

Lucjan Stapp · Adam Roman  
Michael Pilaeten

ISTQB®

Probador certificado  
Base  
Nivel

Un programa de estudios de guía de autoestudio v4.0

## Nivel básico de probador certificado ISTQB®

Lucjan Stapp • Adam Roman • Michaël Pilaeten

**ISTQB®**  
Probador certificado  
**Base**  
**Nivel**

Un programa de estudios de guía de autoestudio v4.0



**Springer**

Lucjan Stapp  
Varsovia, Polonia

Adán Romano  
Universidad Jagellónica  
Cracovia, Polonia

Michael Pilaeten  
Londerzeel, Bélgica

ISBN 978-3-031-42766-4 <https://doi.org/10.1007/978-3-031-42767-1>

ISBN 978-3-031-42767-1 (libro electrónico)

Traducción de la edición en polaco: "Certyfikowany Tester ISTQB. Przygotowanie do egzaminu wedlug sylabusu w wersji 4.0" por Lucjan Stapp et al., © Helion.pl sp. z oo, Gliwice, Polonia (<https://helion.pl/>) - Derechos polacos únicamente, todos los demás derechos con los autores 2023. Publicado por Helion.pl sp. z oo, Gliwice, Polonia (<https://helion.pl/>). Reservados todos los derechos.

© El(es) editor(es) (si corresponde) y el(es) autor(es), bajo licencia exclusiva para Springer Nature Switzerland AG 2024 Este trabajo está

sujeto a derechos de autor. Todos los derechos son licenciados única y exclusivamente por el Editor, ya sea sobre la totalidad o parte del material, específicamente los derechos de reimpresión, reutilización de ilustraciones, recitación, radiodifusión, reproducción en microfilmes o en cualquier otro medio físico, y transmisión o información, almacenamiento y recuperación, adaptación electrónica, software de computadora o mediante metodología similar o diferente ahora conocida o desarrollada en el futuro.

El uso de nombres descriptivos generales, nombres registrados, marcas comerciales, marcas de servicio, etc. en esta publicación no implica, incluso en ausencia de una declaración específica, que dichos nombres estén exentos de las leyes y regulaciones protectoras pertinentes y, por lo tanto, libres para uso general. usar.

El editor, los autores y los editores pueden asumir con seguridad que los consejos y la información de este libro se consideran verdaderos y precisos en la fecha de publicación. Ni el editor ni los autores ni los editores ofrecen garantía, expresa o implícita, con respecto al material contenido en este documento o por cualquier error u omisión que se haya cometido. El editor se mantiene neutral con respecto a reclamos jurisdiccionales en mapas publicados y afiliaciones institucionales.

Ilustración de portada: © trahko / Stock.adobe.com

Este aviso legal de Springer es publicado por la empresa registrada Springer Nature Switzerland AG.  
La dirección registrada de la empresa es: Gewerbestrasse 11, 6330 Cham, Suiza.

El papel de este producto es reciclable.

# Prefacio

## Propósito de este libro

Este libro está dirigido a quienes se preparan para el examen ISTQB® Certified Tester—Foundation Level basado en el programa de estudios Foundation Level (versión 4.0) publicado en 2023.

Nuestro objetivo era proporcionar a los candidatos conocimientos fiables basados en este documento. Sabemos por experiencia que se puede encontrar mucha información sobre los programas y exámenes de ISTQB® en Internet, pero lamentablemente gran parte de ella es de mala calidad. Incluso sucede que los materiales que se encuentran en la Web contienen errores graves. Además, debido a los importantes cambios que se han producido en el programa de estudios en comparación con la versión anterior (3.1.1) publicada en 2018, la cantidad de material disponible para los candidatos basado en el nuevo programa de estudios es todavía pequeña.

Este libro amplía y detalla muchas cuestiones que se describen en el propio programa de estudios de manera superficial o general. De acuerdo con las pautas de ISTQB® para la capacitación basada en programas de estudios, se debe proporcionar un ejercicio para cada objetivo de aprendizaje en el nivel K3 y se debe proporcionar un ejemplo práctico para cada objetivo en el nivel K2 o K3.1 Para satisfacer estos requisitos, Hemos preparado ejercicios y ejemplos para todos los objetivos de aprendizaje en estos niveles. Además, para cada objetivo de aprendizaje, presentamos uno o más preguntas de examen de muestra similares a las que el candidato verá en el examen. Esto hace que el libro sea una excelente ayuda para estudiar, prepararse para el examen y verificar los conocimientos adquiridos.

## Estructura del libro

El libro consta de cuatro partes principales.

---

<sup>1</sup> A continuación se proporciona más información sobre los objetivos de aprendizaje y los niveles K.

## Parte I: Certificado, plan de estudios y examen de nivel básico

La Parte I proporciona información oficial sobre el contenido y la estructura del programa de estudios y el examen ISTQB® Certified Tester—Foundation Level. También analiza la estructura de certificación ISTQB®. Esta sección también explica los conceptos técnicos básicos en los que se basan el programa de estudios y la estructura del examen. Explicamos cuáles son los objetivos de aprendizaje y los niveles K y cuáles son las reglas para construir y administrar el examen real. Vale la pena familiarizarse con estos temas, ya que comprenderlos le ayudará a prepararse mucho mejor para el examen.

## Parte II: Discusión del contenido del programa de estudios

La Parte II es la parte principal del libro de texto. Aquí, analizamos en detalle todo el contenido y los objetivos de aprendizaje del programa de estudios de nivel básico. Esta parte consta de seis capítulos, correspondientes a los seis capítulos del programa de estudios. Cada objetivo de aprendizaje en el nivel K2 o K3 se ilustra con un ejemplo práctico, y cada objetivo de aprendizaje en el nivel K3 se ilustra con un ejercicio práctico.

Al comienzo de cada capítulo, se dan definiciones de las palabras clave aplicables al capítulo. Cada palabra clave, en el lugar de su primer uso relevante en el texto, está marcada en negrita y con un icono de libro.



Al final de cada capítulo, el lector encontrará ejemplos de preguntas de examen que cubren todos los objetivos de aprendizaje incluidos en este capítulo. El libro contiene 70 preguntas de examen de muestra originales que cubren todos los objetivos de aprendizaje, así como 14 ejercicios prácticos correspondientes a los objetivos de aprendizaje del nivel K3. Estas preguntas y ejercicios no forman parte de los materiales oficiales de ISTQB®, pero los autores los construyen utilizando los principios y reglas que se aplican a su creación para los exámenes reales. Así, son material adicional para los lectores, permitiéndoles verificar sus conocimientos luego de la lectura de cada capítulo y comprender mejor el material presentado.

### Material opcional El

texto en el cuadro indica material opcional. Se relaciona con el contenido del programa de estudios pero va más allá y no está sujeto a examen. Es "para aquellos que tienen curiosidad".

Las secciones con títulos marcados con un asterisco (\*) son opcionales. Cubren el material que era obligatorio para el examen según la versión anterior del programa de estudios. Decidimos dejar estos capítulos en el libro debido a su importancia y aplicación práctica. El lector que utilice el libro de texto sólo para estudiar para el examen puede saltarse estos capítulos mientras lee. Estas seccionesopcionales son:

Sección 3.2.6—Técnicas de revisión  
Sección 4.2.5—Pruebas de casos de uso

## Parte III: Respuestas a preguntas y ejercicios

En la Parte III, proporcionamos soluciones a todos los ejemplos de preguntas y ejercicios del examen que aparecen en la Parte II del libro. Las soluciones no se limitan a dar las respuestas correctas sino que también incluyen sus justificaciones. Ayudarán al lector a comprender mejor cómo se crean las preguntas del examen real y a prepararse mejor para resolverlas durante el examen real.

## Parte IV: Examen oficial de muestra y preguntas adicionales

La última parte, la Parte IV del libro de texto, contiene el ejemplo oficial del examen ISTQB® para la certificación Foundation Level, preguntas adicionales que cubren objetivos de aprendizaje no cubiertos en el examen e información sobre las respuestas correctas y las justificaciones de esas respuestas.

Por lo tanto, el libro está estructurado de tal manera que todos los aspectos importantes y útiles La información está en un solo lugar:

- Estructura y reglas del examen.
- El contenido discutido en el programa de estudios con su discusión integral y ejemplos
- Definiciones de términos cuyo conocimiento es obligatorio para el examen. • Ejemplos de preguntas y ejercicios originales del examen, con respuestas correctas y sus justificación
- Ejemplo de examen ISTQB® con respuestas correctas y su justificación

Esperamos que el material presentado en esta publicación ayude a todos aquellos interesados en obtener la certificación ISTQB® Certified Tester—Foundation Level.

Varsovia, Polonia  
Cracovia, Polonia  
Londerzeel, Bélgica

Lucjan Stapp  
Adán Romano  
Michael Pilaten

# Contenido

Certificación de la Parte I, plan de estudios y examen de nivel básico	
Certificado de nivel básico .....	3
Historia del Certificado de Nivel Básico .....	3
Trayectorias profesionales para probadores .....	
Público objetivo .....	5
Objetivos del Sistema Internacional de Cualificaciones .....	5
Programa de estudios del nivel básico .....	7
Resultados comerciales .....	7
Objetivos de aprendizaje y niveles K. ....	7
Requisitos para los candidatos .....	9
Referencias a normas y estándares .....	9
Actualización continua.: 10 .....	
Notas de la versión para el programa de estudios de nivel básico v4.0.: 10- .....	
Contenido del Plan de Estudios. 11- .....	
Capítulo 1. Fundamentos de las pruebas. 11 .....	
Capítulo 2. Pruebas a lo largo del ciclo de vida del desarrollo de software... 12	
Capítulo 3. Pruebas estáticas. 13- .....	
Capítulo 4. Análisis y diseño de pruebas... 13 .....	
Capítulo 5. Gestión de las actividades de prueba. 14 .....	
Capítulo 6. Herramientas de prueba .....	15
Examen de nivel básico.. . ..	17
Estructura de las reglas .....	17
del examen. . ..	17
Distribución de Preguntas. ....	18
Consejos: antes y durante el examen .....	20

