo que el proyecto de referencia.

Hay varios estudios que describen el impacto del **SPI** en la satisfacción del cliente:

- ▶ Yamamura (1999) muestra una correlación entre la satisfacción de los empleados y la madurez del proceso. El promedio de satisfacción se incrementó de un 57% a un 83% (antes y después de las actividades **SPI**).
- ▶ Herbsleb y Goldenson (1996) demostraron que la satisfacción del cliente aumentó del 80% al 100% al pasar del nivel 1 al nivel 3 de **CMM**.
- ▶ Telcordia Technologies (Pitterman, 2000): Informa de un aumento de la satisfacción del cliente del 60% al 95% (de 1992 a 2000).

▶ Resumen

Este capítulo presentó diversos casos de éxito de implementación de nuevos procesos de software en reconocidas organizaciones desarrolladoras de software a nivel mundial.

El proyecto Sentinel del **FBI** tenía como objetivo digitalizar los miles de casos de la agencia. El proyecto inició con la aplicación del proceso en cascada, pero tuvo excesivos retrasos en costo y tiempo. El **FBI** decidió implementar *scrum* para superar los problemas presentados y pudo completar con éxito el proyecto.

Un proyecto en Intel describe cómo fue la adaptación de diferentes prácticas de *scrum* al proyecto y la realización de un escalamiento del proceso, el cual se dividió en las fases de preparación, supervivencia y preparación de la fabricación. Intel se benefició del cambio al proceso ágil con un tiempo de ciclo reducido, desempeño según el cronograma, moral mejorada y mayor transparencia.

El proyecto AdWorks de Google tenía varios problemas debido a su complejidad de gestión. Para ello, se decidió implementar algunas técnicas ágiles como un *backlog* de lanzamientos, *burndown charts* y una reunión semanal de punto de control.

En la **BBC**, se implementó un proceso *lean* para gestionar el desarrollo de software de su departamento de webmedia. El equipo analizó el flujo de trabajo utilizando gráficos de control del proceso. Los cambios implantados evidenciaron que durante un periodo de 12 meses el tiempo de entrega mejoró en un 37%, la consistencia de la entrega aumentó en un 47% y los defectos se redujeron en un 24%.

En **BBVA**, se decidió implementar técnicas ágiles y crear equipos *scrum*. Se inició con las unidades relacionadas con soluciones para clientes e ingeniería. Sin embargo, luego la unidad de recursos humanos implementó *scrum* en todos los países donde contaban con oficinas de desarrollo.

Spotify ha adoptado una organización ágil compuesta de escuadrones y los escuadrones en tribus. A su vez, los capítulos son grupos transversales de diferentes escuadrones, pero del mismo tipo.

Los sistemas aeroespaciales de la **NASA** basados en enjambres deben optimizar sus diseños en función del rendimiento, la confiabilidad y el costo. Para cumplir con estos requerimientos, el proceso de software para tales sistemas debe enfatizar en etapas como la verificación, validación, certificación y prueba.

Para medir el grado de agilidad en Apple, se utilizaron tres métricas: cuota de mercado, capacidad de respuesta y rentabilidad, las cuales dieron resultados positivos con respecto a otros productos de la competencia.

En Amazon, sus equipos de desarrollo para iniciar una actividad deben especificar por medio de métricas cómo responderán los clientes en tiempo real. Estos informes son presentados a la gerencia, eliminando mandos intermedios. Estas métricas estarán disponibles para todos en tiempo real.

► Ejercicios

1. Buscar en internet otros casos de procesos de software de empresas del sector de software reconocidas a nivel mundial.
2. Visitar una organización de software de su ciudad y consultar sobre los procesos de software que utiliza. ¿Es un proceso tradicional o ágil? ¿La empresa tiene algún certificado de calidad de sus procesos de software?

► Referencias

Apple. (2004, 14 de enero). *Apple Reports First Quarter Results*. <https://www.apple.com/newsroom/2004/01/14Apple-Reports-First-Quarter-Results/#:~:text=CUPERTINO%2C%20California%E2%80%94January%2014%2C,of%20%248%20million%2C%20or%20%24>.

- Apple. (2007, 17 de enero). *Apple reports third quarter results*. <https://www.apple.com/newsroom/2007/01/17Apple-Reports-First-Quarter-Results/#:~:text=CUPERTINO%2C%20California%E2%80%94January%2017%2C,or%20%241.14%20per%20diluted%20share>.
- Bäcklander, G. (2019). Doing complexity leadership theory: How agile coaches at Spotify practise enabling leadership. *Creativity and Innovation Management*, 28(1), 42-60. <https://doi.org/10.1111/caim.12303>
- Butler, K. (1995). The economic benefits of software process improvement. *Crosstalk*, 8(7), 14-17.
- Denning, S. (2019). How Amazon practices the three laws of agile management. *Strategy & Leadership*, 47(5), 36-41. <https://doi.org/10.1108/SL-07-2019-0104>
- Diaz, M. y Sligo, J. (1997). How software process improvement helped Motorola. *IEEE software*, 14(5), 75-81. <https://doi.org/10.1109/52.605934>
- Dion, R. (1993). Process improvement and the corporate balance sheet. *IEEE Software*, 10(4), 28-35. <https://doi.org/10.1109/52.219618>
- Elwer, P. (2008). *Agile Project Development at Intel: A scrum Odyssey*. <http://www.michael-james.org/Intel-case-study.pdf>
- Forcano, R. (2018a, 14 de junio). *HR goes Agile: A case study in BBVA*. <https://www.linkedin.com/pulse/hr-goes-agile-case-study-bbva-ricardo-forcano>
- Forcano, R. (2018b, 16 de julio). *RRHH se transforma a 'agile': Un caso de estudio en BBVA*. <https://www.bbva.com/es/opinion/rrhh-transforma-agile-caso-estudio-bbva/>
- Ganguly, A., Nilchiani, R. y Farr, J. V. (2009). Evaluating agility in corporate enterprises. *International Journal of Production Economics*, 118(2), 410-423. <https://doi.org/10.1016/j.ijpe.2008.12.009>
- Garzás, J. y Paulk, M. C. (2013). A case study of software process improvement with CMMI-DEV and scrum in Spanish companies. *Journal of Software: Evolution and Process*, 25(12), 1325-1333. <https://doi.org/10.1002/smr.1605>
- Gregory, P., Barroca, L., Taylor, K., Salah, D. y Sharp, H. (2015). Agile challenges in practice: A thematic analysis. En C. Lassenius, T. Dingsøyr y M. Paasivaara (eds.), *Agile Processes in Software Engineering and Extreme Programming. XP 2015. Lecture Notes in Business Information Processing* (vol. 212, pp. 64-80). Springer. https://doi.org/10.1007/978-3-319-18612-2_6
- Haley, T. J. (1996). Software process improvement at Raytheon. *IEEE Software*, 13(6), 33-41. <https://doi.org/10.1109/52.542292>
- Herbsleb, J. D. y Goldenson, D. R. (1996). A systematic survey of CMM experience and results. En *Proceedings of IEEE 18th International Conference on Software Engineering* (pp. 323-330). IEEE. <https://doi.org/10.1109/ICSE.1996.493427>
- Herbsleb, J., Carleton, A., Rozum, J., Siegel, J. y Zubrow, D. (1994). *Benefits of CMM-based software process improvement: Executive summary of initial results*. Carnegie Mellon University. https://insights.sei.cmu.edu/documents/1112/1994_005_001_16310.pdf
- Humphrey, W. S., Snyder, T. R. y Willis, R. R. (1991). Software process improvement at Hughes Aircraft. *IEEE Software*, 8(4), 11-23. <https://doi.org/10.1109/52.300031>

- Middleton, P. y Joyce, D. (2011). Lean software management: BBC worldwide case study. *IEEE Transactions on Engineering Management*, 59(1), 20-32. <https://doi.org/10.1109/TEM.2010.2081675>
- NASA. (1997). *Software Safety*. NASA Technical Standard **NASA-STD-8719.13A**.
- Niazi, M. (2006). Software process improvement: A road to success. En J. Münch y M. Vierimaa (eds.), *Product-Focused Software Process Improvement*. **PROFES 2006. Lecture Notes in Computer Science** (vol. 4034, pp. 395-401). Springer. https://doi.org/10.1007/11767718_34
- Olszewski, L. y Wingreen, S. C. (2011). The **FBI** sentinel project. *Journal of Cases on Information Technology (JCIT)*, 13(3), 84-102. <https://doi.org/10.4018/jcit.2011070105>
- Pitterman, B. (2000). Telcordia technologies: The journey to high maturity. *IEEE Software*, 17(4), 89-96. <https://doi.org/10.1109/52.854074>
- Smite, D., Moe, N. B., Floryan, M., Levinta, G. y Chatzipetrou, P. (2020). Spotify guilds. *Communications of the ACM*, 63(3), 56-61. <http://dx.doi.org/10.1145/3343146>
- Standish Group. (2020). *CHAOS report: Beyond infinity*. <https://standishgroup.myshopify.com/>
- Striebeck, M. (2006). Ssh! We are adding a process... [agile practices]. En **AGILE 2006 (AGILE'06)** (pp. 9-193). **IEEE**. <https://doi.org/10.1109/AGILE.2006.48>
- Vashev, E., Sterritt, R., Rouff, C. y Hinchey, M. (2012). Swarm technology at NASA: Building resilient systems. *IT Professional*, 14(2), 36-42. <https://doi.org/10.1109/MITP.2012.18>
- Yamamura, G. (1999). Software process satisfied employees. *IEEE Software*, 16(5), 83-85.
- Zelkowitz, M. V. (2009). An update to experimental models for validating computer technology. *Journal of Systems and Software*, 82(3), 373-376. <https://doi.org/10.1016/j.jss.2008.06.040>
- Zelkowitz, M. V. y Wallace, D. R. (1998). Experimental models for validating technology. *Computer*, 31(5), 23-31. <https://doi.org/10.1109/2.675630>

Tendencias actuales de investigación del proceso de software

► Objetivos

Los objetivos de este capítulo están orientados a conocer las principales tendencias de investigación en el área de la ingeniería de procesos de software. Al finalizar este capítulo, el lector estará en capacidad de:

- Conocer diferentes tendencias de investigación en procesos de software propuestas por el marco de trabajo de investigación de procesos propuesto por el International Process Research Consortium (**IPRC**).
- Entender en qué consiste la minería de procesos de software.
- Obtener un panorama global de las técnicas de aprendizaje automático y aprendizaje profundo y sus implicaciones al aplicarlas al dominio del proceso de software.
- Comprender nuevos modelos de desarrollo de software que han surgido debido a cambios en los aspectos económicos y tecnológicos a nivel mundial.

La investigación en procesos de software se ocupa de los métodos y las tecnologías utilizados para evaluar, respaldar y mejorar las actividades de desarrollo de software. El campo ha crecido desde la década de 1980 para abordar la creciente complejidad y criticidad de las actividades de desarrollo de software

(Fuggetta, 2000). Afrontar los problemas y las cuestiones del desarrollo de software no es solo una cuestión de introducir herramientas y entornos de desarrollo efectivos, sino que es necesario profundizar en nuevos conceptos, líneas de investigación, tecnologías emergentes y campos interdisciplinarios que pueden contribuir a mejorar las prácticas existentes.

La investigación en procesos de software tiene un papel importante que desempeñar en el futuro de la investigación y práctica de la ingeniería de software. En general, los investigadores y profesionales en procesos de software deben reutilizar las experiencias y los logros de otras áreas y disciplinas.

A continuación, se presenta un compendio de las principales tendencias de investigación en esta área basadas en directrices internacionales. Luego, se abordan temas concretos de investigación relevantes, como la minería del proceso de software, el aprendizaje automático y el aprendizaje profundo. El capítulo finaliza con una descripción de nuevos modelos de desarrollo de software.

► 14.1. Tendencias de investigación en procesos de software

Un documento que establece un programa bastante formalizado para realizar investigaciones en el área del conocimiento del proceso de software es el propuesto por el International Process Research Consortium (**IPRC**). El **IPRC** fue creado en 2004 por el Software Engineering Institute (**SEI**) para explorar diferentes direcciones de investigación estratégica en procesos de software. El **IPRC** definió un marco de trabajo de investigación de procesos (*process research framework*) para formular iniciativas de alta prioridad en procesos (Forrester, 2006).

El contexto del marco de trabajo desarrollado se circunscribe a problemas que surgen de cuatro áreas fundamentales, las 4P de la ingeniería de software:

- *Productos*: Aprovechan la tecnología informática y tienen atributos de calidad internos y externos.
- *Proyectos*: Involucran diferentes estructuras organizacionales y tienen un determinado alcance.
- *Personas*: Tienen motivaciones, habilidades y cultura para desarrollar sus actividades.

- *Procesos*: Proporcionan el “pegamento” para mantener unidas todas las áreas fundamentales.

El marco de trabajo abarca las principales corrientes de investigación y cuatro temas de investigación: relaciones entre calidad del proceso y producto, ingeniería de procesos, gestión de procesos de proyectos y despliegue del proceso. Además, incluye un conjunto de factores tecnológicos en el área de tendencias y desafíos tecnológicos emergentes. Cada tema tiene asociado un conjunto de nodos de investigación y cada nodo incluye una serie de preguntas de investigación. En total, los temas comprenden 20 nodos de investigación que abordan más de 150 preguntas de investigación. En la figura 14.1, se presenta una visión general del marco de trabajo **IPRC** con los diferentes temas y nodos de investigación.

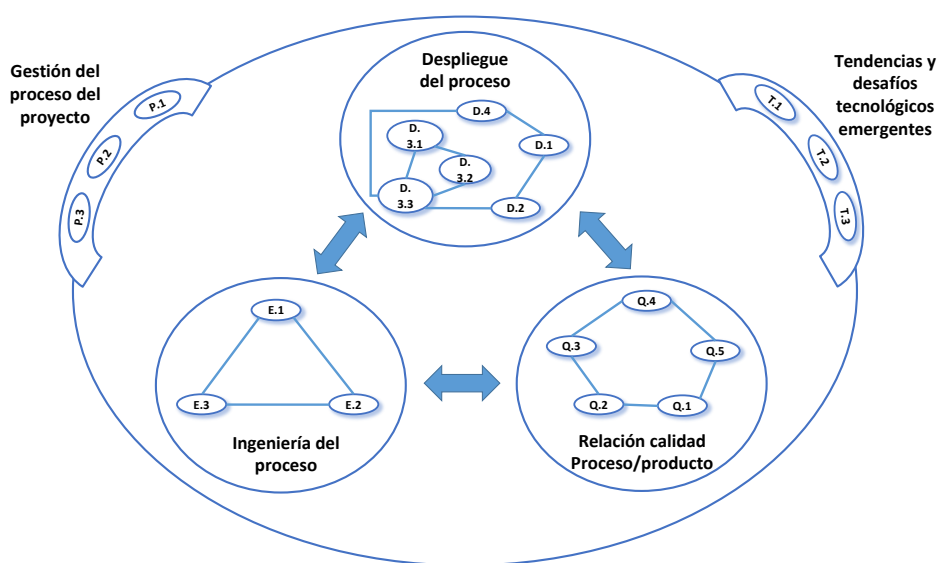


Figura 14.1. Arquitectura de los temas de investigación **IPRC**

Fuente: elaboración propia con base en Forrester (2006).