## Parte II El contenido del programa de estudios

<b>Capítulo 1 Fundamentos de las pruebas.</b>	25
<b>1.1 ¿Qué son las pruebas?</b>	27
1.1.1 Objetivos de la prueba.	28
1.1.2 Pruebas y depuración.	29
<b>1.2 ¿Por qué son necesarias las pruebas?</b>	31
1.2.1 Contribución de las pruebas al éxito.	34
1.2.2 Pruebas y garantía de calidad (QA).	35
1.2.3 Errores, defectos, fallas y causas fundamentales.	36
<b>1.3 Principios de prueba.</b>	41
<b>1.4 Actividades de prueba, software de prueba y funciones de prueba</b>	47
1.4.1 Actividades y tareas de prueba	47
1.4.2 Proceso de prueba en contexto.	53
1.4.3 Software de prueba.	54
1.4.4 Trazabilidad entre la base de prueba y el software de prueba.	60
1.4.5 Funciones en las pruebas.	61
<b>1.5 Habilidades Esenciales y Buenas Prácticas en Pruebas.</b>	64
1.5.1 Habilidades genéricas requeridas para las pruebas.	64
1.5.2 Enfoque de equipo completo.	66
1.5.3 Independencia de las pruebas.	67
<b>Preguntas de muestra.</b>	69
<b>Capítulo 2 Pruebas a lo largo del ciclo de vida del desarrollo de software.</b>	75
<b>2.1 Pruebas en el contexto de un ciclo de desarrollo de software.</b>	76
2.1.1 Impacto del ciclo de vida del desarrollo de software en las pruebas.	77
2.1.2 Ciclo de vida del desarrollo de software y buenas prácticas de prueba	87
2.1.3 Las pruebas como impulsor del desarrollo de software.	87
2.1.4 DevOps y pruebas.	92
2.1.5 Enfoque de desplazamiento a la izquierda.	96
2.1.6 Retrospectivas y Mejora de Procesos.	97
<b>2.2 Niveles de prueba y tipos de prueba.</b>	99
2.2.1 Niveles de prueba.	99
2.2.2 Tipos de prueba.	115
2.2.3 Pruebas de confirmación y pruebas de regresión.	124
<b>2.3 Pruebas de mantenimiento.</b>	127
<b>Preguntas de muestra.</b>	130
<b>Capítulo 3 Pruebas estáticas.</b>	133
<b>3.1 Conceptos básicos de las pruebas estáticas</b>	134
3.1.1 Productos de trabajo examinables mediante pruebas estáticas.	135
3.1.2 Valor de las pruebas estáticas.	136
3.1.3 Diferencias entre pruebas estáticas y pruebas dinámicas.	140
<b>3.2 Proceso de retroalimentación y revisión</b>	143
3.2.1 Beneficios de la retroalimentación temprana y frecuente de las partes interesadas	143
3.2.2 Actividades del proceso de revisión.	144

Contenido	xii
3.2.3 Funciones y responsabilidades en las revisiones. . . . .	150
3.2.4 Tipos de revisión. . . . .	151
3.2.5 Factores de éxito de las revisiones. . . . .	160
3.2.6 (*) Técnicas de Revisión. . . . .	162
Preguntas de muestra. . . . .	165
<b>Capítulo 4 Análisis y diseño de pruebas. . . . .</b>	<b>169</b>
4.1 Descripción general de las técnicas de prueba. . . . .	171
4.2 Técnicas de prueba de caja negra. . . . .	176
4.2.1 Partición de equivalencia (EP) . . . . .	177
4.2.2 Análisis de valor en la frontera (BVA). . . . .	187
4.2.3 Pruebas de la tabla de decisiones. . . . .	194
4.2.4 Pruebas de transición estatal. . . . .	199
4.2.5 (*) Pruebas de casos de uso. . . . .	208
4.3 Técnicas de prueba de caja blanca. . . . .	211
4.3.1 Pruebas de declaraciones y cobertura de declaraciones. . . . .	212
4.3.2 Pruebas de sucursales y cobertura de sucursales. . . . .	214
4.3.3 El valor de las pruebas de caja blanca. . . . .	217
4.4 Técnicas de prueba basadas en la experiencia. . . . .	219
4.4.1 Error al adivinar. . . . .	220
4.4.2 Pruebas exploratorias. . . . .	223
4.4.3 Pruebas basadas en listas de verificación. . . . .	226
4.5 Enfoques de prueba basados en la colaboración. . . . .	229
4.5.1 Escritura colaborativa de historias de usuario. . . . .	229
4.5.2 Criterios de aceptación. . . . .	232
4.5.3 Desarrollo basado en pruebas de aceptación (ATDD). . . . .	234
Preguntas de muestra. . . . .	237
Ejercicios . . . . .	245
<b>Capítulo 5 Gestión de las actividades de prueba. . . . .</b>	<b>251</b>
5.1 Planificación de pruebas. . . . .	252
5.1.1 Propósito y contenido de un plan de prueba . . . . .	253
5.1.2 Contribución del probador a la planificación de iteraciones y lanzamientos: . . . . .	257
5.1.3 Criterios de Entrada y Criterios de Salida. . . . .	258
5.1.4 Técnicas de estimación. . . . .	259
5.1.5 Priorización de casos de prueba . . . . .	268
5.1.6 Pirámide de pruebas. . . . .	275
5.1.7 Prueba de cuadrantes. . . . .	276
5.2 Gestión de Riesgos. . . . .	277
5.2.1 Definición de riesgos y atributos de riesgo . . . . .	279
5.2.2 Riesgos del proyecto y riesgos del producto . . . . .	279
5.2.3 Análisis de riesgos del producto. . . . .	281
5.2.4 Control de riesgos del producto. . . . .	284
5.3 Monitoreo de pruebas, control de pruebas y finalización de pruebas. . . . .	286

5.3.1 Métricas utilizadas en las pruebas. . . . .	287
5.3.2 Propósito, contenido y destinatario de los informes de prueba. . . . .	288
5.3.3 Comunicación del estado de las pruebas. . . . .	291
5.4 Gestión de la configuración. . . . .	292
5.5 Gestión de defectos. . . . .	294
Preguntas de muestra. . . . .	296
Ejercicios para el Capítulo 5. . . . .	302
Capítulo 6 Herramientas de prueba. . . . .	307
6.1 Soporte de herramientas para pruebas. . . . .	307
6.2 Beneficios y riesgos de las preguntas de ejemplo sobre automatización de pruebas. . . . .	309
. . . . .	310
<b>Parte III Respuestas a preguntas y ejercicios</b>	
<b>Respuestas a preguntas de muestra. . . . .</b>	<b>313</b>
<b>Respuestas a las preguntas del cap. 1 . . . . .</b>	<b>313</b>
<b>Respuestas a las preguntas del cap. 2 . . . . .</b>	<b>317</b>
<b>Respuestas a las preguntas del cap. 3 . . . . .</b>	<b>321</b>
<b>Respuestas a las preguntas del cap. 4 . . . . .</b>	<b>323</b>
<b>respuestas a las preguntas del cap. 5 . . . . .</b>	<b>332</b>
<b>Respuestas a las preguntas del cap. 6 . . . . .</b>	<b>337</b>
<b>Soluciones a los ejercicios. . . . .</b>	<b>339</b>
<b>Soluciones a los ejercicios del cap. 4 . . . . .</b>	<b>339</b>
<b>Soluciones a los ejercicios del cap. 5 . . . . .</b>	<b>349</b>
<b>Parte IV Examen oficial de muestra</b>	
<b>Conjunto de exámenes A . . . . .</b>	<b>355</b>
<b>Preguntas de muestra adicionales. . . . .</b>	<b>369</b>
<b>Conjunto de exámenes A: Respuestas. . . . .</b>	<b>379</b>
<b>Preguntas de muestra adicionales: respuestas. . . . .</b>	<b>391</b>
<b>Referencias. . . . .</b>	<b>399</b>
<b>Índice. . . . .</b>	<b>403</b>

## Acerca de los autores<sup>2</sup>

Lucjan Stapp, PhD, es un investigador jubilado y profesor de la Universidad Tecnológica de Varsovia, donde durante muchos años impartió conferencias y seminarios sobre pruebas de software y control de calidad. Es autor de más de 40 publicaciones, incluidas 12 sobre diversos problemas relacionados con las pruebas y el control de calidad. Como tester, en su carrera profesional, pasó de tester junior a líder del equipo de pruebas en más de una docena de proyectos. Ha desempeñado el papel de coorganizador y orador en muchas conferencias de pruebas (incluida TestWarez, la conferencia de pruebas más grande de Polonia). Stapp es miembro fundador de la Information Systems Quality Association ([www.sjsi.org](http://www.sjsi.org)), actualmente su vicepresidente. También es un tester certificado (incluidos ISTQB® CTAL-TM, CTAL-TA, Agile Tester, Acceptance Tester).

Adam Roman, PhD, DSc, es profesor de informática y miembro de investigación y docencia en el Instituto de Ciencias de la Computación y Matemáticas Informáticas de la Universidad Jagellónica, donde ha impartido conferencias y seminarios sobre pruebas de software y garantía de calidad durante muchos años. Dirige el Departamento de Ingeniería de Software y es cofundador del programa de posgrado "Pruebas de software" de la Universidad Jagellónica. Sus intereses de investigación incluyen investigación sobre medición de software, modelos de predicción de defectos y técnicas efectivas de diseño de pruebas. Como parte del Comité Polaco de Normalización, colaboró en el estándar internacional de pruebas de software ISO/IEEE 29119. Roman es autor de las monografías *Testing and Software Quality: Models, Techniques, Tools, Thinking-Driven Testing* y *A Study Guide to the ISTQB® Foundation Level 2018 Syllabus: Test Techniques and Sample Mock Exams*, así como de muchas publicaciones científicas y populares en el campo de las pruebas de software. Ha desempeñado el papel de orador en muchos congresos polacos y

---

<sup>2</sup>Los tres autores son expertos en pruebas de software. Son coautores del programa de estudios Foundation Level versión 4.0, así como de otros programas de estudios de ISTQB®. También tienen experiencia práctica en la redacción de preguntas de exámenes.

conferencias de pruebas internacionales (incluidas EuroSTAR, TestWell, TestingCup y TestWarez). Posee varias certificaciones, entre ellas ASQ Certified Software Quality Engineer, ISTQB® Full Advanced Level y ISTQB® Expert Level—Mejora del proceso de prueba. Roman es miembro de la Asociación de Calidad de Sistemas de Información ([www.sjsi.org](http://www.sjsi.org)).