En la tabla 14.1, se presentan las descripciones de los cuatro temas de investigación en procesos de software y los factores tecnológicos de las tendencias emergentes.

Tabla 14.1. Descripción de los temas de investigación IPRC

Tema de investigación	Descripción
Las relaciones entre calidad del proceso y producto	Hace énfasis en la perspectiva del producto. Se interesa por comprender si las características particulares del proceso pueden afectar las cualidades deseadas del producto (y servicio).
Ingeniería de procesos	Hace énfasis en la perspectiva del proceso. Estudia cómo definir y construir procesos y comprender su desempeño.
Gestión de procesos de proyectos	Hace énfasis en la perspectiva organizacional del proyecto. Considera los valores de las partes interesadas que impulsan las estructuras organizacionales utilizadas para emprender el trabajo del proyecto.
Despliegue del proceso	Hace énfasis en la perspectiva de las personas. Estudia la implementación de procesos en estructuras organizacionales adecuadas para que las personas estén capacitadas de manera más eficiente al satisfacer las necesidades del negocio.
Tendencias y desafíos tecnológicos emergentes	Identifica los factores tecnológicos que tienen un impacto significativo en los procesos dentro de un ciclo de vida del sistema.

Fuente: elaboración propia con base en Forrester (2006).

La tabla 14.2 muestra los temas de investigación del marco de trabajo y sus nodos asociados. Las fuerzas conductoras del marco de trabajo se enmarcan en el valor agregado, la diversificación del negocio, el cambio tecnológico, la complejidad del sistema, la calidad del producto, la rotación de productos, la seguridad y protección, las regulaciones y la globalización.

Tabla 14.2. Descripción de nodos de investigación IPRC

Tema	Nodo de investigación	Descripción	Preguntas de investigación
Relaciones entre calidad del proceso y producto	Q.1 Obtención y especificación de requisitos de calidad del producto	Determinar la mejor manera de obtener requisitos de calidad para un producto y cómo especificarlos adecuadamente.	Q1-Q5
	Q.2 Establecimiento de relaciones proceso-producto	Establecer si existe una relación directa entre la calidad de un producto dado y los procesos utilizados para desarrollarlo.	Q6-Q9
	Q.3 Modelado de las relaciones de calidad proceso-producto	Permitir a los gerentes seleccionar procesos que generen una calidad de producto deseada en el nivel y grado requeridos dado el nivel de madurez de la organización.	Q10-Q18
	Q.4 Verificación y validación de las cualidades del producto	Permitir a los gerentes seleccionar procesos de verificación y validación apropiados para confirmar el cumplimiento de los requisitos de calidad.	Q19-Q23
	Q.5 Mantenimiento de las cualidades del producto	Permitir a los gerentes seleccionar los procesos que faciliten el establecimiento, mantenimiento y evolución de la calidad del producto durante su ciclo de vida.	Q24-Q26
Ingeniería de procesos	E.1 Especificación de procesos usando evidencias	Proporcionar las habilidades para especificar, seleccionar y evaluar procesos basados en evidencia.	E1-E21
	E.2 Procesos de la organización para la reutilización	Diseñar procesos a partir de componentes de proceso reutilizables para cumplir con los requisitos específicos del proceso.	E22-E43
	E.3 Suministro de infraestructura de ingeniería de procesos	Proporcionar una infraestructura para soportar la ingeniería de procesos.	E44-E60

Tema	Nodo de investigación	Descripción	Preguntas de investigación
Gestión de procesos de proyecto	P.1 Operación bajo control autónomo	Enfocarse en proyectos con control autónomo para lograr resultados predecibles en términos de funcionalidad, costo, cronograma y atributos de calidad.	P1-P16
	P.2 Gestión a través de la cooperación centralizada	Centrarse en proyectos con una autoridad de decisión principal, pero utilizando diferentes instancias de proceso de proyecto.	P17-P27
	P.3 Acuerdo bajo colaboración federada	Enfocarse en proyectos con autoridad de decisión y objetivos del proyecto compartidos pero con diferentes objetivos y procesos organizacionales.	P28-P32
Despliegue del proceso	D.1 Establecimiento de la infraestructura para el despliegue	Establecer los mecanismos para buscar posibles mejoras de procesos y modelos de mejora de procesos.	D1-D6
	D.2 Motivación del uso del proceso	Estudiar cómo motivar la adopción del proceso desde el punto de vista organizacional e individual (psicológico).	D7-D19
	D.3 Adopción y despliegue efectivos	Proporcionar estructura y conocimiento sobre actividades efectivas de implementación de procesos.	D20-D450
	D.4 Gestión del despliegue de procesos en curso	Garantizar que el despliegue continuo de procesos en la organización pueda ser monitoreada y controlada de manera efectiva.	D46-D60
Tendencias y desafíos tecnológicos emergentes	T.1 Evolución continua de requisitos	Iniciar, administrar y desarrollar conjuntos de requisitos incompletos en múltiples iteraciones de evolución y reconfiguración del sistema.	T1-T23
	T.2 Conocimiento incompleto	Definir niveles aceptables de incompletitud o incertidumbre para desarrolladores en diferentes contextos del sistema.	T24-T37
	T.3 Integración de sistemas basada en componentes homogéneos	Establecer procesos efectivos para concebir, orquestar, construir, desplegar y evolucionar sistemas complejos compuestos de componentes heterogéneos.	T38-T56

Fuente: elaboración propia con base en Forrester (2006).

► 14.2. Minería del proceso de software

La minería de procesos de software se refiere a revelar el modelo de proceso automáticamente utilizando datos del proceso de software para ayudar a identificar, comprender, analizar y optimizar el desarrollo de software y finalmente mejorar la calidad de los productos de software (Caldeira y Abreu, 2017; Dong *et al.*, 2018; Rui *et al.*, 2018).

Los tres tipos principales de uso de la minería de procesos son (Van der Aalst, 2011):

- *Descubrimiento*: Descubre o identifica el proceso como realmente se ejecutó a partir de los datos de eventos disponibles. Es bastante útil para determinar el flujo de control del proceso y la estructura organizacional que subyace a este.
- *Conformidad*: Compara el proceso ejecutado con el proceso definido como una forma de aseguramiento de la calidad. Evalúa si el proceso fue ejecutado realmente por los roles estipulados, si los productos de trabajo definidos fueron los realmente producidos, si el orden de ejecución de las actividades se cumplió, etc.
- *Mejora*: Utiliza la información derivada de los datos para mejorar el proceso, por ejemplo, para reducir cuellos de botella, analizar las reglas de negocio, adaptar el proceso, etc.

En la figura 14.2, se presenta una visión general de la minería de procesos de software en la que a partir de un repositorio de software que contiene registros de eventos del proceso se generan modelos de procesos (descubrimiento), se evalúa su cumplimiento (conformidad) y se plantean estrategias de mejora.

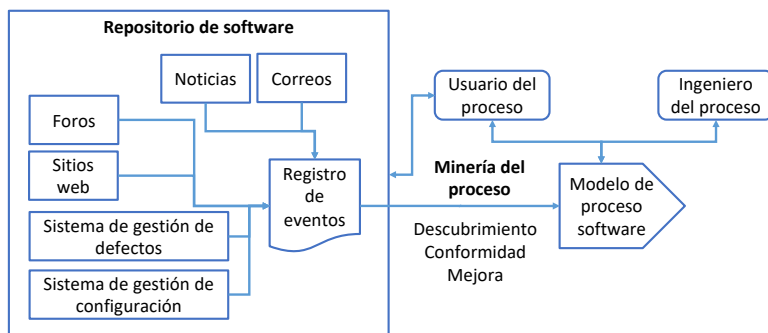


Figura 14.2. Minería de procesos de software

Fuente: elaboración propia con base en Rubin *et al.* (2007).

La base de la minería de procesos son los registros de eventos que contienen información sobre los pasos individuales realizados. Un registro de eventos del proceso de software es una tabla bidimensional que se utiliza para describir la información en el registro de eventos. El concepto de registros de eventos se basa en los siguientes supuestos (Blum, 2016):

- La ejecución de un proceso consta de casos o instancias individuales: Los casos o instancias pueden ser los proyectos específicos.
- Un caso consta de diferentes eventos en los que cada evento pertenece exactamente a un caso. Por ejemplo, un evento puede ser la finalización de la etapa de análisis. Un evento siempre ocurre en un punto en el tiempo no durante un periodo.
- Los eventos que pertenecen a un caso se ordenan. Este orden se utiliza para analizar el flujo de control del proceso. La secuencia de eventos registrados en un caso se denomina traza.

Un registro de eventos debe contener al menos un caso relevante de cada evento, una actividad del proceso realizada y el orden de los eventos, como se presenta en el ejemplo de la tabla 14.3.

Tabla 14.3. Ejemplo de registro de eventos de un proyecto

Identificador del proyecto	Identificador de evento	Marca de tiempo	Actividad	Recurso	Costo (en dólares)
405	271	25-4-2020 8:00 a. m.	Reunión inicial	Persona A	3.000
405	272	25-4-2020 8:00 a. m.	Análisis de requisitos	Persona B	5.000
303	421	25-4-2020 10:00 a. m.	Reunión inicial	Persona A	2.000

Fuente: elaboración propia con base en Blum (2016).

La figura 14.3 muestra una selección de posibles aplicaciones de la minería de procesos en el contexto de las etapas del desarrollo de software. El ciclo de desarrollo presentado es genérico y las disciplinas de ingeniería de software están basadas en el proceso unificado.

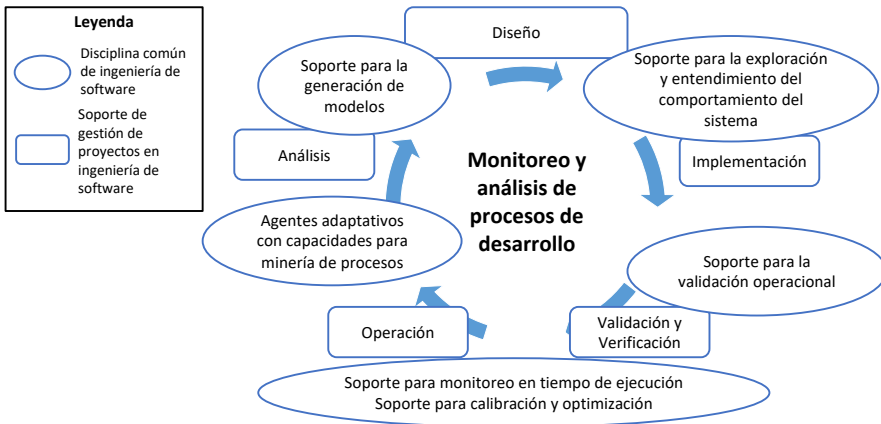


Figura 14.3. Visión general de las actividades de minería de procesos en el desarrollo de software

Fuente: elaboración propia con base en Cabac y Denz (2008).

La minería de procesos de software como área relativamente joven presenta una serie de desafíos por investigar y resolver. Los principales desafíos del uso de la minería de procesos en los procesos de software son (Kneuper, 2018):

- ▶ Los procesos generalmente no están bien estructurados, con una variación considerable entre diferentes instancias del mismo proceso.
- ▶ El número de repeticiones del mismo proceso suele ser bastante bajo. Aunque hay fases de un proceso que se repiten con bastante frecuencia (corrección de errores, gestión de requisitos individuales, etc.) que permiten que la minería de procesos se pueda aplicar para verificar la conformidad con las definiciones del proceso o la detección de cuellos de botella que ralentizan el proceso.

▶ 14.3. Aprendizaje automático y aprendizaje profundo

Recientemente, se ha manifestado un creciente interés de la comunidad tecnológica por la aplicación de la minería de datos en diversas áreas. Las técnicas de minería más ampliamente utilizadas actualmente y con una gran proyección son el aprendizaje automático (*machine learning*) y entre estas últimas se encuentra el aprendizaje profundo (*deep learning*).

14.3.1. Aprendizaje automático

El aprendizaje automático es un conjunto de métodos que pueden detectar automáticamente patrones en los datos y luego utilizar esos patrones descubiertos para predecir datos futuros o realizar otros tipos de toma de decisiones bajo incertidumbre (Murphy, 2011). El aprendizaje significa que el conocimiento novedoso se genera a partir de observaciones y que este conocimiento se utiliza para lograr objetivos definidos.

Los métodos de aprendizaje automático generalmente se dividen en tres tipos principales, como se observa en la figura 14.4 (Marshall, 2015):

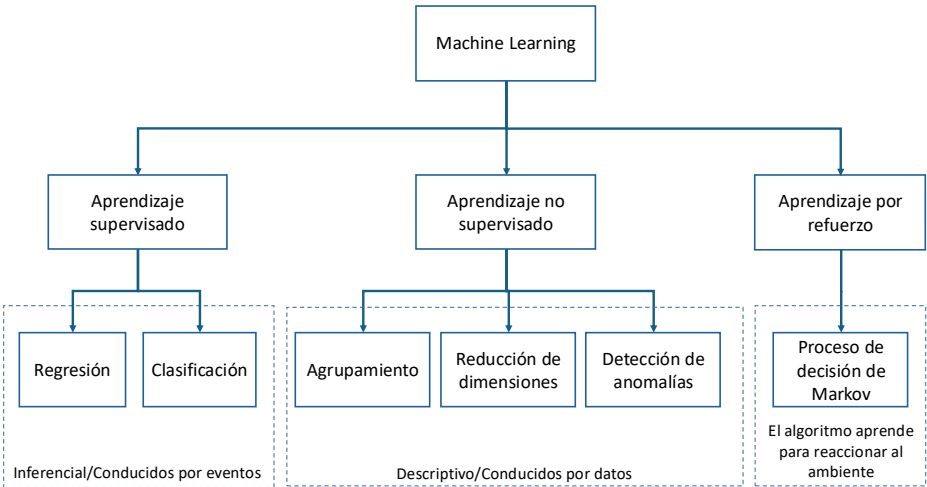


Figura 14.4. Tipos de aprendizaje automático

Fuente: elaboración propia con base en Swamynathan (2017).