Michael Pilaeten. Romper el sistema, ayudar a reconstruirlo y brindar asesoramiento y orientación sobre cómo evitar problemas. Así es Michaël en pocas palabras. Con casi 20 años de experiencia en consultoría de pruebas en una variedad de entornos, ha visto lo mejor (y lo peor) en el desarrollo de software. En su función actual como Gerente de Aprendizaje y Desarrollo, es responsable de guiar a sus consultores, socios y clientes en su camino personal y profesional hacia la calidad y la excelencia. Es el presidente del grupo de trabajo Agile de ISTQB y propietario del producto del programa de estudios ISTQB® CTFL 4.0. Además, es miembro de la BNTQB (Junta de Cualificaciones de Pruebas de Bélgica y Países Bajos), una formación acreditada para la mayoría de las formaciones ISTQB® e IREB, y orador principal internacional y facilitador de talleres

# lista de abreviaciones

C.A.	Criterios de aceptación
API	Interfaz de programación de aplicaciones
ASQF	Der Arbeitskreis für Software-Qualität und Fortbildung
Desarrollo basado en pruebas de aceptación de ATDD	
BDD	Desarrollo impulsado por el comportamiento
Modelo y notación de procesos de negocio BPMN	
BVA	Análisis de valor límite
CC	Complejidad ciclomática
CD	Entrega continua
CFG	Gráfico de flujo de control
CI	Integración continua
Integración del modelo de madurez de capacidad CMMI	
CUNAS	Comercial disponible
UPC	Unidad Central de procesamiento
Gestión de relaciones con clientes CRM	
DDD	Diseño basado en dominio
Denegación de servicio distribuida DDoS	
Desarrollo y operaciones DevOps	
	Definición de hecho
Insecto	Definición de listo
Desarrollo, pruebas, aceptación y producción de DTAP.	
PE	Partición de equivalencia
FDD	Desarrollo impulsado por funciones
FMEA	Modo de falla y análisis de efectos.
GUI	Interfaz gráfica del usuario
CEI	Comisión Electrotécnica Internacional
IEEE	Instituto de Ingenieros Eléctricos y Electrónicos
IaC	Infraestructura como código
INVERTIR	Independiente, Negociable, Valioso, Estimable, Pequeño y Comprobable
IoT	Internet de las Cosas

IREB	Junta de ingeniería de requisitos internacionales
ISEB	Junta de examen de sistemas de información
YO ASI	Organización Internacional de Normalización
Junta Internacional de Cualificaciones de Pruebas de Software de ISTQB	
KPI	Indicador clave de rendimiento
LO	Objetivo de aprendizaje
LOC	Líneas de código
Pruebas basadas en modelos MBT	
MC/DC	Cobertura de decisión/condición modificada
Revisión del código moderno de MCR	
MTTF	Tiempo medio hasta el fallo
MTTR	Tiempo medio de reparación
N / A	No aplica
IMPERTINENTE	Programa de Evaluación y Revisión Técnica
AVENA	Pruebas de aceptación operativa
control de calidad	Seguro de calidad
control de calidad	Control de calidad
control de calidad	Gestión de la calidad
Requisito	Requisito
Ciclo de vida del desarrollo de software SDLC	
SMART	Especifico, mensurable, alcanzable, realista y con plazos determinados
SQL	Lenguaje de consulta estructurado
TC	Caso de prueba
TDD	Desarrollo basado en pruebas
Mapa	Enfoque de gestión de pruebas
Pruebas de aceptación del usuario UAT	
Interfaz de usuario	
Lenguaje de modelado unificado UML	
ARRIBA	Proceso unificado
A AGOTAR	Historia del usuario
WBS	Estructura de desglose del trabajo
WIP	Trabajo en proceso (o trabajo en progreso)
experiencia	Programación extrema
XSS	secuencias de comandos entre sitios

## Parte I

Certificación, plan de estudios y fundamento

Examen de nivel

# Certificado de nivel básico



## Historia del Certificado de Nivel Básico

La certificación independiente de probadores de software comenzó en 1998 en el Reino Unido.

En ese momento, bajo los auspicios de la Junta de Examen de Sistemas de Información (ISEB) de la Sociedad Británica de Computación, se estableció una unidad especial para pruebas de software: la Junta de Pruebas de Software ([www.bcs.org.uk/iseb](http://www.bcs.org.uk/iseb)). En 2002, la ASQF alemana ([www.asqf.de](http://www.asqf.de))

También creó su propio sistema de calificación para evaluadores. El programa de estudios Foundation Level se creó sobre la base de los programas de estudios ISEB y ASQF, y la información contenida en ellos se reorganizó, actualizó y complementó. La atención se ha centrado principalmente en temas que proporcionan el mayor apoyo práctico a los evaluadores.

El programa de estudios de nivel básico fue creado para:

- Enfatizar las pruebas como una de las principales especialidades profesionales dentro del software ingeniería
- Crear un marco estándar para el desarrollo profesional de los evaluadores. • Establecer un sistema que permita el reconocimiento de las calificaciones profesionales de los evaluadores por parte de empleadores, clientes y otros evaluadores y elevar el estatus de los evaluadores. • Promover buenas prácticas de prueba consistentes en toda la ingeniería de software. disciplinas
- Identificar cuestiones de prueba que sean relevantes y valiosas para la industria de TI en su conjunto. • Crear oportunidades para que los proveedores de software contraten evaluadores certificados y obtener una ventaja comercial sobre sus competidores al anunciar su política adoptada de reclutamiento de evaluadores. • Proporcionar una oportunidad para los evaluadores y aquellos interesados. en pruebas para obtener una calificación reconocida internacionalmente en el campo

Las certificaciones básicas existentes en el campo de las pruebas de software (por ejemplo, certificaciones emitidas por los consejos nacionales ISEB, ASQF o ISTQB®) que se otorgaron antes del inicio del certificado internacional se consideran equivalentes a este certificado.

El certificado básico no caduca y no es necesario renovarlo. La fecha de certificación figura en el certificado.

Las condiciones locales en cada país participante son responsabilidad de los consejos nacionales de ISTQB® (o "Juntas de Miembros"). Las responsabilidades de los consejos nacionales están definidas por ISTQB®, mientras que la implementación de estas responsabilidades se deja a las organizaciones miembros individuales. Las responsabilidades de los consejos nacionales suelen incluir la acreditación de proveedores de formación y la programación de exámenes.

#### Trayectorias profesionales para evaluadores

El sistema creado por ISTQB® permite definir trayectorias profesionales para los profesionales de pruebas en base a un programa de certificación de tres niveles que incluye Nivel Básico, Nivel Avanzado y Nivel Experto. El punto de entrada es el Nivel Básico descrito en este libro. Tener la certificación Foundation Level es un requisito previo para obtener certificaciones posteriores. Los titulares de un certificado de Nivel Básico pueden ampliar sus conocimientos sobre las pruebas obteniendo una calificación de Nivel Avanzado. En este nivel, ISTQB® ofrece una serie de programas educativos. En cuanto a la ruta básica, existen tres programas posibles:

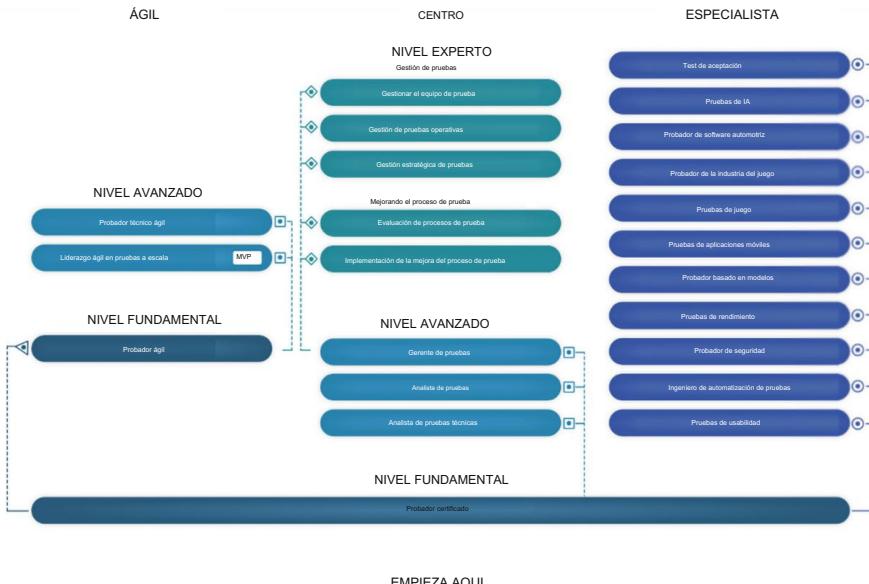
- Analista de pruebas técnicas (orientado a la tecnología, pruebas no funcionales, análisis estático, técnicas de prueba de caja blanca, trabajo con código fuente)
- Analista de pruebas (orientado al cliente, comprensión del negocio, pruebas funcionales, técnicas de prueba de caja negra y pruebas basadas en la experiencia)
- Test Manager (orientado al proceso de prueba y a la gestión del equipo de prueba)

El nivel avanzado es el punto de partida para adquirir más conocimientos y habilidades a nivel experto. Una persona que ya ha adquirido experiencia como director de pruebas, por ejemplo, puede optar por desarrollar aún más su carrera como evaluador obteniendo certificaciones de nivel experto en las áreas de gestión de pruebas y mejora de procesos de pruebas.

Además de la pista principal, ISTQB® también ofrece programas educativos especializados en temas como pruebas de aceptación, pruebas de inteligencia artificial, pruebas automotrices, pruebas de máquinas de juego, pruebas de juegos, pruebas de aplicaciones móviles, pruebas basadas en modelos, pruebas de rendimiento, pruebas de seguridad, automatización de pruebas o pruebas de usabilidad. En términos de metodologías ágiles, es Technical Agile Tester o Agile Test Leadership at Scale.

La Figura 1 muestra el esquema de certificación ofrecido por ISTQB® al 25 de junio de 2023.  
La última versión de la revisión de la trayectoria profesional de ISTQB® está disponible en [www.istqb.org](http://www.istqb.org).

## Objetivos del Sistema Internacional de Cualificaciones

Fig. 1 Esquema de certificación oficial ISTQB® (fuente: [www.istqb.org](http://www.istqb.org))

## Público objetivo

La calificación Foundation Level está diseñada para cualquier persona interesada en las pruebas de software. Esto puede incluir evaluadores, analistas de pruebas, ingenieros de pruebas, consultores de pruebas, administradores de pruebas, usuarios que realizan pruebas de aceptación, miembros de equipos ágiles y desarrolladores. Además, la calificación Foundation Level es adecuada para aquellos que buscan adquirir conocimientos básicos en pruebas de software, como gerentes de proyectos, gerentes de calidad, gerentes de desarrollo de software, analistas de negocios, CIO y consultores de gestión.

## Objetivos del Sistema Internacional de Cualificaciones

- Sentar las bases para comparar el conocimiento sobre pruebas en diferentes países • Facilitar que los evaluadores encuentren trabajo en otros países • Garantizar una comprensión común de las cuestiones de pruebas en proyectos internacionales • Incrementar el número de evaluadores calificados en todo el mundo • Crear una iniciativa internacional que proporcione mayores beneficios y una mayor impacto que las iniciativas implementadas a nivel nacional
- Desarrollar un cuerpo internacional común de información y conocimientos sobre pruebas sobre la base de programas de estudio y un glosario de términos de pruebas y elevar el nivel de conocimientos sobre pruebas entre los trabajadores de TI.

- Promover la profesión de evaluación en más países. • Permitir que los evaluadores obtengan calificaciones ampliamente reconocidas en su país de origen. idioma
- Crear condiciones para que los evaluadores de diferentes países compartan conocimientos y recursos
- Garantizar el reconocimiento internacional del estatus de los evaluadores y de esta calificación.

## Programa de estudios de nivel básico



### Resultados comerciales

Asociado a cada plan de estudios ISTQB® hay un conjunto de los llamados resultados comerciales (objetivos comerciales). Un resultado empresarial es un resultado o cambio conciso, definido y observable en el desempeño empresarial, respaldado por una medida específica. La Tabla 1 enumera 14 resultados comerciales a los que debería contribuir un candidato que reciba la certificación Foundation Level.

### Objetivos de aprendizaje y niveles K

El contenido de cada programa de estudios se crea para cubrir el conjunto de objetivos de aprendizaje establecidos para ese programa de estudios. Los objetivos de aprendizaje respaldan el logro de las metas comerciales y se utilizan para crear exámenes para el certificado de nivel básico.

Comprender cuáles son los objetivos de aprendizaje y conocer la relación entre los objetivos de aprendizaje y las preguntas del examen son clave para una preparación eficaz para el examen de certificación.

Todos los objetivos de aprendizaje están definidos en el plan de estudios de tal forma que cada uno de ellos constituye un todo indivisible. Los objetivos de aprendizaje se definen al comienzo de cada sección del programa de estudios. Cada sección del programa de estudios aborda exactamente un objetivo de aprendizaje. Esto hace posible vincular sin ambigüedades cada objetivo de aprendizaje (y preguntas del examen) con partes bien definidas del material.

La siguiente sección describe los objetivos de aprendizaje aplicables al programa de estudios del nivel básico. El conocimiento de cada tema cubierto en el programa de estudios se evaluará en el examen de acuerdo con el objetivo de aprendizaje asignado. A cada objetivo de aprendizaje se le asigna un llamado nivel de conocimiento, también conocido como nivel cognitivo (o nivel K), K1, K2 o K3, que determina el grado en que se debe asimilar un determinado material. Los niveles de conocimiento para cada objetivo de aprendizaje son

Tabla 1 Resultados comerciales perseguidos por un evaluador certificado de nivel básico

FL-BO1	Comprender qué son las pruebas y por qué son beneficiosas
FL-BO2	Comprender los conceptos fundamentales de las pruebas de software.
FL-BO3	Identificar el enfoque de prueba y las actividades a implementar dependiendo del contexto de prueba
FL-BO4	Evaluar y mejorar la calidad de la documentación.
FL-BO5	Incrementar la efectividad y eficiencia de las pruebas.
FL-BO6	Alinear el proceso de prueba con el ciclo de vida del desarrollo de software.
FL-BO7	Comprender los principios de gestión de pruebas
FL-BO8	Redactar y comunicar informes de defectos claros y comprensibles.
FL-BO9	Comprender los factores que influyen en las prioridades y esfuerzos relacionados con las pruebas.
FL-BO10	Trabajar como parte de un equipo multifuncional.
FL-BO11	Conozca los riesgos y beneficios relacionados con la automatización de pruebas
FL-BO12	Identificar las habilidades esenciales necesarias para las pruebas.
FL-BO13	Comprender el impacto del riesgo en las pruebas
FL-BO14	Informar eficazmente sobre el progreso y la calidad de las pruebas.

presentado junto a cada objetivo de aprendizaje enumerado al comienzo de cada capítulo de las sílabas.

Nivel 1: Recordar (K1): el candidato recuerda, reconoce o recuerda un término o concepto.

Verbos de acción: identificar, recordar, recordar, reconocer

Ejemplos:

- "Identificar objetivos de prueba típicos".
- "Recordar los conceptos de la pirámide de pruebas".
- "Reconocer cómo un evaluador agrega valor a la planificación de iteraciones y lanzamientos".

Nivel 2: Comprender (K2): el candidato puede seleccionar las razones o explicaciones de declaraciones relacionadas con el tema y pueden resumir, comparar, clasificar y dar Ejemplos para el concepto de prueba.

Verbos de acción: clasificar, comparar, contrastar, diferenciar, distinguir, ejemplificar, explicar, dar ejemplos, interpretar, resumir

Ejemplos:

- "Clasificar las diferentes opciones para redactar criterios de aceptación".
- "Comparar los diferentes roles en las pruebas" (buscar similitudes, diferencias, o ambos).
- "Distinguir entre riesgos del proyecto y riesgos del producto" (permite que los conceptos sean diferenciado).
- "Ejemplificar el propósito y el contenido de un plan de prueba".
- "Explicar el impacto del contexto en el proceso de prueba".
- "Resumir las actividades del proceso de revisión".

Nivel 3: Aplicar (K3): el candidato puede realizar un procedimiento cuando se enfrenta a una tarea familiar o seleccionar el procedimiento correcto y aplicarlo en un contexto determinado.

Verbos de acción: Aplicar, implementar, preparar, usar

Ejemplos:

- “Aplicar la priorización de casos de prueba”.
- “Preparar un informe de defectos”.
- “Usar análisis de valores límite para derivar casos de prueba”.

Referencias para los niveles cognitivos de los objetivos de aprendizaje:

Anderson, LW y Krathwohl, DR (eds) (2001) Una taxonomía para el aprendizaje, la enseñanza y la evaluación: una revisión de la taxonomía de objetivos educativos de Bloom, Allyn & Bacon

## Requisitos para los candidatos

Los candidatos que toman el examen ISTQB® Certified Tester Foundation Level solo deben tener interés en las pruebas de software. Sin embargo, se recomienda encarecidamente que los candidatos:

- Tener al menos experiencia básica en desarrollo o pruebas de software, como 6 meses de experiencia como probador realizando pruebas de sistemas o pruebas de aceptación o como desarrollador. • Haber completado un curso de capacitación ISTQB® (acreditado por uno de los consejos nacionales de ISTQB® en acuerdo con los estándares ISTQB®)

Estos no son requisitos formales, aunque cumplirlos hace que sea mucho más fácil prepararse y aprobar el examen de certificación. Cualquiera puede realizar el examen, independientemente de su interés o experiencia como tester.

## Referencias a normas y estándares

El programa de estudios contiene referencias a estándares y normas IEEE, ISO, etc. El propósito de estas referencias es proporcionar un marco conceptual o remitir al lector a una fuente que pueda utilizar para obtener información adicional. Cabe señalar, sin embargo, que sólo podrán ser objeto del examen aquellas disposiciones de las normas o estándares referenciados a las que se refieren los apartados específicamente discutidos del programa de estudios. El contenido de las normas y estándares no es el tema del examen y las referencias a estos documentos tienen únicamente fines informativos.

La versión actual del programa de estudios (v4.0) se refiere a los siguientes estándares:

- ISO/IEC/IEEE 29119—Estándar de prueba de software. Esta norma consta de varias partes, las más importantes desde el punto de vista del programa de estudios son la Parte 1 (conceptos generales) [1], la Parte 2 (procesos de prueba) [2], la Parte 3 (documentación de prueba) [3] y Parte 4 (técnicas de prueba) [4]. La parte 3 de este estándar reemplaza el estándar IEEE 829 retirado.
- ISO/IEC 25010: Evaluación y requisitos de calidad de sistemas y software (también conocido como SQuaRE), modelos de calidad de sistemas y software [5]. Este estándar describe el modelo de calidad del software y reemplaza el estándar ISO 9126 retirado. • ISO/IEC 20246—Revisiones de productos de trabajo [6]. Esta norma describe cuestiones relacionadas con las revisiones de productos de trabajo. Reemplaza el estándar IEEE 1028 retirado. • ISO 31000: Gestión de riesgos, principios y directrices [7]. Esta norma describe el proceso de gestión de riesgos.

## Actualización continua

La industria de TI está experimentando cambios dinámicos. Para tener en cuenta la situación cambiante y garantizar que las partes interesadas tengan acceso a información útil y actualizada, los grupos de trabajo de ISTQB® han creado una lista de enlaces a documentos de respaldo y cambios en los estándares, que está disponible en [www.istqb.org](http://www.istqb.org). La información anterior no está sujeta al examen del programa de estudios de nivel básico.

## Notas de la versión para el programa de estudios de nivel básico v4.0

El programa de estudios de Foundation Level v4.0 incluye mejores prácticas y técnicas que han resistido la prueba del tiempo, pero se han realizado una serie de cambios significativos con respecto a la versión anterior (v3.1) en 2018 para presentar el material de una manera más moderna. hacerlo más relevante para el nivel básico y tener en cuenta los cambios que se han producido en la ingeniería de software en los últimos años. En particular:

- Mayor énfasis en los métodos y prácticas utilizados en los modelos ágiles de desarrollo de software (enfoque de todo el equipo, enfoque de desplazamiento a la izquierda, planificación de iteraciones y lanzamientos, pirámide de pruebas, cuadrantes de pruebas, prácticas de “prueba primero” como TDD, BDD o ATDD).
- La sección sobre evaluación de habilidades, particularmente habilidades interpersonales, se ha ampliado y profundizado.
- Se ha reorganizado y mejor estructurado la sección de gestión de riesgos. • Se ha agregado una sección que analiza el enfoque DevOps. • Se ha agregado una sección que analiza en detalle algunas técnicas de estimación de pruebas. • La técnica de prueba de decisión fue reemplazada por prueba de rama. • Se ha eliminado una sección que describe técnicas de revisión detalladas.

## Contenido del plan de estudios

- Se ha eliminado la técnica de prueba basada en casos de uso (se analiza en el Plan de estudios de nivel avanzado "Analista de pruebas").
- Se ha eliminado una sección que analiza estrategias de prueba de muestra. • Se han eliminado algunos contenidos relacionados con las herramientas, en particular la cuestión de introducir una herramienta en una organización o realizar un proyecto piloto.

## Contenido del plan de estudios

### Capítulo 1. Fundamentos de las pruebas

- El lector aprende los principios básicos relacionados con las pruebas, las razones por las que las pruebas son requeridos y cuáles son los objetivos de la prueba.
- El lector comprende el proceso de prueba, las principales actividades de prueba y el software de prueba. • El lector comprende las habilidades esenciales para realizar pruebas.

#### Objetivos de aprendizaje

##### 1.1. ¿Qué son las pruebas?

FL-1.1.1 (K1) Identificar objetivos de prueba típicos.

FL-1.1.2 (K2) Diferenciar las pruebas de la depuración.

##### 1.2. ¿Por qué son necesarias las pruebas?

FL-1.2.1 (K2) Ejemplifique por qué las pruebas son necesarias.

FL-1.2.2 (K1) Recordar la relación entre pruebas y aseguramiento de la calidad.

FL-1.2.3 (K2) Distinguir entre causa raíz, error, defecto y falla.

##### 1.3. Principios de prueba

FL-1.3.1 (K2) Explicar los siete principios de prueba.

##### 1.4. Actividades de prueba, software de prueba y roles de prueba

FL-1.4.1 (K2) Resumir las diferentes actividades y tareas de prueba.

FL-1.4.2 (K2) Explicar el impacto del contexto en el proceso de prueba.

FL-1.4.3 (K2) Diferenciar el software de prueba que soporta las actividades de prueba.

FL-1.4.4 (K2) Explicar el valor de mantener la trazabilidad.

FL-1.4.5 (K2) Comparar los diferentes roles en las pruebas.

##### 1.5. Habilidades esenciales y buenas prácticas en las pruebas.

FL-1.5.1 (K2) Dé ejemplos de las habilidades genéricas requeridas para las pruebas.