- *Supervisado*: Su objetivo es aprender un mapeo entre las entradas y las salidas dado un conjunto etiquetado de pares entrada-salida. Incluye técnicas como:
 - *Regresión*: El resultado a predecir es un número continuo en relevancia con un conjunto dado de datos de entrada.
 - *Clasificación*: La salida a predecir es la probabilidad real o la de un evento/clase. El número de clases a predecir puede ser dos o más. Incluye técnicas más específicas como árboles de decisión, *naïves bayes*, máquinas de vectores de soporte (Support

Vector Machines [**svm**]), reglas de asociación, redes bayesianas, etc.

- ▶ *No supervisado*: Su objetivo es encontrar automáticamente patrones de datos que inicialmente no son aparentes. Este proceso a veces se denomina descubrimiento de conocimiento. Incluye técnicas como:
 - *Agrupamiento*: Asume que los grupos o clases no se conocen de antemano para un conjunto dado de datos. Su objetivo es dividir el conjunto de datos de entrada en grupos lógicos de elementos relacionados. Incluye técnicas específicas como *k-means*, mapas autoorganizados, etc.
 - *Reducción de dimensiones*: Su objetivo es simplificar un gran conjunto de datos de entrada asignándolos a un espacio dimensional más bajo.
 - *Detección de anomalías*: Detecta valores atípicos identificando elementos, eventos u observaciones que no se ajustan a un patrón o comportamiento esperado en comparación con otros elementos en un conjunto dado de datos.
- ▶ *Aprendizaje reforzado*: Consiste en que un agente aprende a comportarse en un escenario desconocido a través de las señales de recompensa y castigo proporcionadas por un crítico. Incluye técnicas específicas, como procesos de decisión de Markov, *Q-learning*, métodos de diferencia temporal y métodos de Montecarlo.

Las técnicas de aprendizaje automático se utilizan como una forma de descubrir información valiosa y procesable y conocimiento dirigido a diferentes fases del proceso de software: errores reportados, defectos de software, predicciones de esfuerzo, patrones de cambio de clases y detección de código clonado (Godfrey *et al.*, 2009).

Otros resultados de la aplicación de técnicas de aprendizaje automático al proceso de software son (Fernández del Carpio y Bermón Angarita, 2018):

- ▶ Las técnicas más utilizadas son las redes neuronales artificiales, las reglas de asociación y las redes bayesianas. Además, las fases del proceso de software en las que se aplican más estas técnicas son la estimación del esfuerzo del proyecto y las pruebas de software.
- ▶ Los principales problemas con la utilización de estas técnicas están relacionados con la dependencia de los conjuntos de datos utili-

zados, la definición de parámetros, la combinación de técnicas, la selección de datos de entrenamiento y la construcción de modelos especializados.

- La aplicación de estas técnicas muestra amplias mejoras del proceso de software en aspectos tales como asignación de recursos humanos, identificación de riesgos, toma de decisiones, predicción de esfuerzo, productividad y estimación de costos, localización de fallas, asignación, detección y categorización de errores, informes de problemas, predicción y evaluación de la gestión del cambio y gestión de la deuda técnica.

14.3.2. Aprendizaje profundo

El aprendizaje profundo o redes neuronales profundas (Deep Neural Network [DNN]) es una rama del aprendizaje automático basada en redes neuronales artificiales. Estas consisten en unidades de procesamiento organizadas en capas de entrada, ocultas y de salida. Los nodos o unidades en cada capa están conectados a nodos en capas adyacentes. Cada conexión tiene un valor numérico (peso). Las entradas se multiplican por los pesos respectivos y se suman en cada unidad. Luego, la suma sufre una transformación basada en una función de activación, que en la mayoría de los casos es una función sigmoidea, una unidad lineal hiperbólica o rectificada (Shrestha y Mahmood, 2019). En la figura 14.5, se presenta un ejemplo de red neuronal profunda.

Una DNN es un tipo de red neuronal modelada como un perceptrón multicapa que está entrenada con algoritmos para aprender representaciones de conjuntos de datos sin ningún diseño manual de extractores de características. El término *deep* sugiere un número mayor o más profundo de capas de procesamiento. Comprende múltiples capas ocultas entre las capas de entrada y salida con neuronas para la transformación y extracción de características (Shrestha y Mahmood, 2019). La estructura de la red se puede personalizar mediante la configuración de ciertos atributos llamados hiperparámetros, es decir, tamaño del lote, número de capas ocultas, número de unidades por capa, épocas, etc. (Falcini *et al.*, 2017). La red se puede entrenar con los diferentes métodos de aprendizaje: supervisado, semisupervisado o no supervisado.

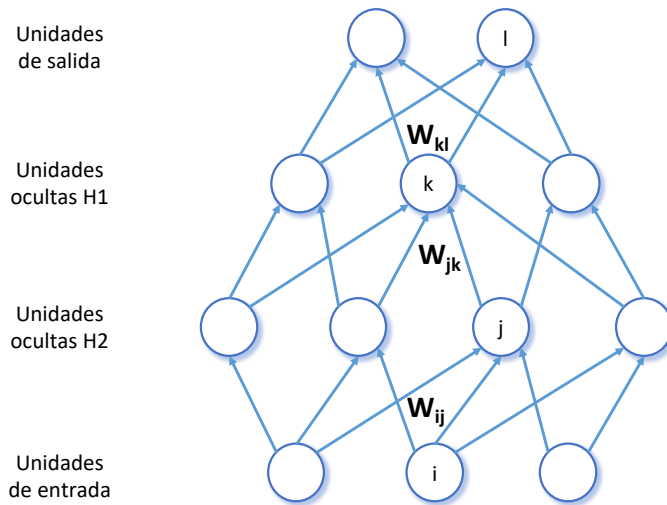


Figura 14.5. Ejemplo de red neuronal

Fuente: elaboración propia con base en LeCun *et al.* (2015).

Las arquitecturas **DNN** se pueden clasificar en los siguientes tipos:

- ▶ *Redes neuronales convolucionales (Convolutional Neural Network [CNN])*, o *ConvNets*: Se caracterizan por capas para caracterizar representaciones. Se compone de capas convolucionales agrupadas y totalmente conectadas. Estas redes han ganado popularidad para procesar imágenes como matrices (Fakhoury *et al.*, 2018).
- ▶ *Autoencoder (AE)*: Comprende las partes de un codificador y un decodificador. Utiliza un algoritmo no supervisado para aprender representaciones de un conjunto de datos que se utiliza para reducir la dimensionalidad y recrear el conjunto de datos original (Ng *et al.*, 2013).
- ▶ *Redes neuronales recurrentes (Recurrent Neural Network [RNN])*: Su estructura presenta una circulación direccional y utiliza una memoria interna para manejar registros históricos más grandes. Se utiliza principalmente para cualquier forma de datos secuenciales. Se denomina recurrente debido al uso de la salida de cálculos anteriores para el cálculo actual de cada elemento de una secuencia (Guo *et al.*, 2017). Entre las redes **RNN** se encuentran la memoria a corto plazo (Long Short-Term Memory [LSTM]) y la unidad recurrente cerrada (Gated Recurrent Unit [GRU]).

- *Redes adversarias generativas (Generative Adversarial Networks [GAN]):* Se componen de dos subredes: el generador y el discriminador. Se aplican principalmente en la generación y resolución de imágenes (Li *et al.*, 2019).
- *Red de creencias profundas (Deep Belief Network [DBN]):* Es un modelo gráfico generativo que comprende capas ocultas y variables latentes. Una característica de esta clase de **DNN** son las conexiones solo entre las capas y no entre los nodos (Akhtar y Mian, 2018).

Las **DNN** pueden aplicarse en muchos campos del conocimiento. En el dominio de los procesos de software, han tenido éxito resolviendo muchos problemas orientados a diversas fases del proceso como (Fernández del Carpio y Bermón Angarita, 2020):

- Resolver ciertos tipos de problemas de clasificación, sistemas de recomendación, generación de secuencias, predicción, recuperación/extracción de información, balanceo de datos, reducción de dimensionalidad y modelado.
- Se pueden aplicar a diversas fases del proceso de software; sin embargo, las fases con más aplicaciones son las pruebas y el mantenimiento de software. En esas fases, las técnicas de aprendizaje profundo, como **CNN**, **RNN** y **LSTM**, se utilizan ampliamente para procesar informes de errores, clasificación de *malware*, gestión de bibliotecas de código, generar recomendaciones y tareas relacionadas con defectos de software (predicción de defectos, identificación de fallas, etc.).
- Las principales contribuciones de las técnicas de aprendizaje profundo están orientadas a la estimación del esfuerzo, clasificación de requisitos software, identificación de elementos visuales en **GUI**, identificación de autores de códigos, identificación de similitud de códigos fuente, predicción y clasificación de defectos y procesamiento de informes de errores en los procesos de prueba y mantenimiento.

► 14.4. Nuevos modelos de procesos de software

Las situaciones actuales a nivel económico, social y tecnológico han permitido que surjan nuevas propuestas de procesos de software orientadas a cubrir las necesidades de tales situaciones. En la figura 14.6, se ilustran

estos nuevos modelos orientados a las nuevas necesidades originadas por la globalización económica, la consolidación de internet con aplicaciones móviles, computación en la nube, dispositivos electrónicos y caseros conectados permanentemente a internet, fusión de plataformas tecnológicas y nuevos esquemas de desarrollo de software.

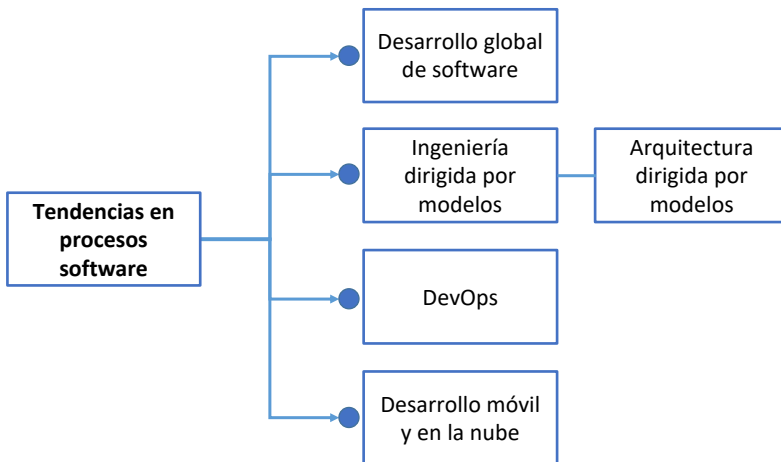


Figura 14.6. Nuevos enfoques de procesos de software

Fuente: elaboración propia.

14.4.1. Desarrollo global de software

El desarrollo global de software (**DGS**) nace de la externalización de servicios llevada al dominio de la ingeniería de software. El **DGS** implica la utilización de equipos ubicados en lugares dispersos en todo el mundo para desarrollar software comercialmente viable (Beecham *et al.*, 2015).

En un mundo globalizado, es de vital importancia que las empresas externalicen sus operaciones a países cercanos o lejanos logrando que se disminuyan costos de desarrollo gracias a los enlaces de internet de alta velocidad. Entre algunos beneficios que tiene este tipo de desarrollo, se encuentran los siguientes (Vizcaíno *et al.*, 2015):

- Desarrollar proyectos con recurso humano disperso a nivel mundial sin necesidad de incurrir en costos asociados al traslado de personal.

- Producir software para clientes lejanos sin necesidad de trasladar el equipo de desarrollo de software.
- Realizar jornadas de trabajo más extensas cuando los equipos de desarrollo se encuentran distribuidos en lugares con amplias diferencias horarias.
- Aprovechar el diverso conocimiento y múltiples destrezas de los equipos distribuidos (denominados equipos virtuales) a nivel mundial.

El desarrollo global de software presenta una serie de problemas agrupados en las siguientes categorías (Conchúir, 2010; Manoj Ray y Samuel, 2016):

- *Desafíos en la comunicación:* Relacionados con el intercambio de información completa y sin ambigüedades entre un emisor y un receptor para alcanzar un objetivo en común. Las reuniones sincrónicas son difíciles debido a las amplias zonas horarias.
- *Desafíos en la coordinación:* Relacionados con la ejecución de tareas para alcanzar objetivos e intereses comunes por medio de tareas asignadas a diferentes unidades jerárquicas. El costo de coordinación incluye la reparación de resultados mal interpretados y aumento de las modificaciones por realizar.
- *Desafíos en el control:* Relacionados con la gestión del proyecto que permite alcanzar los objetivos, las medidas o los niveles de calidad. La separación geográfica de las tareas causa problemas para transmitir la visión y la técnica adecuada para el desarrollo del proyecto.

Además, existen tres tipos de distancia que tienen influencia en el desarrollo en los proyectos de DGS (Ågerfalk *et al.*, 2005):

- *Distancia geográfica:* La medida de esfuerzo que un individuo necesita realizar para visitar otro punto alejado del primero.
- *Distancia temporal:* La medida de la deslocalización en tiempo (debido a múltiples zonas horarias con tecnologías de comunicación asincrónica) experimentada por los individuos que desean interactuar.
- *Distancia sociocultural:* La medida en que un individuo comprende las costumbres y la cultura de otros individuos.

El DGS presenta actualmente desafíos que deben ser abordados por la investigación en procesos de software. Es importante diseñar una estrategia de implementación de procesos para los equipos de desarrollo de software distribuido. Algunos desafíos concretos en esta área son la selección de

procesos apropiados para diferentes equipos distribuidos, la adaptación de procesos y la selección de un modelo apropiado de ciclo de vida de desarrollo de software (figura 14.7).