FL-1.5.2 (K1) Recuerde las ventajas del enfoque de equipo completo.

FL-1.5.3 (K2) Distinguir los beneficios y desventajas de la independencia de las pruebas.

## Capítulo 2. Pruebas a lo largo del ciclo de vida del desarrollo de software

- El lector aprende cómo se incorporan las pruebas en diferentes desarrollos.  
enfoques.
- El lector aprende los conceptos de enfoques de prueba primero, así como DevOps. • El lector aprende sobre los diferentes niveles de prueba, tipos de prueba y mantenimiento.  
pruebas.

### Objetivos de aprendizaje

2.1. Pruebas en el contexto de un ciclo de vida de desarrollo de software.

FL-2.1.1 (K2) Explicar el impacto del ciclo de vida de desarrollo de software elegido en las pruebas.

FL-2.1.2 (K1) Recordar buenas prácticas de prueba que se aplican a todos los ciclos de vida de desarrollo de software.

FL-2.1.3 (K1) Recuerde los ejemplos de enfoques de desarrollo basados en la prueba.

FL-2.1.4 (K2) Resuma cómo DevOps podría tener un impacto en las pruebas.

FL-2.1.5 (K2) Explique el enfoque de desplazamiento hacia la izquierda.

FL-2.1.6 (K2) Explicar cómo se pueden utilizar las retrospectivas como mecanismo para la mejora de procesos.

### 2.2 Niveles de prueba y tipos de prueba

FL-2.2.1 (K2) Distinguir los diferentes niveles de prueba.

FL-2.2.2 (K2) Distinguir los diferentes tipos de pruebas.

FL-2.2.3 (K2) Distinguir las pruebas de confirmación de las pruebas de regresión.

### 2.3 Pruebas de mantenimiento

FL-2.3.1 (K2) Resumir las pruebas de mantenimiento y sus desencadenantes.

## Capítulo 3. Pruebas estáticas

- El lector aprende sobre los conceptos básicos de las pruebas estáticas, la retroalimentación y el proceso de revisión.

### Objetivos de aprendizaje

#### 3.1. Conceptos básicos de las pruebas estáticas

FL-3.1.1 (K1) Reconocer tipos de productos que pueden ser examinados mediante las diferentes técnicas de ensayo estático.

FL-3.1.2 (K2) Explicar el valor de las pruebas estáticas.

FL-3.1.3 (K2) Comparar y contrastar pruebas estáticas y dinámicas.

#### 3.2. Proceso de retroalimentación y revisión

FL-3.2.1 (K1) Identificar los beneficios de la retroalimentación temprana y frecuente de las partes interesadas.

FL-3.2.2 (K2) Resumir las actividades del proceso de revisión.

FL-3.2.3 (K1) Recordar qué responsabilidades se asignan a los roles principales al realizar revisiones.

FL-3.2.4 (K2) Comparar y contrastar los diferentes tipos de revisión.

FL-3.2.5 (K1) Recuerde los factores que contribuyen a una revisión exitosa.

## Capítulo 4. Análisis y diseño de pruebas

- El lector aprende cómo aplicar técnicas de prueba de caja negra, caja blanca y basadas en la experiencia para derivar casos de prueba de diversos productos de trabajo de software.
- El lector aprende sobre el enfoque de prueba basado en colaboración.

### Objetivos de aprendizaje

#### 4.1. Descripción general de las técnicas de prueba

FL-4.1.1 (K2) Distinguir técnicas de prueba de caja negra, de caja blanca y basadas en la experiencia.

#### 4.2. Técnicas de prueba de caja negra

FL-4.2.1 (K3) Utilice la partición de equivalencia para derivar casos de prueba.

FL-4.2.2 (K3) Utilice el análisis de valores límite para derivar casos de prueba.

FL-4.2.3 (K3) Utilice pruebas de tablas de decisión para derivar casos de prueba.

FL-4.2.4 (K3) Utilice pruebas de transición de estado para derivar casos de prueba.

#### 4.3. Técnicas de prueba de caja blanca

- FL-4.3.1 (K2) Explicar la prueba de declaraciones.
- FL-4.3.2 (K2) Explicar las pruebas de rama.
- FL-4.3.3 (K2) Explicar el valor de las pruebas de caja blanca.

#### 4.4. Técnicas de prueba basadas en la experiencia.

- FL-4.4.1 (K2) Explicar los errores de adivinación.
- FL-4.4.2 (K2) Explicar las pruebas exploratorias.
- FL-4.4.3 (K2) Explicar las pruebas basadas en listas de verificación.

#### 4.5. Enfoques de prueba basados en la colaboración

- FL-4.5.1 (K2) Explicar cómo escribir historias de usuarios en colaboración con desarrolladores y representantes comerciales.
- FL-4.5.2 (K2) Clasificar las diferentes opciones para redactar criterios de aceptación.
- FL-4.5.3 (K3) Utilice el desarrollo basado en pruebas de aceptación (ATDD) para derivar pruebas casos.

### Capítulo 5. Gestión de las actividades de prueba

- El lector aprende cómo planificar pruebas en general y cómo estimar el esfuerzo de las pruebas.
- El lector aprende cómo los riesgos pueden influir en el alcance de las pruebas.
- El lector aprende cómo monitorear y controlar las actividades de prueba.
- El lector aprende cómo la gestión de la configuración respalda las pruebas. • El lector aprende a informar defectos de forma clara y comprensible.

#### Objetivos de aprendizaje

##### 5.1 Planificación de pruebas

- FL-5.1.1 (K2) Ejemplificar el propósito y contenido de un plan de prueba.
- FL-5.1.2 (K1) Reconocer cómo un evaluador agrega valor a la planificación de iteraciones y lanzamientos.
- FL-5.1.3 (K2) Comparar y contrastar criterios de entrada y criterios de salida.
- FL-5.1.4 (K3) Utilizar técnicas de estimación para calcular el esfuerzo de prueba requerido.
- FL-5.1.5 (K3) Aplicar priorización de casos de prueba.
- FL-5.1.6 (K1) Recordar los conceptos de la pirámide de pruebas.
- FL-5.1.7 (K2) Resumir los cuadrantes de prueba y sus relaciones con los niveles y tipos de prueba.

### 5.2 Gestión de riesgos

FL-5.2.1 (K1) Identificar el nivel de riesgo utilizando la probabilidad del riesgo y el impacto del riesgo.

FL-5.2.2 (K2) Distinguir entre riesgos del proyecto y riesgos del producto.

FL-5.2.3 (K2) Explicar cómo el análisis de riesgos del producto puede influir en la minuciosidad y el alcance de las pruebas.

FL-5.2.4 (K2) Explicar qué medidas se pueden tomar en respuesta a los riesgos del producto analizado.

### 5.3 Monitoreo de pruebas, control de pruebas y finalización de pruebas

FL-5.3.1 (K1) Recuperar métricas utilizadas para las pruebas.

FL-5.3.2 (K2) Resumir los propósitos, el contenido y las audiencias de los informes de prueba.

FL-5.3.3 (K2) Ejemplificar cómo comunicar el estado de las pruebas.

### 5.4 Gestión de la configuración

FL-5.4.1 (K2) Resumir cómo la gestión de la configuración respalda las pruebas.

### 5.5 Gestión de defectos

FL-5.5.1 (K3) Elaborar un informe de defectos.

## Capítulo 6. Herramientas de prueba

- El lector aprende a clasificar herramientas y a comprender los riesgos y beneficios de las pruebas. automatización.

### Objetivos de aprendizaje

#### 6.1 Soporte de herramientas para pruebas

FL-6.1.1 (K2) Explicar cómo los diferentes tipos de herramientas de prueba respaldan las pruebas.

#### 6.2. Beneficios y riesgos de la automatización de pruebas

FL-6.2.1 (K1) Recordar los beneficios y riesgos de la automatización de pruebas.

# Examen de nivel básico



## Estructura del examen

La descripción del examen de certificación Foundation Level se define en un documento titulado "Reglas de estructura del examen", que está disponible en [www.istqb.org](http://www.istqb.org).

El examen tiene la forma de una prueba de opción múltiple y consta de 40 preguntas. Para aprobar el examen es necesario responder correctamente al menos el 65% de las preguntas (es decir, 26 preguntas).

Los exámenes pueden realizarse como parte de un curso de formación acreditado o de forma independiente (por ejemplo, en un centro examinador o en un examen público). Completar un curso acreditado no es un requisito previo para realizar el examen, pero se recomienda asistir a dicho curso, ya que le permite comprender mejor el material y aumenta significativamente sus posibilidades de aprobar el examen. Si no apruebas el examen, podrás volver a realizarlo tantas veces como quieras.

## Reglas del examen

- Los exámenes de Foundation Level se basan en el programa de estudios de Foundation Level [8]. • Para responder una pregunta del examen puede ser necesario utilizar material de más de una sección del programa de estudios. •

Todos los objetivos de aprendizaje incluidos en el programa de estudios (con niveles cognitivos desde K1 hasta K3) están sujetos a examen.

- Todas las definiciones de palabras clave del programa de estudios están sujetas a examen (en el nivel K1). Hay un diccionario en línea disponible en [www.glossary.istqb.org](http://www.glossary.istqb.org). • Cada examen de Foundation Level consta de un conjunto de preguntas de opción múltiple basadas en los objetivos de aprendizaje del programa de estudios de Foundation Level. El nivel de cobertura y distribución de las preguntas se basa en los objetivos de aprendizaje, sus niveles K y su nivel de importancia según la evaluación ISTQB®. Para detalles sobre

Tabla 1 Distribución de preguntas del examen por nivel K

Nivel K	Número de preguntas	Tiempo por pregunta [en min]	Tiempo promedio para el nivel K [en min]
K1	8	1	8
K2	24	1	24
K3	8	3	24
Total	40		56

la estructura de cada componente del examen, consulte la "Distribución de preguntas" subsección siguiente.

- En general, se espera que el tiempo para leer, analizar y responder preguntas en Los niveles K1 y K2 no deben exceder 1 min, y en el nivel K3, puede tardar 3 min. Sin embargo, esto es sólo una guía para el tiempo promedio y es probable que algunos las preguntas requerirán más y otras menos tiempo para responder.
- El examen consta de 40 preguntas de opción múltiple. Cada respuesta correcta vale un punto (independientemente del nivel K del objetivo de aprendizaje al que se refiere la pregunta). aplica). La puntuación máxima posible del examen es de 40 puntos.
- El tiempo asignado para el examen es exactamente de 60 min. Si el candidato es nativo El idioma no es el idioma del examen, al candidato se le permite un adicional 25% del tiempo (en el caso del examen Foundation Level, esto significa que el El examen tendrá una duración de 75 min).
- Se requiere un mínimo del 65% (26 puntos) para aprobar.
- En la Tabla 1 se muestra un desglose general de las preguntas por nivel K.

Las reglas y recomendaciones para escribir preguntas de opción múltiple se pueden encontrar en ISTQB®: documento de información del examen [9].