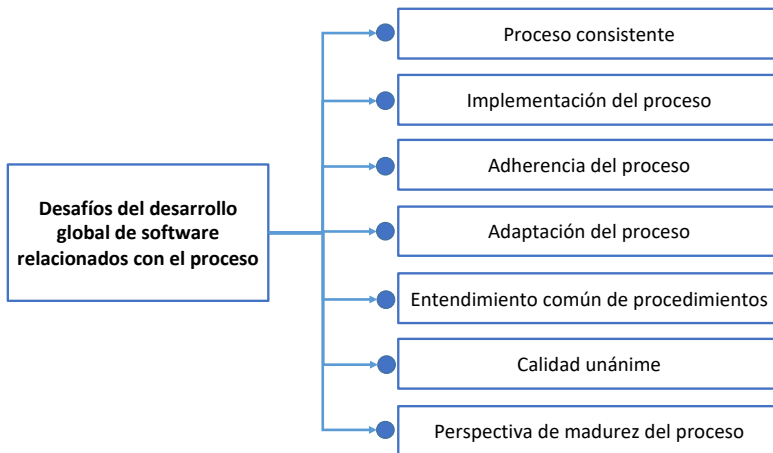


Figura 14.7. Desafíos del desarrollo global de software relacionados con procesos

Fuente: elaboración propia con base en Bhatti y Ahsan (2016).

Los procesos no equivalentes en sitios distribuidos ocurren cuando diferentes equipos siguen diferentes procesos para lograr objetivos similares. Esta diferencia de procesos en diferentes sitios de un proyecto causa productos de baja calidad. Es difícil garantizar los procesos consistentes y la calidad en todos los sitios de un proyecto porque los diferentes equipos tienen diferentes niveles de madurez del proceso y diferentes niveles de comprensión de los procedimientos de desarrollo de productos.

14.4.2. Ingeniería dirigida por modelos y arquitectura dirigida por modelos

La ingeniería dirigida por modelos (Model-Driven Engineering [**MDE**]) es la aplicación, basada en principios de modelos estructurados y procesables por máquina, a la ingeniería de sistemas (Schmidt, 2006). Los lenguajes de modelado que se utilizan en **MDE** se definen en general (pero no exclusivamente) mediante metamodelos que capturan la sintaxis abstracta y ciertas restricciones semánticas estáticas asociadas con el lenguaje (Paige *et al.*, 2014).

Un entorno de ingeniería basado en modelos permite a un modelador crear un modelo en un entorno de modelado. Posteriormente, el motor de ejecución del modelo (utilizando un enfoque de ejecución del modelo) traduce el modelo a una aplicación, como se observa en la figura 14.8.

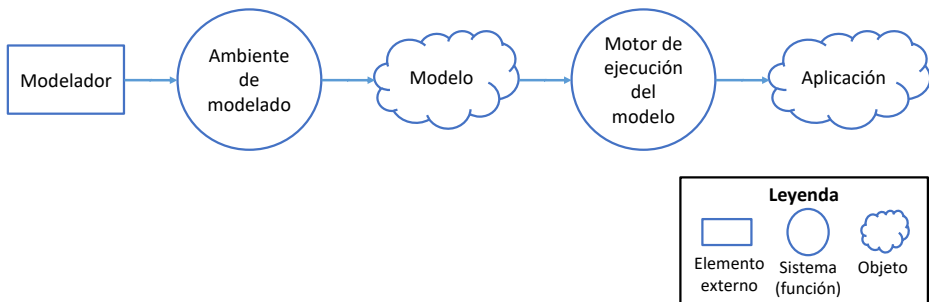


Figura 14.8. Ambiente MDE

Fuente: elaboración propia con base en Overeem *et al.* (2018).

En el enfoque **MDE**, conviven variados conceptos, entre ellos la arquitectura dirigida por modelos (Model Driven Architecture [**MDA**]). La **MDA** fue formalizada en 2001 y es un enfoque de diseño de software propuesto por el Object Management Group (**OMG**) para mejorar el desarrollo de aplicaciones por medio de la separación entre la lógica de negocios y la tecnología de plataforma subyacente (**OMG**, 2020).

La **MDA** recomienda separar “la especificación de la funcionalidad del sistema” de la “especificación de la implementación” de esa funcionalidad en una plataforma tecnológica específica. Para ello, se sugiere el uso de tres tipos de grupos de modelos (Miralles y Rouge, 2008; Verdier *et al.*, 2019):

- Modelo independiente de computación (Computing Independent Model [**CIM**]): Son modelos que describen el dominio del negocio sin describir cómo se realiza la aplicación.
- Modelos independientes de plataforma (Platform-Independent Model [**PIM**]): Son modelos que proporcionan una descripción de la estructura y las funciones de un sistema sin especificaciones técnicas del procesamiento de datos. Contienen solo conceptos del negocio.
- Modelos específicos de plataforma (Platform Specific Model [**PSM**]): Son modelos que definen cómo se implementan la estructura y las

funciones de un sistema en una plataforma específica. Contienen tanto conceptos de negocio como conceptos de implementación.

Un modelo **PSM** siempre se deriva de un modelo **PIM** a través de una o más transformaciones. Se puede derivar más de un **PSM** de un mismo **PIM**. Igualmente, también a través de transformaciones se puede derivar el modelo **PIM** desde un modelo **PSM**. El paso final consistirá en proyectar el modelo **PSM** en un determinado lenguaje de programación. La figura 14.9 presenta los modelos y su transformación.

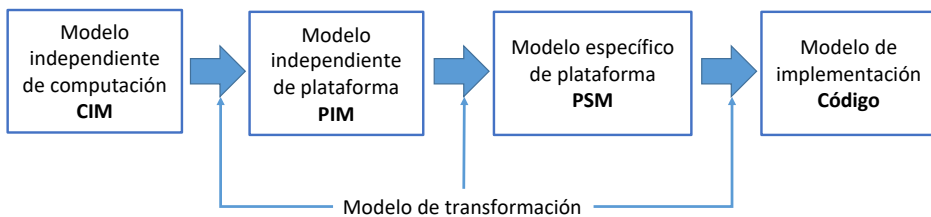


Figura 14.9. Proceso de transformación MDA

Fuente: elaboración propia con base en Kardoš y Drozdová (2010).

Los modelos al ser independientes ya no están vinculados entre sí. Por tanto, los aspectos de negocio y técnicos de un sistema pueden evolucionar a su propio ritmo y de acuerdo con la lógica de negocios que responde a las necesidades comerciales y a la tecnología que aprovecha los nuevos desarrollos según lo requiera el negocio.

Algunos impactos que tiene la **MDA** en el proceso de software son (OMG, 2014):

- ▶ Los *stakeholders* están más involucrados en comprender y validar el sistema. Hay más esfuerzo en producir un modelo preciso y completo.
- ▶ Existe un mayor esfuerzo previo en definir o ajustar y probar las transformaciones para un conjunto específico de tecnologías de plataforma. Estas inversiones iniciales producen rendimientos a medida que los requisitos del sistema evolucionan con el tiempo.
- ▶ Los componentes definidos en los modelos tienen límites e interfaces claros que los desarrolladores pueden codificar y el control de calidad puede validar.

- Los recursos del proyecto se dividen entre un equipo **PIM** y un equipo **PSM** que pueden ser independientes, pero cuando se combinan producen una solución sólida.
- Se dedica mucho menos tiempo a la codificación y depuración.
- Se puede simular un sistema antes de comprometerse con la implementación completa.
- Los diseños basados en modelos también pueden ayudar a automatizar el control de calidad y los procesos de aseguramiento de la calidad de software.

La **MDA** proporciona una base para el soporte automatizado de la ejecución del proceso de software. Los esfuerzos de investigación actuales se concentran principalmente en la automatización de las transformaciones de modelos para (Miralles y Rouge, 2008):

- Llevar a cabo un proceso **MDA** completo, es decir, un proceso que automatiza la evolución de los modelos desde el análisis hasta la implementación.
- Aumentar la productividad durante el desarrollo y facilitar la reutilización del código.

14.4.3. DevOps

El surgimiento de DevOps proviene de algunos desafíos presentes actualmente en los proyectos de desarrollo de software (Kneuper, 2018):

- La transición del software desarrollado a la producción comenzó a convertirse en un importante cuello de botella (presionado, además, por el tiempo de comercialización y la entrega más rápida de funcionalidades).
- Las diferentes culturas entre las unidades organizacionales de desarrollo y las unidades organizacionales de operaciones hacen esta transición difícil. Por lo general, demoran semanas, incluso pequeños cambios.
- El aumento en la cantidad de máquinas a operar, pasando de unas pocas máquinas físicas a cientos o miles de máquinas virtuales.

DevOps es una palabra artificial acuñada desde desarrollo (*development*) y operaciones (*operations*). La interacción de los ingenieros de software y los administradores de sistemas ha dado lugar a este término (Developer Operations [DevOps]), debido a la forma en que estos equipos

cruzan los límites de cada uno para funcionar como una única unidad lógica (Sacks, 2012).

El término DevOps es una combinación de desarrollo (desarrolladores que incluyen programadores, evaluadores y personal de control de calidad) y operaciones (representantes de expertos que ponen software en producción y administran la infraestructura de producción que incluyen administradores de sistemas, administradores de bases de datos y técnicos de redes). DevOps describe prácticas que agilizan el proceso de entrega de software, enfatizando en el aprendizaje, transmitiendo retroalimentación desde la producción al desarrollo y mejorando el tiempo de desarrollo. DevOps no solo permite entregar software más rápidamente, sino que también ayuda a producir un software de mayor calidad que esté más alineado con los requisitos individuales y las condiciones básicas (Hüttermann, 2012).

Una base conceptual de DevOps es la automatización del flujo de procesos que va desde la programación a la operación. Este flujo se denomina tubería de despliegue o flujo de valor (figura 14.10). Las flechas discontinuas denotan aquellas partes de la tubería que están automatizadas en integración/entrega/despliegue continuo.

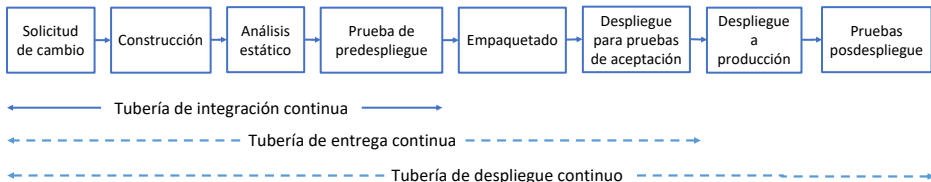


Figura 14.10. Tubería de despliegue

Fuente: elaboración propia con base en Kneuper (2018).

- Integración continua: Es una práctica incluida en la programación extrema (eXtreme Programming [XP]) (apartado 3.2) que consiste en reducir el tamaño de los cambios que conducen a una nueva compilación y prueba. En lugar de realizar una integración diaria, el producto se integra y se construye después de cada pequeño cambio completado. Se recomienda que cada desarrollador registre sus cambios y los integre al menos una vez al día (Kim *et al.*, 2016).

- Entrega continua: Extiende la integración continua al automatizar el empaquetado y la entrega del producto, además de la realización de pruebas de aceptación del paquete completo. El proceso se realiza desde el momento en que se efectúe un pequeño cambio en el producto (Kim *et al.*, 2016).
- Despliegue continuo: Incluye el último paso de despliegue en el entorno de producción sin ninguna intervención manual. Cada cambio que el desarrollador considere completo y que pase todas las comprobaciones y pruebas de control de calidad automatizadas pasará directamente a producción. Para ello, se necesita una estrecha cooperación entre el equipo de desarrollo y el equipo de operaciones (Kim *et al.*, 2016).

Todas las etapas de integración continua, entrega continua y despliegue continuo deben automatizarse utilizando herramientas adecuadas.

La figura 14.11 muestra el proceso genérico, que tiene como objetivo integrar mejor los procesos de negocio de desarrollo, producción y operaciones con la tecnología adecuada. En lugar de quedarse con conceptos de procesos altamente artificiales que nunca funcionarán, las organizaciones configuran la entrega continua con pequeñas actualizaciones (Ebert *et al.*, 2016).

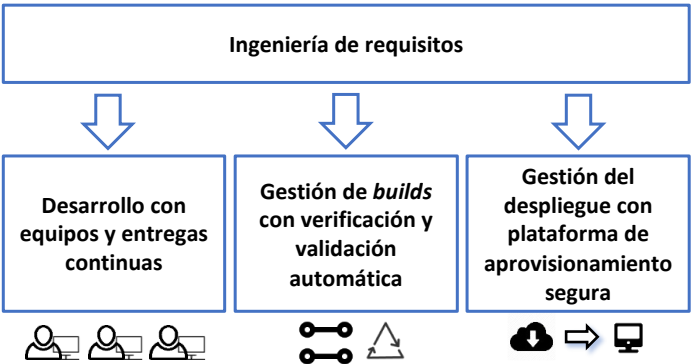


Figura 14.11. Procesos de producción y entrega genéricos de DevOps

Fuente: elaboración propia con base en Ebert *et al.* (2016).

DevOps abarca numerosas actividades y aspectos, como (Lwakatare *et al.*, 2015):

- ▶ *Monitoreo*: El personal de operaciones supervisa los sistemas y la infraestructura subyacente para determinar la asignación de recursos adecuada y para detectar, informar y corregir los problemas que ocurren durante o después de las actualizaciones del sistema.
- ▶ *Automatización*: La automatización es esencial para DevOps al obtener una retroalimentación rápida. Es necesaria en los procesos de operaciones y es necesaria una mayor automatización de las pruebas en el proceso de desarrollo de software.
- ▶ *Medición*: La medición en DevOps se logra midiendo el esfuerzo del proceso de software más allá del aseguramiento de la calidad utilizando el rendimiento en tiempo real y los datos de uso de la funcionalidad del software en el entorno de producción. El impacto radica en que los esfuerzos de desarrollo de software se miden de manera efectiva.
- ▶ *Colaboración*: Crea una cultura en la que las personas comparten ideas, procesos y herramientas. La colaboración afecta la estructura del equipo y las habilidades requeridas de los desarrolladores de software y el personal de operaciones.

Se requiere seguir investigando en arquitecturas adecuadas para DevOps y más evidencias empíricas sobre la adopción de DevOps en la industria de desarrollo de software.

14.4.4. Desarrollo móvil y en la nube

A medida que las aplicaciones móviles se vuelven más complejas y de misión crítica, las empresas desarrolladoras de software deben introducir nuevos procesos. El desarrollo de aplicaciones móviles requiere entender las características de esas aplicaciones, como se presentan en la tabla 14.4.

Con respecto al proceso de desarrollo de software de estas aplicaciones, se destaca que el cambio es más frecuente, los requisitos de usuario cambian con el tiempo, el tiempo de desarrollo es corto y se realiza mucho énfasis en la interfaz de usuario (Ashishdeep *et al.*, 2016). Por tanto, existe una mayor necesidad de crear prototipos de interfaces de usuario, particularmente cuando se admitirán múltiples dispositivos.

Tabla 14.4. Características de las aplicaciones móviles

Característica	Descripción
Energía limitada	Cada dispositivo móvil tiene una batería con un periodo de vida específico. Por consiguiente, la aplicación no debe requerir una gran cantidad de recursos de <i>hardware</i> .
Interfaz gráfica	El diseño de la interfaz es limitado debido al pequeño tamaño de la pantalla.
Interfaz de entrada	Se identifica por los medios de interacción entre el usuario y la aplicación, como teclado, pantalla táctil o de voz, y el reconocimiento de imágenes.
Ancho de banda	Se debe considerar un ancho de banda máximo en ocasiones y el mínimo en otras circunstancias.
Tipo de conectividad	Las aplicaciones móviles pueden admitir varios tipos de conectividad como Bluetooth, 3G, 4G, wifi, inalámbrico, etc.
Interrupción constante de actividades	Las aplicaciones móviles deben estar diseñadas para diferentes escenarios debido a la interrupción de actividades con frecuencia, como poca batería o falta de conexión, etc.
Rendimiento limitado	Los dispositivos móviles tienen limitaciones de ciertos recursos, como procesamiento, memoria y conectividad.
Memoria reducida	Debido al tamaño limitado del <i>hardware</i> , se puede acceder a menos memoria.
Portabilidad	Debe soportar un amplio rango de dispositivos y sistemas operativos.

Fuente: elaboración propia con base en Aldayel y Alnafjan (2017).

Las aplicaciones móviles se pueden clasificar en (Kaur y Kaur, 2015; Popa, 2013):

- *Aplicaciones nativas*: Se implementan para una determinada plataforma utilizando kits de desarrollo de software (Software Development Kit [**SDK**]) y herramientas proporcionadas por el propietario de la plataforma. Pueden limitar la cantidad de dispositivos móviles que ejecutan la aplicación, pero puede beneficiarse de las características de la plataforma ofrecidas para los desarrolladores.
- *Aplicaciones web*: Se implementan utilizando las tecnologías web estándar (Hypertext Markup Language [**HTML**], Cascading Style Sheets [**CSS**], JavaScript [**JS**], etc.) y se acceden a través de un navegador web móvil. Garantizan la disponibilidad de aplicaciones para diferentes plataformas móviles, pero no pueden utilizar todas las capacidades ofrecidas ni pueden utilizarse sin conexión a internet.

- *Aplicaciones híbridas*: Combinan las ventajas tanto de las aplicaciones nativas como web.

Las pruebas son un área importante de investigación en procesos de desarrollo de software móvil. No es suficiente probar una aplicación móvil en un emulador, también debe probarse en muchos dispositivos diferentes que ejecutan diferentes versiones del sistema operativo en varias redes de telecomunicaciones. Las *suites* de pruebas integradas simplificarían este proceso. Otra área de investigación implica el mantenimiento de aplicaciones en un contexto de rápido cambio de las plataformas móviles. Si bien los consumidores “tempranos” están dispuestos a actualizar su dispositivo y sus aplicaciones, es menos probable que la mayoría de los usuarios lo hagan (Wasserman, 2010).

Entre algunos modelos de procesos de software orientados hacia las aplicaciones móviles se encuentran:

- *Mobile-D*: Propuesto en 2005 y está basado en las metodologías Crystal, Rational Unified Process (**RUP**) y programación extrema. Consta de las fases: explorar, inicializar, caracterizar el producto, estabilizar, pruebas y reparación del sistema. Se recomienda que sea utilizado por un pequeño equipo involucrado en un ciclo de desarrollo corto (Abrahamsson *et al.*, 2004).
- *Ingeniería de métodos híbridos (Hybrid Method Engineering [HMR])*: Propuesta en 2008. Tiene cuatro iteraciones: La primera iteración tiene en cuenta las consideraciones del mercado; la segunda iteración, el desarrollo de nuevos productos; la tercera iteración, las ideas de desarrollo de software adaptativo, y la cuarta, la creación de prototipos (Rahimian y Ramsin, 2008).
- *Metodología ágil de software de aplicación móvil (Mobile Application Software Agile Methodology [MASAM])*: Se basa en **XP** y Software Process Engineering Metamodel (**SPEM**). Tiene cuatro fases: la fase de preparación que asigna roles y responsabilidades; la fase de realización que se centra en comprender los requisitos del usuario; la fase de desarrollo del producto que proporciona una secuencia de desarrollo iterativa de **XP**, y la fase de comercialización que se centra en el lanzamiento de productos y las actividades de venta de productos (Jeong *et al.*, 2008).
- *SleSS*: Propuesto en 2011. Asimila *scrum* y Lean Six Sigma centrándose en la mejora de procesos y la gestión de proyectos (Da Cunha *et al.*, 2011).

El desarrollo y la implementación de aplicaciones móviles se están moviendo actualmente hacia la nube. La computación en la nube se refiere al uso de software, infraestructura de red y capacidades para proporcionar recursos a los usuarios a través del modelo “pago por uso”. Es una arquitectura heterogénea, que se beneficia de una serie de tecnologías diferentes para proporcionar servicios remotos (Moreira *et al.*, 2015). Este nuevo paradigma informático afectará, entre otros muchos aspectos, los procesos de software y las herramientas de desarrollo.

Entre los modelos de la computación en la nube se encuentra software como servicio (software as a service [SaaS]) que se ejecuta en la plataforma de internet. SaaS considera el software como un servicio que se proporcionará a los usuarios. Los proveedores de servicios SaaS implementan la aplicación en sus servidores y los clientes utilizan servicios de software a través de los navegadores. Con este modelo, los proveedores de servicios son responsables del mantenimiento del software, de la gestión de software y de las instalaciones de *hardware* para ejecutar software (Wang *et al.*, 2012).

Se requiere investigación adicional sobre procesos de software cuyas definiciones, modelado, control, mejora y automatización estén soportados con servicios implementados en la nube como servicios SaaS.

► Desafíos

La necesidad de coordinación y agilidad son un desafío en un entorno de desarrollo de software global, donde el proceso de software es ejecutado por equipos distribuidos en diferentes países y zonas horarias (Herbsleb, 2007). Las prácticas para coordinar estos equipos difieren de las tradicionales considerando nuevos aspectos, como cultura, establecimiento de contactos fuertes y directos con clientes en ubicaciones remotas, y creación de redes y marcos de cooperación con otras organizaciones (Fuggetta y Di Nitto, 2014).

Por otro lado, aunque el enfoque de desarrollo de software dirigido por modelos fue propuesto hace varios años, aún su implementación total en la industria es limitada. Este enfoque plantea una transición de los roles de los modelos de software desde la documentación hasta el desarrollo, lo que implica una mejora de los modelos en aspectos como la integridad y la precisión, en comparación con el uso tradicional de los modelos. Además,

exige un mecanismo eficiente de mapeos de implementación de modelos, que no se trata solo de generar modelos con código prescrito, sino la gestión de la coherencia entre modelo y código. Este enfoque puede sobrevivir a largo plazo si el costo del mapeo de implementación del modelo excede significativamente trabajar con el código directamente.

La automatización del proceso de software también es un desafío en que se deben apoyar las fases finales de desarrollo de software y las fases transversales, como la gestión de la configuración, el aseguramiento de la calidad y la integración de componentes de software. Sin embargo, el desarrollo de software es un proceso centrado en las personas, en el que la creatividad y la autonomía desempeñan un papel crucial. Por tanto, la mayoría de las actividades que ocurren en un proceso de software no pueden automatizarse de manera rígida (Basha *et al.*, 2012).

Con respecto a DevOps, los principales desafíos y dificultades se presentan en áreas como (Senapathi *et al.*, 2018):

- ▶ *Personal con las habilidades técnicas adecuadas*: La falta de personal debidamente capacitado puede hacer que el proceso de adopción de DevOps se ralentice porque faltan las capacidades necesarias.
- ▶ *Resistencia al cambio*: La transición a una forma de trabajar DevOps necesita motivación para superar la resistencia a este cambio y hacer frente a la incertidumbre de cómo este cambio afectará al equipo de desarrollo.
- ▶ *Cambiar la pila de tecnología y las herramientas*: La transición del producto a nuevos entornos (como la nube) y nuevas arquitecturas (como microservicios) son facilitadores de la adopción de DevOps. Estos cambios se pueden considerar complejos y desafiantes, ya que para los equipos de desarrollo experimentar y configurar los entornos, las tecnologías y herramientas son un desafío.
- ▶ *Incertidumbre en las responsabilidades*: El cambio en las responsabilidades asociadas con la adopción de DevOps es gradual, lo que conlleva malentendidos sobre quién es responsable de las actividades.

Todas estas cuestiones destacan que la distinción clásica entre diseño, implementación y operación tiende a desaparecer o ser redefinidos radicalmente. El diseño de un producto de software depende de las características de la infraestructura para implementarlo.

► Ejemplos

El ejemplo de DevOps es la empresa FinCom1, una de las primeras grandes organizaciones que comenzó a implementar DevOps en los Países Bajos. A continuación, se presentan las respuestas a una breve entrevista sobre la implementación de DevOps en esa empresa (Erich *et al.*, 2017).

1. ¿Por qué FinCom1 decidió implementar DevOps? FinCom1 tenía tres objetivos principales que quería lograr mediante la implementación de DevOps:
 - Reducir el tiempo de entrega: Comenzar nuevos proyectos tomó mucho tiempo en la organización, ya que los equipos tenían problemas para obtener recursos de desarrollo como servidores y software.
 - Mejorar la resolución de problemas: Durante el propio proyecto, era difícil resolver problemas en los que se necesitaba una estrecha colaboración entre el personal de desarrollo y el de operaciones. Al utilizar DevOps, FinCom1 quería reducir el tiempo necesario para resolver estos problemas.
 - Mejorar la retroalimentación: La organización también quería aumentar el conocimiento disponible para el personal de desarrollo con respecto a la satisfacción del cliente. Antes de adoptar DevOps, el personal de operaciones recopilaba información sobre la satisfacción del cliente, pero rara vez se compartía con el personal de desarrollo.
2. ¿Cómo implementó FinCom1 DevOps?
 - Adoptó *scrum*, Information Technology Infrastructure Library (ITIL) y Capability Maturity Model Integration (CMMI) antes de DevOps.
 - Presentó los equipos DevOps en los que el personal de desarrollo y operaciones trabajan juntos a diario.
 - Introdujo un nuevo rol de empleado llamado ingeniero DevOps con habilidades tanto en desarrollo como en operaciones.
 - Capacitó a su personal administrativo para que pudiera administrar personal de desarrollo y operaciones.
 - El departamento de recursos humanos cambió los procedimientos de evaluación para permitir una evaluación adecuada del personal activo en un equipo DevOps.

3. ¿Qué problemas encontró FinCom1 al implementar DevOps?

- FinCom1 consideró que la apertura es un componente principal de una cultura DevOps, se refiere a que los empleados comparten lo que tienen en mente tanto en cuestiones positivas como negativas.
- El personal de desarrollo también se resistió al aumento de responsabilidad. En el pasado, el personal de operaciones era responsable de la disponibilidad y el rendimiento de los sistemas. FinCom1 hizo que los equipos DevOps, que incluían tanto personal de desarrollo como de operaciones, fueran responsables de la disponibilidad y el rendimiento de los sistemas. Por tanto, el personal de desarrollo tuvo que asumir más responsabilidades que antes.
- En DevOps, el equipo es responsable de la disponibilidad de su servicio o sistema en desarrollo. Los equipos organizan rotaciones de guardia cuando se espera que un servicio o sistema esté disponible en todo momento. Son periodos a los que se asignan personas como responsables del sistema.

Las respuestas de la entrevista se resumen en la tabla 14.5.

Tabla 14.5. Resumen de respuestas de FinCom1

Aspecto	Opinión de FinCom
DevOps	Define DevOps como la formalización del concepto ágil de que un equipo es responsable de todos los aspectos de un producto, lo que incluye las operaciones.
Individual	Utiliza el título de ingeniero DevOps para todo el personal de operaciones y desarrollo que trabaja en equipos DevOps.
Equipo	Utiliza equipos DevOps que son responsables tanto del desarrollo como de los sistemas y servicios operativos.
Departamento	Departamentos fusionados de desarrollo y operaciones en departamentos DevOps.
Cultura de colaboración	Tiene personal de desarrollo y operaciones trabajando juntos diariamente.
Automatización	Automatiza el proceso de lanzamiento de software a través de pruebas automatizadas y está implementando entrega continua.
Medición	Mide la eficacia de los sistemas y equipos mediante un ciclo de inspección y adaptación. Mide el tiempo entre el inicio de una idea y su disponibilidad como un servicio para los clientes. La principal medida de rendimiento es la satisfacción de los clientes.

Aspecto	Opinión de FinCom
Monitoreo	Tiene sistemas de monitoreo del personal de operaciones mientras está en contacto cercano con los usuarios.
Lean	Aplica principios <i>lean</i> para mejorar tanto la eficacia como la eficiencia de los procesos, pero considera que está débilmente relacionado con DevOps.
Compartir	Alienta a los empleados a decir lo que piensan con respecto a las mejoras de procesos y productos, y utiliza la apertura como una medida del desempeño de los empleados.
Microservicios	Contempla DevOps como una forma de organizar procesos, separada de la arquitectura.

Fuente: elaboración propia con base en Erich *et al.* (2017).

► Resumen

El área del conocimiento de la ingeniería del proceso software sigue generando bastantes esfuerzos de investigación que se reflejan en amplias y diversas publicaciones, eventos y herramientas a nivel mundial.

El **IPRC** ha generado un marco de investigación en procesos con cuatro temas de investigación: relaciones entre calidad del proceso y producto, ingeniería de procesos, gestión de procesos de proyectos y despliegue del proceso. Cada tema contiene un conjunto de nodos de investigación y, a su vez, cada nodo incluye un conjunto de preguntas de investigación bastante relevantes, aún no resueltas y que pueden servir como base para formular proyectos de investigación en el área.

La minería del proceso de software es otra área de investigación cuyo objetivo es descubrir en forma automática el modelo del proceso de software subyacente a partir de los datos del registro de eventos del proceso. La minería de datos está orientada al descubrimiento, la conformidad y la mejora del proceso de software.