Si sumas el tiempo esperado para responder las preguntas según las reglas dado anteriormente, entonces—teniendo en cuenta la distribución de preguntas por niveles K— Descubrirá que le llevará unos 56 minutos responder todas las preguntas. Esto deja una reserva de 4 min.

Cada pregunta del examen debe evaluar al menos un objetivo de aprendizaje (LO) del Programa de estudios del nivel básico. Las preguntas pueden incluir términos y conceptos que existen en las secciones de nivel K1, ya que se espera que los candidatos estén familiarizados con ellas. En caso de que el preguntas están relacionadas con más de un LO, deben referirse (y ser asignadas) al objetivo de aprendizaje con el valor más alto del nivel K.

## Distribución de preguntas

La estructura del examen de nivel básico se muestra en la Tabla 2. Cada examen requiere preguntas obligatorias que cubren objetivos de aprendizaje específicos, así como un cierto número de preguntas basadas en los objetivos de aprendizaje seleccionados. Si el número de Los objetivos de aprendizaje son mayores que el número de preguntas para un grupo específico de

## Distribución de preguntas

Tabla 2 Distribución detallada de las preguntas del examen por niveles K y capítulos

La distribución de preguntas	k-nivel	Número de preguntas de el grupo de aprendizaje seleccionado objetivos	Puntuación de un solo pregunta	
<b>Capítulo 1</b>				
FL-1.1.1, FL-1.2.2 K1 1			1	
FL-1.5.2	K1 1		1	
FL-1.1.2, FL-1.2.1, FL-1.2.3	K2 1		1	
FL-1.3.1	K2 1		1	
FL-1.4.1, FL-1.4.2, FL-1.4.3, FL-1.4.4, FL-1.4.5	K2 3		1	
FL-1.5.1, FL-1.5.3 K2 1	Capítulo		1	
<b>2</b>				
FL-2.1.2	K1 1		1	
FL-2.1.3	K1 1		1	
FL-2.2.1, FL-2.2.2 K2 1	FL-2.2.3,		1	
FL-2.3.1 K2 1	FL-2.1.1, FL-2.1.6		1	
K2 1	FL-2.1.4, FL-2.1.5	K2 1		
			1	
<b>Capítulo 3</b>				
FL-3.1.1, FL-3.2.1, FL-3.2.3, FL-3.2.5	K1 2		1	
FL-3.1.2, FL-3.1.3 K2 1			1	
FL-3.2.2, FL-3.2.4 K2 1			1	
<b>Capítulo 4</b>				
FL-4.1.1	K1 1		1	
FL-4.3.1, FL-4.3.2, FL-4.3.3	K2 2		1	
FL-4.4.1, FL-4.4.2, FL-4.4.3	K2 2		1	
FL-4.5.1, FL-4.5.2 K2 1	FL-4.2.1,		1	
FL-4.2.2, FL-4.2.3, FL-4.2.4, FL-4.5.3	K3 5		1	
<b>Capítulo 5</b>				
FL-5.1.2, FL-5.1.6, FL-5.2.1, FL-5.3.1	K1 1		1	
FL-5.1.1, FL-5.1.3 K2 1			1	
FL-5.1.7	K2 1		1	
FL-5.2.2, FL-5.2.3, FL-5.2.4	K2 1		1	
FL-5.3.2, FL-5.3.3 K2 1			1	

(continuado)

Tabla 2 (continuación)

La distribución de preguntas	k-nivel	Número de preguntas del grupo seleccionado de objetivos de aprendizaje	Puntuación de una sola pregunta	
FL-5.4.1	K2 1		1	
FL-5.1.4, FL-5.1.5, FL-5.5.1	K3 3		1	
<b>Capítulo 6</b>				
FL-6.1.1	K2 1		1	Se requieren un total de 2 preguntas para el Cap. 6 K1 = 1 K2 = 1 K3 = 0 Número de puntos para este capítulo = 2
FL-6.2.1	K1 1		1	
Probador certificado: nivel básico RESUMEN			40 puntos, 60 minutos	Un total de 40 preguntas

objetivos de aprendizaje descritos en la tabla, cada pregunta debe cubrir un objetivo de aprendizaje diferente.

El análisis de la Tabla 2 muestra que los siguientes 17 objetivos de aprendizaje son seguro que será cubierto y examinado:

- Cinco preguntas de nivel K1 (FL-1.5.2, FL-2.1.2, FL-2.1.3, FL-4.1.1, FL-6.2.1) • Cuatro preguntas de nivel K2 (FL-1.3.1, FL-5.1.7, FL-5.4.1, FL-6.1.1) • Ocho preguntas que cubren los ocho objetivos de aprendizaje en el nivel K3

Cada una de las 23 preguntas restantes se seleccionará de un grupo de dos o más objetivos de aprendizaje. Dado que no se sabe cuál de estos objetivos de aprendizaje se cubrirá, el candidato debe dominar todo el material del programa de estudios (todos los objetivos de aprendizaje) de todos modos.

## Consejos: antes y durante el examen

Para aprobar con éxito el examen, lo primero que debe hacer es leer atentamente el programa de estudios y el glosario de términos, cuyo conocimiento se requiere en el nivel básico. Esto se debe a que el examen se basa en estos dos documentos. Es aconsejable

También resuelva preguntas de prueba de muestra y exámenes de muestra. En el sitio web de ISTQB® ([www.istqb.org](http://www.istqb.org)), Puede encontrar conjuntos de exámenes de muestra oficiales en inglés y en los sitios web de las Juntas de miembros en otros idiomas. La lista de todas las juntas miembros de ISTQB® y sus sitios web se publica en [www.istqb.org/certifications/member-board-list](http://www.istqb.org/certifications/member-board-list).

Esta publicación, además de una serie de preguntas de muestra para cada capítulo de El programa de estudios también incluye el examen de muestra oficial ISTQB®.

Durante el examen en sí, debes:

- Lea las preguntas con atención; a veces una palabra cambia todo el significado de la pregunta o es una pista para dar la respuesta correcta!
- Preste atención a las palabras clave (por ejemplo, en qué modelo de ciclo de vida de desarrollo de software se ejecuta el proyecto). • Intente hacer coincidir la pregunta con el objetivo de aprendizaje; entonces será más fácil comprender la idea de la pregunta y justificar la corrección e incorrección de las respuestas individuales.
- Tenga cuidado con las preguntas que contienen negación (p. ej., “cuál de las siguientes NO es...”): en tales preguntas, tres respuestas serán afirmaciones verdaderas y una será una afirmación falsa. Debe indicar la respuesta que contiene la afirmación falsa. • Elija la opción que responda directamente a la pregunta. Algunas respuestas pueden ser oraciones completamente correctas, pero no responden a la pregunta formulada; por ejemplo, la pregunta es sobre los riesgos de la automatización y una de las respuestas menciona algún beneficio de la automatización.
- Adivina si no sabes qué respuesta elegir: no hay puntos negativos, por eso no vale la pena dejar preguntas sin respuesta.
- Recuerde que las respuestas con frases fuertes y categóricas suelen ser incorrectas (p. ej., “siempre”, “debe ser”, “nunca”, “en cualquier caso”), aunque es posible que esta regla no se aplique en todos los casos.

## Parte II

El contenido del programa de estudios

# Capítulo 1 Fundamentos de las pruebas



Palabras clave:

Cobertura	el grado en que se ejercen los elementos de cobertura específicos por un conjunto de pruebas expresado como porcentaje. Sinónimos: prueba cobertura.
Depuración	el proceso de encontrar, analizar y eliminar las causas de fallas en un componente o sistema.
Defecto	una imperfección o deficiencia en un producto de trabajo donde no cumple con sus requisitos o especificaciones. Despues de ISO 24765. Sinónimos: error, fallo.
Error	una acción humana que produce un resultado incorrecto. Despues de ISO 24765. Sinónimos: error.
Falla	Un evento en el que un componente o sistema no funciona. una función requerida dentro de límites especificados. Despues de ISO 24765.
Calidad	El grado en que un producto de trabajo satisface lo establecido y necesidades implícitas de sus stakeholders. Despues del IREB.
Seguro de calidad	actividades enfocadas a brindar confianza en que la calidad se cumplirán los requisitos. Abreviatura: control de calidad. Despues ISO 24765.
Causa principal	una fuente de un defecto tal que si se elimina, el La aparición del tipo de defecto disminuye o se elimina.
Análisis de prueba	Referencias: CMMI la actividad que identifica las condiciones de prueba analizando la base de prueba.
Base de prueba	El conjunto de conocimientos utilizados como base para el análisis de las pruebas. y diseño. Despues de TMap.

Caso de prueba	un conjunto de condiciones previas, insumos, acciones (cuando corresponda), resultados esperados y postcondiciones, desarrollados en base a condiciones de la prueba.
Finalización de la prueba	la actividad que hace que el software de prueba esté disponible para su uso posterior, deja los entornos de prueba en condiciones satisfactorias, y Comunica los resultados de las pruebas a las autoridades pertinentes, partes interesadas.
Condición de prueba	un aspecto comprobable de un componente o sistema identificado como una base para las pruebas. Referencias: ISO 29119-1. Sinónimos: prueba, situación, requisito de prueba.
Control de prueba	la actividad que desarrolla y aplica acciones correctivas para poner en marcha un proyecto de prueba cuando se desvía de lo que estaba planificado.
Datos de prueba	datos necesarios para la ejecución de la prueba. Sinónimos: conjunto de datos de prueba.
Diseño de prueba	La actividad que deriva y especifica casos de prueba a partir de pruebas, condiciones.
Ejecución de pruebas	La actividad que ejecuta una prueba en un componente o sistema, produciendo resultados reales.
Implementación de pruebas:	la actividad que prepara el software de prueba necesario para la prueba, ejecución basada en el análisis y diseño de pruebas.
Monitoreo de pruebas	la actividad que verifica el estado de las actividades de prueba, identifica cualquier variación respecto de lo planificado o esperado, y informa el estado a las partes interesadas.
Objeto de prueba	el producto de trabajo a probar.
Objetivo de la prueba	el propósito de la prueba. Sinónimos: objetivo de prueba.
Planificación de pruebas	la actividad de establecer o actualizar un plan de prueba.
Procedimiento de prueba	una secuencia de casos de prueba en orden de ejecución y cualquier acciones asociadas que pueden ser necesarias para configurar el condiciones previas iniciales y cualquier actividad final posterior ejecución. Referencias: ISO 29119-1.
Resultado de la prueba	la consecuencia/resultado de la ejecución de una prueba. Sinónimos: resultado, resultado de la prueba, resultado.
Pruebas	El proceso dentro del ciclo de vida del desarrollo de software que Evalúa la calidad de un componente o sistema y sus elementos relacionados, productos del trabajo.
Software de prueba	productos de trabajo producidos durante el proceso de prueba para su uso en planificar, diseñar, ejecutar, evaluar y generar informes en las pruebas. Según ISO 29119-1.
Validación	Confirmación mediante examen de que un producto de trabajo coincide las necesidades de una parte interesada. Después del IREB.
Verificación	Confirmación mediante examen y mediante la provisión de evidencia objetiva de que los requisitos especificados han sido cumplido. Referencias: ISO 9000.