El aprendizaje automático y el aprendizaje profundo son dos áreas de las ciencias de la computación y la inteligencia artificial (**IA**) que permiten que los computadores aprendan. Estas dos áreas tienen una amplia gama de aplicaciones que incluye entre ellas el proceso de software. En el área del proceso de software, el aprendizaje automático ha tenido bastante éxito en resolver problemas como asignación de recursos humanos, identificación de riesgos, predicción de esfuerzo, estimación de costos, localización de

fallas y detección de errores. A su vez, el aprendizaje profundo se ha destacado en problemas de clasificación, sistemas de recomendación, generación de secuencias y predicción y recuperación/extracción de información.

Por último, los nuevos modelos de procesos de software continúan surgiendo, especialmente orientados a contextos específicos. Los nuevos modelos propuestos requieren aún evidencia empírica que permitan masificar con éxito su uso a nivel industrial. Entre estos modelos se encuentran desarrollo global de software, procesos dirigidos por modelos, orientados al desarrollo de aplicaciones móviles y en la nube, que vinculen en forma efectiva los departamentos de desarrollo y operaciones.

► Ejercicios

1. Consulte el documento **IPRC** (Forrester, 2006) y seleccione diez preguntas de investigación que considere las más relevantes del nodo de ingeniería de procesos.
2. ¿En qué consiste la minería del proceso y qué implicaciones tiene en el proceso de software?
3. ¿Qué etapas del proceso de software se pueden beneficiar de la aplicación de técnicas de *machine learning* y *deep learning*?
4. Investigar en internet qué métricas software se han definido e implementado para proyectos DevOps.
5. Investigar en internet algunas herramientas que implementen las diferentes fases de la tubería de despliegue de DevOps.
6. ¿Qué soluciones puede proponer para afrontar los desafíos en la comunicación, coordinación y control presentes en el desarrollo global de software?
7. Explicar las diferencias entre **CIM**, **PIM** y **PSM** en el desarrollo de software dirigido por modelos.
8. Investigar en internet diferentes herramientas software libres y comerciales orientadas al desarrollo de software dirigido por modelos.
9. ¿Qué ventajas tiene para un equipo de desarrollo construir software en la nube?

► Referencias

- Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jäälinoja, J., Korkala, M. ... y Salo, O. (2004). Mobile-D: An agile approach for mobile application development. En *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (pp. 174-175). <https://doi.org/10.1145/1028664.1028736>
- Ågerfalk, P. J., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B. y Conchúir, E. Ó. (2005). A framework for considering opportunities and threats in distributed software development. En *Proceedings of the International Workshop on Distributed Software Development* (pp. 47-61). Austrian Computer Society. <https://researchrepository.ul.ie/ndownloader/files/35267047/1>
- Akhtar, N. y Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6, 14410-14430. <https://doi.org/10.1109/ACCESS.2018.2807385>
- Aldayel, A. y Alnafjan, K. (2017). Challenges and best practices for mobile application development. En *Proceedings of the International Conference on Compute and Data Analysis* (pp. 41-48). <https://doi.org/10.1145/3093241.3093245>
- Ashishdeep, A., Bhatia, J. y Varma, K. (2016). Software process models for mobile application development: A review. *Computer Science and Electronic Journal*, 7(1), 150-153. <https://csjournals.com/IJCSC/PDF7-1/20.%20Anitha.pdf>
- Basha, N. M. J., Moiz, S. A. y Rizwanullah, M. (2012). Model based software development: Issues & challenges. *Special Issue of International Journal of Computer Science & Informatics (IJCSI)*, 2(1), 226-230. <https://doi.org/10.47893/ijcsi.2013.1123>
- Beecham, S., Richardson, I. y Noll, J. (2015). Assessing the strength of global teaming practices: A pilot study. En *2015 IEEE 10th International Conference on Global Software Engineering* (pp. 110-114). *IEEE*. <https://doi.org/10.1109/ICGSE.2015.14>
- Bhatti, M. W. y Ahsan, A. (2016). Global software development: An exploratory study of challenges of globalization, HRM practices and process improvement. *Review of Managerial Science*, 10(4), 649-682. <https://doi.org/10.1007/s11846-015-0171-y>
- Blum, F. R. (2016). Mining software process lines. En *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 839-842). <https://doi.org/10.1145/2889160.2889267>
- Cabac, L. y Denz, N. (2008). Net components for the integration of process mining into agent-oriented software engineering. En K. Jensen, W. M. P. van der Aalst y J. Billington (eds.), *Transactions on Petri nets and other models of concurrency I. Lecture notes in computer science* (pp. 86-103). Springer. https://doi.org/10.1007/978-3-540-89287-8_6
- Caldeira, J. y Abreu, F. B. e. (2016). Software development process mining: Discovery, conformance checking and enhancement. En *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)* (pp. 254-259). *IEEE*. <https://doi.org/10.1109/QUATIC.2016.061>

- Conchúir, E. Ó. (2010). *Global software development: A multiple-case study of the realisation of the benefits* [tesis doctoral, University of Limerick]. <https://researchrepository.ul.ie/ndownloader/files/35241937/1>
- Da Cunha, T. F. V., Dantas, V. L. y Andrade, R. M. (2011). SLeSS: A Scrum and Lean Six Sigma integration approach for the development of software customization for mobile phones. En *2011 25th Brazilian Symposium on Software Engineering* (pp. 283-292). **IEEE**. <https://doi.org/10.1109/SBES.2011.38>
- Del Carpio, A. F. y Angarita, L. B. (2020). Trends in software engineering processes using deep learning: A systematic literature review. En *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 445-454). **IEEE**. <https://doi.org/10.1109/SEAA51224.2020.00077>
- Dong, L., Liu, B., Li, Z., Wu, O., Babar, M. A. y Xue, B. (2017). A mapping study on mining software process. En *2017 24th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 51-60). **IEEE**. <https://doi.org/10.1109/APSEC.2017.11>
- Ebert, C., Gallardo, G., Hernantes, J. y Serrano, N. (2016). DevOps. **IEEE Software**, 33(3), 94-100. <https://doi.org/10.1109/MS.2016.68>
- Erich, F. M., Amrit, C. y Daneva, M. (2017). A qualitative study of DevOps usage in practice. *Journal of software: Evolution and Process*, 29(6), e1885. <https://doi.org/10.1002/smr.1885>
- Fakhoury, S., Arnaoudova, V., Noiseux, C., Khomh, F. y Antoniol, G. (2018). Keep it simple: Is deep learning good for linguistic smell detection? En *2018 IEEE 25Th international conference on software analysis, evolution and reengineering (SANER)* (pp. 602-611). **IEEE**. <https://doi.org/10.1109/SANER.2018.8330265>
- Falcini, F., Lami, G. y Costanza, A. M. (2017). Deep learning in automotive software. **IEEE Software**, 34(3), 56-63. <https://doi.org/10.1109/MS.2017.79>
- Fernández del Carpio, A. y Bermón Angarita, L. (2018). Techniques based on data science for software processes: A systematic literature review. En I. Stamelos, R. O'Connor, T. Rout y A. Dorling (eds.), *Software Process Improvement and Capability Determination. SPICE 2018. Communications in Computer and Information Science* (pp. 16-30). Springer. https://doi.org/10.1007/978-3-030-00623-5_2
- Forrester, E. (ed.) (2006). *A process research framework: The international process research consortium*. Carnegie Mellon University, Software Engineering Institute.
- Fuggetta, A. (2000). Software process: A roadmap. En *Proceedings of the Conference on the Future of Software Engineering* (pp. 25-34). <https://dl.acm.org/doi/pdf/10.1145/336512.336521>
- Fuggetta, A. y Di Nitto, E. (2014). Software process. En *Future of Software Engineering Proceedings* (pp. 1-12). <https://doi.org/10.1145/2593882.2593883>
- Godfrey, M. W., Hassan, A. E., Herbsleb, J., Murphy, G. C., Robillard, M., Devanbu, P... y Notkin, D. (2008). Future of mining software archives: A roundtable. **IEEE Software**, 26(1), 67-70. <https://doi.org/10.1109/MS.2009.10>
- Guo, J., Cheng, J. y Cleland-Huang, J. (2017). Semantically enhanced software traceability using deep learning techniques. En *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)* (pp. 3-14). **IEEE**. <https://doi.org/10.1109/ICSE.2017.9>

- Herbsleb, J. D. (2007). Global software engineering: The future of socio-technical coordination. En *Future of software engineering (FOSE'07)* (pp. 188-198). **IEEE**. <https://doi.org/10.1109/FOSE.2007.11>
- Hüttermann, M. (2012). Beginning devops for developers. En *DevOps for Developers* (pp. 3-13). Apress. https://doi.org/10.1007/978-1-4302-4570-4_1
- Jeong, Y. J., Lee, J. H. y Shin, G. S. (2008). Development process of mobile application SW based on agile methodology. En *2008 10th International Conference on Advanced Communication Technology* (vol. 1, pp. 362-366). **IEEE**. <https://doi.org/10.1109/ICACT.2008.4493779>
- Kardoš, M. y Drozdová, M. (2010). Analytical method of **CIM** to **PIM** transformation in model driven architecture (MDA). *Journal of Information and Organizational Sciences*, 34(1), 89-99. <https://hrcak.srce.hr/file/83906>
- Kaur, A. y Kaur, K. (2015). Suitability of existing software development life cycle (**SDLC**) in context of mobile application development life cycle (**MADLC**). *International Journal of Computer Applications*, 116(19), 1-6. <https://research.ijcaonline.org/volume116/number19/pxc3902785.pdf>
- Kim, G., Humble, J., Debois, P., Willis, J. y Forsgren, N. (2016). *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. IT Revolution.
- Kneuper, R. (2018). *Software processes and life cycle models: An introduction to modelling, using and managing agile, plan-driven and hybrid processes*. Springer. <https://doi.org/10.1007/978-3-319-98845-0>
- LeCun, Y., Bengio, Y. y Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-44. <https://doi.org/10.1038/nature14539>
- Li, Z., Zhao, H., Shi, J., Huang, Y. y Xiong, J. (2019). An intelligent fuzzing data generation method based on deep adversarial learning. *IEEE Access*, 7, 49327-49340. <https://doi.org/10.1109/ACCESS.2019.2911121>
- Lwakatare, L. E., Kuvaja, P. y Oivo, M. (2015). Dimensions of DevOps. En C. Lassenius, T. Dingsøyr y M. Paasivaara (eds.), *Agile Processes in Software Engineering and Extreme Programming*. **XP 2015. Lecture Notes in Business Information Processing** (pp. 212-217). Springer. https://doi.org/10.1007/978-3-319-18612-2_19
- Manoj Ray, D. y Samuel, P. (2016). Improving the productivity in global software development. En V. Snášel, A. Abraham, P. Krömer, M. Pant y A. Muda (eds.), *Innovations in bio-inspired computing and applications: Advances in intelligent systems and computing* (pp. 175-185). Springer. https://doi.org/10.1007/978-3-319-28031-8_15
- Marshall, S. (2015). *Machine learning an algorithm perspective*. CRC Press.
- Miralles, A. y Rouge, T. L. (2008). Modeling with enriched model driven architecture. En *Encyclopedia of geographical information sciences* (pp. 700-705). Springer. <https://dx.doi.org/10.1007/978-0-387-35973-1>
- Moreira, F., Cota, M. P. y Gonçalves, R. (2015). The influence of the use of mobile devices and the cloud computing in organizations. En A. Rocha, A. Correia, S. Cos-tanzo y L. Reis (eds.), *New contributions in information systems and technologies: Advances in intelligent systems and computing* (vol. 1, pp. 275-284). Springer. https://doi.org/10.1007/978-3-319-16486-1_28

- Murphy, K. P. (2011). *Machine learning: A probabilistic perspective*. MIT Press.
- Ng, A., Ngiam, J., Foo, C. Y., Mai, Y., Suen, C., Coates, A. ... y Tandon, S. (2013). *Unsupervised feature learning and deep learning*. <https://redirect.cs.umbc.edu/courses/pub/www/courses/graduate/678/spring15/visionaudio.pdf>
- Object Management Group (2014). *Object Management Group Model Driven Architecture (MDA) MDA Guide rev. 2.0*. <https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf>
- Object Management Group (2020). *MDA® - The Architecture of Choice for a Changing World*. <https://www.omg.org/mda/>
- Overeem, M., Jansen, S. y Fortuin, S. (2018). Generative versus Interpretive Model-Driven Development: Moving Past 'It Depends'. En L. Pires, S. Hammoudi y B. Selic (eds.), *Model-Driven Engineering and Software Development. MODELSWARD 2017. Communications in Computer and Information Science* (pp. 222-246). Springer. https://doi.org/10.1007/978-3-319-94764-8_10
- Paige, R. F., Kolovos, D. S. y Polack, F. A. (2014). A tutorial on metamodeling for grammar researchers. *Science of Computer Programming*, 96, 396-416. <https://doi.org/10.1016/j.scico.2014.05.007>
- Popa, M. (2013). Considerations regarding the cross-platform mobile application development process. *Economy Informatics*, 13(1), 40-52. <https://www.economyinformatics.ase.ro/content/EN13/04%20-%20Popa.pdf>
- Rahimian, V. y Ramsin, R. (2008). Designing an agile methodology for mobile software development: A hybrid method engineering approach. En 2008 *Second International Conference on Research Challenges in Information Science* (pp. 337-342). **IEEE**. <https://doi.org/10.1109/RCIS.2008.4632123>
- Rubin, V., Günther, C. W., van der Aalst, W. M. P., Kindler, E., van Dongen, B. F. y Schäfer, W. (2007). Process mining framework for software processes. En Q. Wang, D. Pfahl y D. M. Raffo (eds.), *Software Process Dynamics and Agility. ICSP 2007. Lecture Notes in Computer Science* (pp. 169-181). Springer. https://doi.org/10.1007/978-3-540-72426-1_15
- Rui, Z., Tong, L., Qi, M., Zhenli, H., Qian, Y. y Yiquan, W. (2018). Data-driven bilayer software process mining. *Journal of Software*, 29(11), 3455-3483. <http://dx.doi.org/10.13328/j.cnki.jos.005304>
- Sacks, M. (2012). DevOps principles for successful web sites. En *Pro website development and operations: Streamlining DevOps for large-scale websites* (pp. 1-14). Apress. https://doi.org/10.1007/978-1-4302-3970-3_1
- Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. **IEEE Computer**, 2(39), 25-31. <https://doi.org/10.1109/MC.2006.58>
- Senapathi, M., Buchan, J. y Osman, H. (2018). DevOps capabilities, practices, and challenges: Insights from a case study. En *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (pp. 57-67). <https://doi.org/10.1145/3210459.3210465>
- Shrestha, A. y Mahmood, A. (2019). Review of deep learning algorithms and architectures. **IEEE Access**, 7, 53040-53065. <https://doi.org/10.1109/ACCESS.2019.2912200>

- Swamynathan, M. (2019). *Mastering machine learning with python in six steps: A practical implementation guide to predictive data analytics using python*. Apress. <https://doi.org/10.1007/978-1-4842-4947-5>
- Van der Aalst, W. M. (2011). *Process mining: Discovery, conformance and enhancement of business processes*. Springer.
- Verdier, F., Seriai, A. D. y Tiam, R. T. (2019). Combining model-driven architecture and software product line engineering: Reuse of platform-specific assets. En S. Hammoudi, L. Pires y B. Selic (eds.), *Model-Driven Engineering and Software Development. MODELSWARD 2018. Communications in Computer and Information Science* (pp. 430-454). Springer. https://doi.org/10.1007/978-3-030-11030-7_19
- Vizcaíno, A., García, F. y Piattini, M. (2015). Visión general del desarrollo global de software. *International Journal of Information Systems and Software Engineering for Big Companies*, 1(1), 8-22. <http://www.uajournals.com/ojs/index.php/ijisebc/article/view/1/1>
- Wang, J., Luo, W., Wu, X., Li, T., Qian, Y. y Xie, Z. (2012). An approach to modeling SaaS-oriented software service processes. En *2012 International Conference on System Science and Engineering (ICSSE)* (pp. 573-577). **IEEE**. <https://doi.org/10.1109/ICSSE.2012.6257252>
- Wasserman, A. I. (2010). Software engineering issues for mobile application development. En *Proceedings of the FSE/SDP workshop on Future of software engineering research* (pp. 397-400). <https://doi.org/10.1145/1882362.1882443>

► Reflexión final

El estudio y la investigación del desarrollo de software como un proceso ha ayudado significativamente a identificar las diferentes dimensiones del desarrollo de software y los problemas que deben abordarse para establecer prácticas efectivas durante el desarrollo de proyectos específicos.

La ingeniería de procesos de software nace de preocupaciones originadas desde la perspectiva de la calidad en la mayoría de los sectores industriales y, en concreto, de los procesos de producción. Los procesos son importantes porque la industria se preocupa por sus cualidades intrínsecas (uniformidad de los comportamientos en los diferentes proyectos, productividad, etc.), y así mejorar el tiempo de comercialización y reducir los costos de producción.

En particular, el software como producto tiene particularidades bastante especiales, como complejidad, intangibilidad, incertidumbres iniciales (el cliente no sabe lo que quiere), generación de cambios permanentes en requisitos, etc. Por tanto, el desarrollo de productos de software requiere un flujo disciplinado de actividades bien definidas y gestionadas que tendrán una profunda influencia en la calidad de los productos de software construidos.

La ingeniería de procesos de software recogió los trabajos iniciales sobre procesos a nivel industrial (calidad del proceso, control estadístico del proceso, etc.) y luego adoptó y modificó los estándares de la International Organization for Standardization (**ISO**), u Organización Internacional de Normalización, para adecuarlos a las particularidades de las empresas desarrolladoras de software, generando modelos propios del área, como Capability Maturity Model Integration (**CMMI**). Por último, el proceso de software se consolida como parte fundamental de la ingeniería de software, en especial cuando comenzó a formar parte como un área de conocimiento propia del cuerpo del conocimiento en ingeniería de software (**SWEBOK v3.0**).

Los procesos de software no forman parte del área técnica del desarrollo de software. Por el contrario, lo hacen de la gestión del desarrollo de software teniendo un impacto significativo en las áreas técnicas. El objetivo del proceso de software es fundamental para facilitar la comunicación y coordinación durante el desarrollo de software, gestionar proyectos específicos y medir y mejorar la calidad de los productos de software desarrollados.

Una característica fundamental del proceso de software es que es un proceso centrado en las personas, en el que estas tienen el rol principal al ejecutar las actividades necesarias para cumplir los objetivos del proceso. Al ser ejecutado por personas, estas son propensas a cometer errores, omisiones y malinterpretaciones del proceso, lo cual establece discrepancias entre lo documentado (proceso estándar definido) y lo realmente ejecutado. Por tanto, el proceso requiere amplia cooperación y colaboración entre personas (el cual puede volverse más complejo si son equipos virtuales distribuidos) y debe ser bastante flexible para fomentar la libertad y creatividad de los diferentes equipos de desarrollo (en la práctica, las personas necesitan desviarse un poco de la descripción normativa incorporada en el modelo de proceso, pero el desafío es que no aparezcan inconsistencias en este).

El proceso de software ha sido reconocido por los investigadores de ingeniería de software como un área específica que requiere atención especial y esfuerzos focalizados de investigación para entender sus fundamentos, desarrollar modelos, técnicas, métodos y herramientas que luego facilitarán la definición de estándares que ayuden a gestionar el progreso de esta área.

Cabe resaltar que los conceptos presentados en cada capítulo de este libro tienen asociadas herramientas tecnológicas o prototipos tanto comerciales como académicos, presentes actualmente en la web, los cuales no han sido reseñados debido a su carácter volátil, que hace que sus enlaces de descarga y documentación se desactualicen o desaparezcan rápidamente.

El desarrollo de este libro ha pretendido dar a conocer a profesores, estudiantes, practicantes, desarrolladores de la industria e investigadores los conceptos fundamentales del proceso de software, procesos tradicionales, ágiles y estandarizados, su ciclo de vida desde su definición, modelado, medición, control, ejecución y mejora, y abordando cuestiones de infraestructura, reutilización, adaptaciones orientadas a pymes, procesos orientados a aplicaciones específicas, casos de éxito y tendencias de investigación en el área.

Los procesos tradicionales y ágiles siguen conviviendo actualmente, aunque el predominio de los ágiles hace que los modelos iterativos e incrementales predominen sobre los otros. Lo fundamental es que los tipos de modelos seleccionados e implementados estén guiados por valor conside-

rando su alineación con los objetivos de negocio, las mejoras en productividad y el retorno de la inversión.

Los estándares son fundamentales en todas las disciplinas y el proceso de software no es la excepción. Las iniciativas de estándares están dirigidas a la definición de estándares, métodos de evaluación y de mejora del proceso. La norma tradicional **ISO 9000** comenzó a aplicarse a empresas desarrolladoras de software, pero su ámbito general no se adaptaba bien a estas. Por tanto, surgieron estándares específicos del proceso de software, en los que se destacan organizaciones como la **ISO** o el Institute of Electrical and Electronics Engineers (**IEEE**) con sus normas **ISO 9000-3**, **ISO/IEC 12207**, **IEEE 1074** e **ISO/IEC 15288**. Sin embargo, existen otras propuestas como Métrica v.3, el Unified Process (**UP**) (con su variante Essential Unified Process (EssUP) y Software Engineering Method and Theory (**SEMAT**).

Los lenguajes de modelado se han aplicado para lograr una definición concisa, adecuada y eficaz del proceso de software por medio de diversas notaciones gráficas. Los modelos de procesos de software poseen elementos básicos, como actividades, productos, herramientas, reglas, roles y ejecutores junto con una sintaxis (conjunto de elementos legales) y una semántica (significado). Se destacan Business Process Model and Notation (**BPMN**), Software Process Engineering Metamodel (**SPEM**) y los diagramas de actividad Unified Modeling Language (**UML**) como los lenguajes más ampliamente difundidos para realizar modelos de procesos de software. Aunque otros lenguajes como Integration Definition for Function Modeling (**IDEF0**), Electronic Product Code (**EPC**) y las redes de Petri también pueden utilizarse.

Las métricas de software también han sido especializadas en el proceso. Se requiere contar con un proceso de medición que oriente la recolección de medidas, el establecimiento de métricas y la generación de indicadores para comprender y mejorar aspectos del proceso. El paradigma Goal Question Metric (**GQM**) es un buen punto de partida para el proceso de medición. Los responsables del proceso deben definir entidades y atributos susceptibles de ser medidos y recolectados durante el desarrollo de software. Estas entidades pueden ser objetos recibidos o usados, actividades y sus elementos, y objetos consumidos, mantenidos y producidos.

El control de proceso es otra actividad fundamental para conocer el estado actual del proceso que está basado en la recolección y el análisis de métricas. Generalmente, se aplican conceptos de control estadístico del proceso para evaluar la estabilidad y el rendimiento de un proceso.

Para ello, se pueden generar gráficos de control de variables y atributos. También es fundamental verificar si el proceso está cumpliendo con los plazos previstos durante la ejecución de un proyecto. La técnica de valor ganado y el seguimiento binario se pueden utilizar para evaluar el cumplimiento del proceso.

La mejora del proceso de software es un tema con mucho esfuerzo de investigación en que se destaca la generación de ciclos de mejora, modelos y metodologías. El modelo más ampliamente conocido y utilizado es **CMMI**, el cual se ha convertido en la referencia obligatoria y el “deber ser” para una empresa formal de desarrollo de software. **CMMI** cuenta con el modelo Standard **CMMI** Appraisal Method for Process Improvement (**SCAMPI**) como metodología de evaluación. Sin embargo, existen otros modelos con diferentes alcances como Personal Software Process (**PSP**) (a nivel individual), Team Software Process (**TSP**) (a nivel de equipo), Six Sigma, **ISO 33000**, entre otros, que pueden utilizarse según los objetivos de mejora del proceso.

La infraestructura es un requisito necesario, ya que los procesos, una vez han sido definidos o modificados, deben ser implementados, conocidos y realmente utilizados en las organizaciones. Para ello, se debe realizar un efectivo despliegue del proceso contando con una infraestructura adecuada basada en recursos humanos (con grupos especializados como Software Engineering Process Group [**SPEG**]) y financieros (evaluando el retorno de la inversión), tecnologías (por ejemplo, Process-centered Software Engineering Environment [**PSEE**], manuales de procesos, guías electrónicas de procesos, bibliotecas de activos de proceso y wikis) y formación.

Enfoques formales como **CMMI** pueden ser onerosos para las pymes desarrolladoras de software. Por tanto, en este tipo de empresas se han realizado investigaciones para adaptar modelos formales y “pesados”, y convertirlos en enfoques ligeros pero pragmáticos dirigidos a estándares, infraestructuras, modelos y técnicas de **SPI** en pymes desarrolladoras de software. Se destaca el estándar **ISO/IEC 29110**; sin embargo, existen otras propuestas como Moprosoft, Competisoft, **MPS.BR**, ITmark y Tutelkan, y modelos y métodos específicos como **OWPL**, **MARES**, y muchos más.

El reúso es otro aspecto fundamental que permite ahorrar esfuerzos y costos a partir de la reutilización de conocimiento. El uso de arquitecturas de procesos, líneas de proceso, componentes de proceso y patrones de proceso brindan una serie de recursos y tecnologías que permiten el reúso de conocimiento previo, aplicación de mejores prácticas y establecimiento de elementos de proceso que puedan aplicarse en futuros proyectos.

Los procesos definidos estándares deben adaptarse para desarrollar aplicaciones específicas. Cada aplicación requiere, llegado el caso, conocimiento y actividades especializadas. Por tanto, para cada tipo de aplicación informática, se han definido procesos de software específicos en áreas como el reconocimiento de patrones, internet de las cosas, *machine learning*, entre otras.

Los casos de éxito reportados por la industria del software sobre adaptación o creación de procesos de software pueden consultarse en internet. Comprenden historias bastante interesantes sobre problemas particulares que han tenido las organizaciones y cómo pudieron abordar soluciones basadas en el rediseño y la mejora de sus procesos de desarrollo de software para solucionarlos.

Por último, la ingeniería de procesos de software continúa estudiando e investigando nuevos conceptos que con el tiempo se convertirán en tendencias de investigación en el área. Se destacan como áreas de investigación relevantes para practicantes, profesionales e interesados en procesos de software la minería de procesos, la aplicación de técnicas de aprendizaje automático y aprendizaje profundo en el proceso de software y nuevos modelos de procesos de software.

El proceso de software es un área relevante para las fábricas de software actuales que compiten en un mercado dinámico. Es necesario y fundamental que las organizaciones conozcan los conceptos subyacentes del proceso de software, con el objetivo fundamental (que va más allá de obtener certificaciones de calidad y cumplir las exigencias de un mercado altamente competitivo) de definir y explicitar el conocimiento que requieren y utilizan diariamente para cumplir su razón de existir: construir productos de software que cumplan los requisitos de sus clientes. Espero que este libro haya contribuido a esa visión.