## 1.1 ¿Qué son las pruebas?

## 1.1.1 ¿Qué son las pruebas?

FL-1.1.1 (K1) Identificar objetivos de prueba típicos.

FL-1.1.2 (K2) Diferenciar las pruebas de la depuración.

Hoy en día probablemente no existe ningún ámbito de la vida en el que no se utilice en mayor o menor medida el software. Los sistemas de información están desempeñando un papel cada vez más importante en nuestras vidas, desde soluciones empresariales (sector bancario, seguros) hasta dispositivos de consumo (coches), entretenimiento (juegos de ordenador) o comunicaciones. El uso de software que contiene defectos puede:

- Causar una pérdida de dinero o tiempo • Causar una pérdida de confianza del cliente • Dificultar la obtención de nuevos clientes • Eliminar del mercado • En situaciones extremas: causar una amenaza a la salud o la vida

Pruebas  del software permite evaluar la calidad del software y contribuye a reducir el riesgo de fallo del software en acción. Por lo tanto, unas buenas pruebas son esenciales para el éxito del proyecto. Las pruebas de software son un conjunto de actividades que se llevan a cabo para facilitar la detección de defectos y evaluar las propiedades de los artefactos de software.

Cada uno de estos artefactos de prueba se conoce como objeto de prueba. 

Muchas personas, incluidas aquellas que trabajan en la industria de TI, piensan erróneamente que las pruebas son simplemente ejecutar pruebas, es decir, ejecutar software para encontrar defectos. Sin embargo, ejecutar pruebas es sólo una parte de las pruebas. Hay otras actividades involucradas en las pruebas.

Estas actividades ocurren tanto antes (elementos 1 a 5 a continuación) como después (elemento 7 a continuación) de la ejecución de la prueba. Estos son:

1. Planificación de pruebas
2. Monitoreo y control de pruebas
3. Análisis de pruebas
4. Diseño de pruebas
5. Implementación de pruebas
6. Ejecución de pruebas
7. Finalización de pruebas

Las actividades de prueba se organizan y llevan a cabo de manera diferente en diferentes modelos de ciclo de vida de desarrollo de software (SDLC) (ver Capítulo 2). Además, las pruebas a menudo se consideran una actividad centrada únicamente en la verificación de requisitos  historias de usuarios u otras formas de especificación (es decir, comprobar que el sistema cumple con los requisitos especificados). Pero las pruebas también incluyen la validación, es decir, verificar que el sistema cumple  con los requisitos del usuario y otras necesidades de las partes interesadas en su entorno operativo.

Las pruebas pueden requerir ejecutar el componente o sistema bajo prueba; entonces tenemos lo que se llama pruebas dinámicas. También puede realizar pruebas sin ejecutar el

objeto bajo prueba: dicha prueba se denomina prueba estática. Por lo tanto, las pruebas también incluyen revisiones de productos de trabajo como:

- Requisitos •  
Historias de  
usuarios • Código fuente

Las pruebas estáticas se describen con más detalle en el Capítulo. 3. Las pruebas dinámicas utilizan diferentes tipos de técnicas de prueba (p. ej., de caja negra, de caja blanca y basadas en la experiencia) para derivar casos de prueba y se describen en detalle en el Cap. 4.

Las pruebas no son sólo una actividad técnica. El proceso de prueba también debe planificarse, gestionarse, estimarse, monitorearse y controlarse adecuadamente (ver Capítulo 5). Los evaluadores utilizan ampliamente varios tipos de herramientas en su trabajo diario (ver Capítulo 6), pero es importante recordar que las pruebas son en gran medida una actividad intelectual y inteligente, que requiere que los evaluadores tengan conocimientos especializados, habilidades analíticas, pensamiento crítico y pensamiento sistémico [10, 11].

ISO/IEC/IEEE 29119-1 [1] proporciona información adicional sobre el concepto de prueba de software.

Vale recordar que el testing es un estudio técnico para obtener información sobre la calidad del objeto de prueba:

- Técnico—porque utilizamos un enfoque de ingeniería, usando experimentos, experiencia, técnicas formales, matemáticas, lógica, herramientas (programas de apoyo), mediciones, etc. • Estudio—porque es una búsqueda continua organizada de información

### 1.1.1 Objetivos de la prueba

Las pruebas permiten la detección de fallas o defectos en el producto de trabajo bajo prueba. Esta propiedad fundamental de las pruebas permite alcanzar una serie de objetivos. Los objetivos principales de la prueba.  son:

- Evaluar productos de trabajo como requisitos, historias de usuario, diseños y código • Desencadenar fallas y encontrar defectos • Garantizar la cobertura requerida de un objeto de prueba • Reducir el nivel de riesgo de una calidad inadecuada del software • Verificar si se han cumplido los requisitos especificados • Verificar que un objeto de prueba cumple con los requisitos contractuales, legales y reglamentarios.
- requisitos
- Proporcionar información a las partes interesadas para permitirles tomar decisiones informadas • Generar confianza en la calidad del objeto de prueba • Validar si el objeto de prueba está completo y funciona según lo esperado por el partes interesadas

Diferentes objetivos requieren diferentes estrategias de prueba. Por ejemplo, en el caso de las pruebas de componentes, es decir, probar piezas individuales de una aplicación/sistema (ver Sección 2.2), el objetivo puede ser desencadenar tantas fallas como sea posible para que los defectos que las causan puedan identificarse y solucionarse tempranamente. . También se puede intentar aumentar la cobertura del código mediante pruebas de componentes. Por otra parte , en las pruebas de aceptación (ver Sección 2.2), los objetivos pueden ser:

- Confirmar que el sistema funciona como se espera y cumple con sus requisitos (de usuario).
- Proporcionar a las partes interesadas información sobre los riesgos involucrados en la liberación del sistema en un momento dado.

En las pruebas de aceptación [especialmente las pruebas de aceptación del usuario (UAT)], no esperamos detectar una gran cantidad de fallos o defectos, ya que esto puede provocar una pérdida de confianza por parte de futuros usuarios (consulte la Sección 2.2.1.4). Estas fallas o defectos deben detectarse en etapas anteriores de las pruebas.

## 1.1.2 Pruebas y depuración

Algunas personas piensan que probar se trata de depurar. Sin embargo, es importante recordar que las pruebas y la depuración son dos actividades diferentes. Se supone que las pruebas (especialmente las pruebas dinámicas) revelan fallas causadas por defectos. La depuración,  por otro lado, es una actividad de programación , realizada para identificar la causa de un defecto (falla), corrigiendo el código y verificando que el defecto se haya  solucionado correctamente.

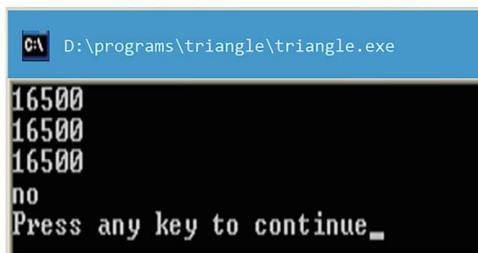
Cuando las pruebas dinámicas detectan una falla, un proceso de depuración típico consistirá en los siguientes pasos:

- Reproducción de fallos (para garantizar que el fallo realmente se produzca y poder desencadenarlo de forma controlada en el proceso de depuración posterior)
- Diagnóstico (encontrar la causa de la falla, como localizar el defecto responsable de la ocurrencia de esa falla)
  - Arreglar la causa (eliminar la causa de la falla, como arreglar un defecto en el código)

La prueba de confirmación posterior (nueva prueba) realizada por el evaluador es para garantizar que la solución realmente solucionó la falla. En la mayoría de los casos, las pruebas de confirmación las realiza la misma persona que realizó la prueba original que reveló el problema. Las pruebas de regresión también se pueden realizar después de la corrección para verificar que la corrección en el código no provocó que el software funcionara mal en otro lugar. Las pruebas de confirmación y las pruebas de regresión se analizan en detalle en la Sección. 2.2.3.

Cuando las pruebas estáticas descubren un defecto, el proceso de depuración consiste simplemente en eliminar el defecto. No es necesario, como en el caso del descubrimiento de fallas en las pruebas dinámicas, realizar la reproducción y el diagnóstico de fallas, porque en las pruebas estáticas, el producto de trabajo bajo prueba no se ejecuta. Es posible que el código fuente relacionado ni siquiera haya sido

Fig. 1.1 Fallo del software



creado. Esto se debe a que las pruebas estáticas no encuentran fallas sino que identifican directamente defectos. Las pruebas estáticas se analizan en detalle en el capítulo. 3.

**Ejemplo** Consideremos una versión simplificada del problema descrito por Myers [10]. Estamos probando un programa que recibe tres números naturales, a, b, c, como entrada. El programa responde “sí” o “no”, dependiendo de si se puede construir un triángulo a partir de segmentos con lados de longitudes a, b, c. El programa tiene la forma de un archivo ejecutable Triangle.exe y toma valores de entrada desde el teclado.

El evaluador preparó varios casos de prueba. En particular, el evaluador ejecutó el programa para los datos de entrada  $a = b = c = 16,500$  (un caso de prueba que involucra la entrada de valores de entrada muy grandes, pero válidos) y recibió el resultado presentado en la Fig. 1.1.

El programa respondió “no”, es decir, afirmó que es imposible construir un triángulo a partir de tres segmentos de 16.500 de longitud cada uno. Esto es un fracaso, porque el resultado esperado es “sí”: es posible construir un triángulo equilátero a partir de esos segmentos. El evaluador informó este defecto al desarrollador. El desarrollador repitió el caso de prueba y obtuvo el mismo resultado. El desarrollador comenzó a analizar el código, que tiene el siguiente aspecto:

```
int principal(int argc, _TCHAR* argv[]) {
    corto a,b,c,d;
    scanf("%d",&a);
    scanf("%d",&b);
    scanf("%d",&c); d=a+b; if
    (abs(ab)<c
    && c<d) printf ("sí");

    demás
    printf("no"); devolver 0;

}
```

El desarrollador afirmó que la condición en la declaración if está definida correctamente, por lo que el defecto debe estar en otra parte. El desarrollador señaló que cuando la suma de las variables a y b se asigna a la variable d, puede haber un problema de desbordamiento del registro. Esto se debe a que las variables a, b, d son del mismo tipo (cortas), por lo que oscilan entre -32,768 y 32,767. Sin embargo, la suma de  $16.500 + 16.500$  es 33.000, más que el máximo

## 1.2 ¿Por qué son necesarias las pruebas?

valor de la variable corta. La operación  $d = a+b$  "rueda sobre el contador" de la variable d, por lo que toma un valor negativo. En este punto, la condición en la declaración if es falsa, porque no es cierto que  $c < d$ .