► Índice analítico

A

Adaptive Software Development 134, 142, 168–169, 171
aprendizaje automático 25, 552, 563–565, 570, 587, 601–602, 609–612, 630, 641
aprendizaje profundo 25, 552, 601–602, 609, 612, 614, 630–631, 641
área(s) de conocimiento 204–209, 215, 553
arquitectura de proceso software 251–523, 544
arquitectura dirigida por modelos 617–618

B

base de datos 163, 356, 383, 394, 399, 427, 555–557, 572
bibliotecas de activos de procesos 423
blockchain 29, 86, 559–560, 572–575

C

calidad 24, 33, 36, 40, 42, 46–47, 49, 51, 57, 61, 66, 69, 72–73, 75, 79, 81–82, 86, 92–93, 105–106, 108, 111, 115, 129–130, 135, 140, 143, 147, 149, 152, 154, 160, 165, 173–175, 178–182, 185, 187, 196, 208–209, 214, 222, 269–270, 272, 276, 279–280, 287–288, 291–292, 297–298, 301, 303, 305, 308–311, 314–315, 317, 329, 345–346, 348, 353–354, 359–360, 362, 366, 368–371, 373, 376, 378–380, 390, 397, 401, 418, 434, 442, 444, 447, 454, 457, 459–462, 466, 468, 471, 476–479, 481–484, 498, 502, 513, 515, 518, 528, 539, 541–542, 549, 555, 568, 582–584, 586, 592–593, 598, 602–607, 616–617, 619–623, 627, 630, 637, 641
cascada, modelo 37, 63–66, 74, 82–86, 182, 229, 508, 559, 578–579, 590, 597
ciclo de vida 24, 27–28, 32, 36, 51–52, 61–63, 66, 75, 80–83, 85, 92, 96, 116, 138–139, 148, 154, 156, 181–182, 184–185, 188–190, 192–193, 196, 198–200, 209–211, 213–215, 221, 233, 261, 336, 414,

442, 444, 478, 492, 504, 508, 513–515, 521, 535, 544, 562, 566, 569, 572, 578, 580, 586, 604–605, 617, 638
código 39–40, 44, 63, 79, 93, 105–106, 108, 110–113, 115, 129–130, 137, 142, 144–148, 160, 163, 186–187, 195, 260, 271–274, 282–283, 295, 297–298, 303, 318, 324, 339, 370–372, 389, 402, 434, 484, 544, 557–559, 563, 565, 583, 585, 589, 596, 611, 614, 620, 627
control de calidad 47, 81, 181, 301, 457, 619–622
control estadístico 28, 69, 309–310, 312–314, 322, 341, 401, 577, 585, 637, 639
crystal, métodos 98, 134, 136–137, 146, 165, 168, 625

D

DevOps 159, 414, 559, 620–623, 627–631, 633–635
diagramas de actividad 27, 231, 248–250, 252, 259, 263–264, 526, 639
Disciplined Agile Delivery 154, 167
desarrollo de aplicaciones 137, 421, 477, 523, 553, 556, 559, 561, 563, 566, 568–569, 572–573, 618, 623, 631
desarrollo global de software 615–617, 631, 636
desarrollo de software 24, 26–28, 31–37, 39, 41–42, 44–46, 49, 54–56, 61–63, 65, 67, 72, 77, 80–82, 84–85, 91, 93, 96–100, 105, 108, 134, 144–145, 159, 165, 168, 173, 175–177, 180, 182, 186–189, 196, 203–204, 206, 210, 214–215, 221, 227–229, 232–233, 266, 270, 284, 286–287, 305–306, 309, 336, 339, 345, 348, 365, 368, 376, 381, 389, 397, 401, 414, 416, 420–421, 429, 441–443, 446, 449, 455, 460, 467, 478, 481–482, 485, 489–490, 501–502, 508, 511–512, 514, 537, 539, 545, 552, 560, 570, 572–573, 579, 584, 590, 597, 601–602, 607–608, 615–617, 620, 623–627, 631, 637, 639–641

Dynamic System Development Method 134, 172

E

equipo de desarrollo 53, 55, 83, 86, 91, 94, 96, 102–105, 107–111, 115, 117–119, 122–127, 129–131, 133, 143, 148, 164–166, 176, 213, 261, 339, 442, 484, 512, 533–534, 567, 584, 587, 616, 622, 627, 631, 643
equipo scrum 117–118, 125, 128–129, 131, 166, 416, 581
EPC 231, 254–256, 259, 263, 265, 639
espiral, modelo en 38, 71, 74–77, 80, 82, 85–86, 229, 508, 590, 643
EssUP 193, 202, 215–216, 218, 639

F

Feature-Driven Development 98, 134, 168, 171
flujo de control 248, 250, 254, 256–257, 607–608

G

gestión del conocimiento 29, 120, 148, 377, 408, 436, 442, 501, 537–540, 545–546
GQM 28, 286, 305–306, 527, 639
gráfico(s) de control 28, 222, 309, 311, 314–315, 317, 319–325, 327, 336–337, 341–342, 390, 401, 584, 597, 640

I

IDEAL, modelo 57, 73, 76, 135, 144, 355–357, 376, 401, 403–404, 468, 470, 495
IDEF0 231, 251–254, 259, 263–264, 266, 639
ingeniería dirigida por modelos 615, 617
ingeniería de procesos de software 26, 36, 523, 541, 551–552, 601, 637, 641
ingeniería de software 23–25, 29, 34, 36–40, 43, 52, 56, 73, 180, 183, 186, 203, 206–209, 215–216, 225, 259, 305, 307, 349, 356, 400, 409, 420, 440, 446, 449, 454, 457, 477, 502, 508, 516, 563, 578, 602, 608, 615, 637–638
ISO 33000 378–379, 403–404, 640

L

Lean Software Development 99, 134, 171
lenguaje de modelado 221, 225, 230–231, 248, 251, 258, 263, 415, 491, 556
lenguaje de programación 71, 142, 271, 372, 536, 560, 619
Large Scale Scrume (LeSS) 150, 156–159, 165
límites de control 310, 315–317, 319, 322–323, 325–328, 336–337, 343
líneas de procesos software 518

M

machine learning 29, 551, 563–564, 573–574, 609, 631, 634–636, 641
manifiesto ágil 32, 98, 101–102, 109, 164–166
mejora continua 107, 128, 134, 179–180, 222, 309, 349–350, 388–389, 401, 456–457, 477, 481, 551, 585
mejora de(l) proceso 28–29, 38, 49–50, 52, 128, 174, 185, 213, 222–223, 276, 291, 294, 296, 345–346, 348, 351–353, 355, 368–369, 372, 378–379, 390, 400–402, 410, 419, 425, 430, 439, 443, 449, 460–461, 468, 470, 485, 522, 595–596, 630, 639–640
método(s) de evaluación 174, 213, 490, 639
método(s) de medición 275, 282
métrica v3 239, 253, 264
minería de datos 29, 551, 554, 568–570, 573, 609, 630
modelo CMMI 357–358, 361, 365, 387–388, 401–402, 440, 457, 529–530
modelo de ciclo de vida 32, 61–62, 80, 83, 181, 189, 193, 508, 515
modelo de proceso(s) 63, 176, 299–300, 349, 449, 464, 485, 497, 568–569, 573
modelo IDEAL 76, 355–357, 401, 468, 470
modelo MPS 454–456
modelo SPEM 235, 262

N

nodo de control 249

P

patrones de proceso 28, 408, 516, 523–525, 527, 542, 545, 640

- PSP 343, 368–372, 401, 403–406, 640
 - proceso ágil 32, 91, 93–94, 96, 108, 145, 160, 162, 165–166, 210, 302, 582, 597
 - proceso de desarrollo de software 46, 55, 91, 93, 98, 173, 232, 270, 306, 368, 401, 420, 442, 446, 460, 579, 623
 - proceso unificado 159, 193, 196–200, 210–211, 215–216, 219, 236, 545, 608
 - proceso(s) de software 29, 31–34, 36–42, 44–45, 47, 49–50, 52, 55–57, 62, 116, 173, 176–177, 189, 208, 210, 221–222, 225, 227–228, 230–232, 238, 257–261, 263–265, 269, 291, 294, 298–299, 306, 309–311, 314, 317, 336, 346, 348, 350–352, 354, 361, 376, 378, 400–401, 407–409, 414, 416–417, 420, 426–427, 429–430, 432–436, 439–440, 443, 447, 453–456, 459–463, 472–473, 476, 479, 481, 485, 489, 491–492, 501–502, 507, 513, 516, 518, 520–523, 529, 531, 537–539, 541, 544, 546, 548, 551–553, 559, 570, 577–579, 584, 595, 597–598, 601–603, 607, 609, 614–616, 625–626, 631, 637, 639, 641
 - producto 24, 31, 33, 35–36, 39–40, 46–49, 53, 56–58, 61–62, 64–65, 68–69, 71–73, 76–79, 82–83, 85, 92–94, 96, 99, 103, 105–106, 110–113, 117–121, 123, 128–131, 133–135, 138, 140, 142, 145, 148, 151–152, 154, 156–158, 165–166, 173–176, 178, 180–184, 186–189, 191–192, 198–200, 202–203, 214, 229–230, 235, 270–273, 276, 278, 281, 283, 285, 288, 290, 296, 298, 304–306, 310, 312, 316–317, 322, 324, 327, 329–331, 346–347, 359–360, 367–368, 370, 372, 377, 388, 391, 393, 396, 418, 422–424, 434, 442, 447, 460, 468, 477–478, 482, 484, 491, 502, 506, 511, 513, 518, 524, 529, 542, 558, 561–564, 570, 578, 580–582, 585–587, 592–593, 603–605, 621–622, 625, 627, 629–630, 637
 - producto de trabajo 235, 317, 330, 422–424, 491, 582
 - product owner 117–119, 123–125, 127, 129, 154, 165
 - producto(s) de software 39, 42, 52, 106, 166, 178, 183, 185, 187, 227, 349, 378, 420–421, 502, 513–514, 518, 541, 544, 552, 562, 607, 637, 641
 - programación extrema 27, 32, 38, 91, 108, 482, 621, 625
 - prototipos 45, 71–74, 80, 85–86, 112, 135, 140, 156, 535, 623, 625, 638
 - proyecto 27–28, 32, 34, 36, 40–41, 45–49, 52–57, 61–64, 67–68, 75–76, 78–85, 91, 94, 96–97, 99–106, 110, 112–115, 118, 121, 123, 126, 133, 135–136, 138, 140, 143, 146–150, 160–162, 164–166, 176, 185, 187, 189, 191–193, 196, 200, 204–205, 211, 213–214, 227–229, 234–235, 253–254, 261, 271, 273–274, 283–286, 304–306, 325, 329–330, 332, 335, 339, 341–342, 346, 348–349, 351, 354–355, 359, 367–368, 370–373, 375–376, 379, 381, 384–385, 387–388, 397–402, 408, 415, 418, 423, 426, 442, 444, 446–447, 452, 460, 466, 468, 470–471, 474, 476–477, 482–484, 489, 501–502, 508, 513, 515, 517–518, 520–522, 527–537, 540–542, 545, 566, 568, 578–583, 596–597, 604, 606, 608, 611, 616–617, 620, 628, 640
- ## R
- redes de Petri 231, 256–257, 263, 639
 - requisito(s) del cliente 47, 86, 179, 201, 290, 376–377, 392, 399, 479
 - resultado 68, 76, 97, 110, 124, 143, 192–193, 199, 211, 229, 238, 242, 270, 275, 278, 280–281, 286, 328, 398, 423, 430, 443, 451, 453, 487–488, 491, 555, 568, 580–581, 590, 610
 - reúso de software 408, 501–502, 506, 513, 527
- ## S
- satisfacción del cliente 46, 62, 145, 178, 180, 291, 304, 306, 348, 365, 377, 389, 444, 447, 484, 536, 595–597, 628
 - Scaled Agile Framework 152, 157, 586
 - SCAMPI 384–387, 401–402, 406, 640, 645
 - scrum 27, 32, 91, 98, 100, 107–108, 116–119, 121, 123–134, 146, 150–152, 154, 156, 158–159, 162, 164–167, 169–172, 247, 264, 268, 302, 306–307, 342, 349,

416, 508, 516, 539, 545, 560, 579–582,
585–587, 589, 597–599, 625, 628, 633
scrum de scrums 150–151, 158–159
Scrum Master 100, 117–118, 124–125,
127–128, 131, 151, 154, 159, 165–166,
580–581
SEMAT 193, 203–205, 215–218, 639
sistema 36, 39, 44, 47–49, 53, 55, 62–64,
69, 71, 73, 77, 79, 86, 94–96, 99, 102–
103, 106, 109–112, 116, 120, 136–137,
140–142, 145, 152, 165, 176, 178–183,
187–198, 200–205, 210, 214–216, 226–
227, 230, 232, 239, 248, 253, 261, 263–
264, 270–271, 284–285, 289, 302, 324,
339, 347, 354, 396, 413, 477, 484, 491,
504, 507, 512, 514–515, 520, 527, 531,
533–534, 536–537, 555, 560, 567, 572,
584–585, 591–592, 604, 606, 618–620,
623, 625, 629
sistema de calidad 178, 181, 477
sistema de gestión de calidad 178

sistema de información 86, 195, 214, 216,
239, 253, 264, 534
sistema(s) de software 41, 83, 515
SPEM 27, 231–238, 258–260, 262–264,
266–268, 298–300, 306–307, 348, 625,
639
sprint 117–119, 121–128, 131, 146–147,
156–157, 162–163, 165, 302–304, 306,
342, 349, 517, 581–582, 587
SwEBok 24, 193, 207–209, 215–217, 637

T

Test Driven Development 144, 168
TSP 365–367, 401

V

valor ganado 222, 284, 304, 330, 332–335,
341, 370, 640

W

wikis 419, 426–427, 434–435, 583, 640

Ciencias
de **Gestión**

Ingeniería de procesos de software ◀

Hace parte de la colección Ciencias de Gestión.

Se editó y diagramó en la Editorial
Universidad Nacional de Colombia,
en octubre de 2024.

Bogotá, D. C., Colombia.

En su composición se utilizaron caracteres
ITC Berkeley Oldstyle Std 11/14 puntos.

Otros títulos de esta colección

- ▶ *Capacidades dinámicas en grupos de investigación y su impacto en ciudades inteligentes*
Leonardo Bermón Angarita
- ▶ *Modelos y marcos de referencia de gestión de riesgos en entornos digitales*
Francisco Javier Valencia Duque,
Carlos Eduardo Marulanda Echeverry y
Marcelo López Trujillo
- ▶ *La naturaleza del conocimiento tecnológico*
Daian Tatiana Flórez Quintero
- ▶ *Emprendimiento corporativo y su relación con la innovación organizacional*
Pablo Felipe Marín Cardona
- ▶ *Impacto de los proyectos culturales
El caso de Villamaría (Caldas, Colombia)*
Winston M. Licona Calpe y
Natalia Osorio Grajales

Esta publicación es una guía práctica que explora principios, técnicas, estándares, aplicaciones, ejemplos y contribuciones realizadas en el área de procesos de software. La ingeniería de procesos de software aborda el desarrollo de software desde un enfoque orientado a procesos como elementos fundamentales para construir con éxito productos de software. El proceso de software se puede considerar como una serie de actividades, procedimientos, roles y artefactos relacionados con el fin de entregar productos de alta calidad a tiempo y dentro del presupuesto establecido.

Los practicantes, docentes, investigadores e interesados que deseen conocer y profundizar en este tipo de procesos encontrarán una fuente de consulta y punto de referencia inicial al fascinante mundo del desarrollo de software desde una perspectiva basada en estándares y referentes internacionales de la industria y la academia.

Este libro tiene como objetivos: a) servir como texto que oriente a profesores, estudiantes, practicantes e interesados de la ingeniería de software en conocer aspectos relacionados con el proceso de software; b) lograr que los lectores comprendan la definición, modelado, medición, ejecución, control y mejora de los procesos de software; c) entender el estado actual de la investigación en el área de los procesos software con el fin de establecer nuevas rutas y preguntas de investigación que al ser resueltas fortalezcan los conocimientos del área, y d) ofrecer a los lectores los conocimientos necesarios para aplicar con éxito un enfoque de procesos en organizaciones desarrolladoras de software o en proyectos de desarrollo de software.