El desarrollador descubre que la forma más sencilla de arreglar el código es eliminar la variable d, por lo que el recuento de la suma de  $a+b$  se realizará "sobre la marcha". El código corregido tiene el siguiente aspecto:

```
int principal(int argc, _TCHAR* argv[]) {
    corto a,b,c;
    scanf("%d",&a);
    scanf("%d",&b);
    scanf("%d",&c); if
    (abs(ab)<c && c<a+b) printf ("sí");
    demás
    printf("no"); devolver 0;
}
```

El desarrollador verifica la prueba nuevamente para  $a = b = c = 16,500$  y ahora el programa funciona correctamente. La solución funciona, porque cuando el compilador tiene que calcular "sobre la marcha" la suma de  $a+b$ , automáticamente asigna tanta memoria como sea necesaria para esta operación. El defecto de la versión anterior del código era que la suma se escribía en la variable d, que tenía un tipo predefinido y por tanto tenía un límite en el valor superior. El desarrollador informa al evaluador que el defecto se ha solucionado, y el evaluador vuelve a ejecutar la prueba y descubre que esta vez el programa devuelve la respuesta correcta. El evaluador cierra el informe de defectos y reconoce que el defecto se ha solucionado correctamente.

En el ejemplo anterior, todas las actividades del evaluador se sometieron a prueba, mientras que la Las actividades del desarrollador (aparte de la ejecución de pruebas) se incluyeron en la depuración.

## 1.2 ¿Por qué son necesarias las pruebas?

FL-1.2.1 (K2) Ejemplifique por qué las pruebas son necesarias.

FL-1.2.2 (K1) Recordar la relación entre pruebas y aseguramiento de la calidad.

FL-1.2.3 (K2) Distinguir entre causa raíz, error, defecto y falla.

Las pruebas de componentes, sistemas y documentación relacionada respaldan la identificación de defectos de software. Las pruebas también detectan lagunas y otras deficiencias en las especificaciones del software. Por lo tanto, las pruebas pueden ayudar a reducir el riesgo de fallas durante la operación.

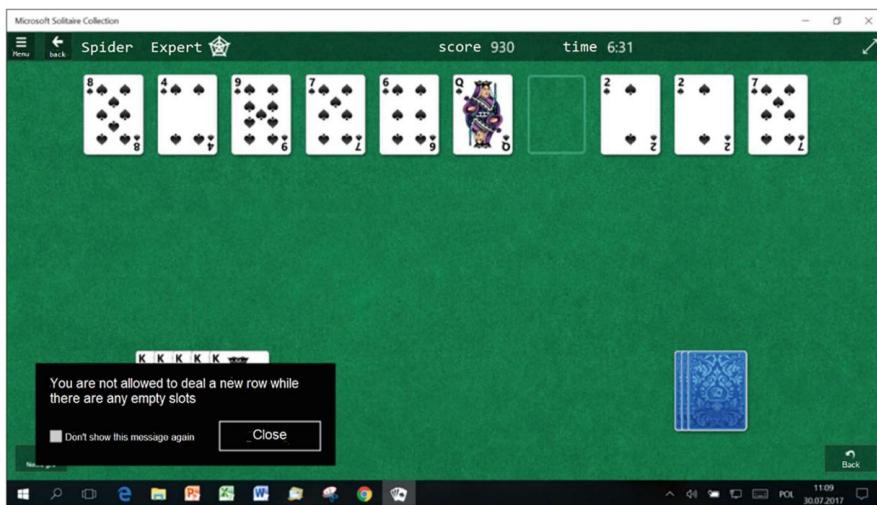


Fig. 1.2 Juego de solitario "Spider"

Cuando se solucionan los defectos detectados, esto contribuye a mejorar la calidad del objeto de prueba. Además, es posible que también se requieran pruebas de software para cumplir con requisitos contractuales o legales o para cumplir con estándares regulatorios.

La mayoría de nosotros nos hemos encontrado con software que no funcionó como esperábamos, incluso fuera del trabajo. A continuación citamos tres ejemplos; aunque algunos ocurrieron hace bastante tiempo, siguen siendo relevantes, ya que los tipos de fallas que describimos también ocurren en el software que se produce hoy.

#### Estudio de caso 1: Juego de solitario "Spider"

El jugador juega con 104 cartas, 54 de las cuales están distribuidas en 10 pilas y 50 se encuentran en el lado derecho de la mesa en filas de 10 cartas. El objetivo del juego es apilar las cartas del rey al as en orden descendente. Cuando llegas a la situación de la figura 1.2: hay 9 cartas sobre la mesa y 30 en grupos (lo que hace un total de  $3 \times 13 = 39$  cartas, 3 de colores completos); Aparece un mensaje: No puedes repartir una nueva fila mientras haya espacios vacíos.

Cuando hay una anomalía en una aplicación siempre se deben responder dos preguntas:

- ¿Cuál es la probabilidad de que ocurra? • ¿Cuál es el impacto de este problema?

En este caso, el impacto es prácticamente nulo: el jugador puede dejar de jugar o hacer una solución: el botón Deshacer devuelve la última pila a la mesa y puede distribuirse entre las columnas libres. La probabilidad de que se produzca tal situación (cuando se juegan los cuatro colores, no sólo las espadas, como en la figura 1.2) es del orden de 10<sup>-6</sup>.

Esto significa que una falla ocurrirá aproximadamente una vez entre un millón de oportunidades para que



Fig. 1.3 Batería de misiles Patriot

ocurrir. Cuando el costo de una falla es cercano a cero y la probabilidad de que ocurra es mínima, los defectos, especialmente en sistemas no críticos, a menudo no se corrigen.

#### Estudio de caso 2: El cohete Ariane 5 El 4

de junio de 1996, el cohete Ariane 5, preparado por la Agencia Espacial Europea, explotó 40 segundos después del despegue en Kourou, Guayana Francesa. Fue el primer vuelo del cohete, después de una década de preparación que costó 7 mil millones de dólares. El cohete y su carga útil valían 500 millones de dólares.

Después de una investigación de dos semanas, resultó que el problema estaba en el software y era muy similar al problema del Triángulo discutido anteriormente. Como parte de la actualización del cohete, el procesador de 16 bits del Ariane 4 fue reemplazado por un procesador de 64 bits. Cuando la velocidad vertical del cohete superó las 32.767 (215 - 1) unidades, el valor de la variable correspondiente se leyó como un número negativo, debido a un desbordamiento del registro. El sistema de control determinó que el cohete había dejado de ascender y caer, por lo que se activó el procedimiento de emergencia, lo que provocó la explosión del cohete como consecuencia del mecanismo de autodestrucción.

Conclusión: como parte de la actualización del sistema, no se realizaron pruebas de regresión adecuadas.

#### Estudio de caso 3: El sistema de misiles Patriot Cada

batería de misiles del sistema Patriot consta de un radar, un puesto de mando (computadora) y lanzadores móviles (ver Fig. 1.3). A cada batería se le asigna un área a proteger; en otras palabras, si un misil apunta a la zona, la batería dispara; si el objetivo del misil está fuera del área, la batería no dispara.

El 25 de febrero de 1991, en Dhahran, Arabia Saudita, durante la primera Guerra del Golfo, un misil Patriot estadounidense no logró interceptar un Scud iraquí que impactó en un cuartel del ejército estadounidense, matando a 28 e hiriendo a más de 100 soldados. Un informe de la Contaduría General,

GAO/IMTEC-92-26, titulado Patriot Missile Defense: Software Problem Lead to System Failure at Dhahran, Arabia Saudita, indica la causa del incidente.

Esto fue un error aritmético. El sistema midió el tiempo en décimas de segundo, utilizando un registro de punto fijo de 24 bits. La fracción 1/10 en el sistema decimal tiene una representación finita (0,1), pero en el sistema binario, es una fracción con expansión infinita (0,000110011001100...). Dado que la computadora representa números en registros finitos, una parte de la fracción debe truncarse, lo que provocará un error de redondeo mínimo. Después de aproximadamente 100 h de funcionamiento, este error de redondeo se acumuló hasta 0,34 s, lo que, a la velocidad del Scud de más de 1676 m/s, dio una distancia de más de 0,5 km. Por lo tanto, el sistema de seguimiento Patriot consideró que el misil iraquí estaba fuera de su alcance y no disparó el misil para destruir al Scud enemigo.

El informe también indicó que el defecto era conocido: su eliminación requería reiniciar el sistema aproximadamente cada 4 h con un tiempo de reinicio de aproximadamente 5 min; Esto estaba indicado en el manual del sistema. Esta sigue siendo una situación típica hoy en día: los usuarios no leen los manuales (si es que existen). Es más, el defecto era conocido y solucionado en la mayoría de los lugares del sistema, aunque no en todos. Esto significa que los desarrolladores copiaron el mismo código en muchos lugares. Esto todavía sucede hoy en día: un defecto que ya se ha solucionado debe volver a analizarse y eliminarse.

Hoy en día, la probabilidad de que se produzca una situación tan catastrófica es menos probable, pero todavía podemos encontrar un software que no funciona correctamente.

### 1.2.1 Contribución de las pruebas al éxito

Las pruebas ayudan a lograr los objetivos acordados dentro del alcance, tiempo, calidad y presupuesto acordados. La contribución de las pruebas al éxito del proyecto se puede considerar en términos de:

- Calidad del producto •
- Calidad del proceso •
- Objetivos del proyecto
- Habilidades interpersonales

#### Las pruebas de calidad

del producto proporcionan un medio rentable para detectar defectos. Las pruebas rigurosas de los sistemas y la documentación pueden reducir el riesgo de problemas (fallas) en el entorno de producción y contribuir a lograr una alta calidad del producto.



La detección y

posterior eliminación de defectos contribuye a la calidad de los componentes o sistemas. También es posible que las disposiciones contractuales, los requisitos legales o los estándares de la industria exijan un nivel adecuado de pruebas.

A través de las pruebas, los usuarios tienen una voz indirecta en el proyecto de desarrollo, lo que permite tener en cuenta sus necesidades durante todo el proceso de desarrollo. Hay muchos ejemplos bien conocidos de software y sistemas (ver arriba) que se han lanzado pero que debido a defectos no han logrado satisfacer las necesidades de las partes interesadas. Sin embargo, con los enfoques de prueba correctos, aplicados por expertos en el

niveles de prueba correctos y en las fases correctas del ciclo de desarrollo de software: la incidencia de tales problemas se puede reducir.

La verificación y validación del software por parte de los evaluadores antes del lanzamiento permite la detección de fallas y facilita la eliminación de defectos relacionados (depuración). Esto facilita las pruebas, reduce el riesgo y aumenta la probabilidad de que el software satisfaga las necesidades de las partes interesadas y cumpla con los requisitos. Por tanto, aumenta la probabilidad de éxito del proyecto.

#### Las pruebas de

calidad del proceso pueden contribuir indirectamente a la calidad del proceso de fabricación. Esto es muy importante porque se sabe que la calidad final de un producto está muy influenciada por la calidad del proceso mediante el cual se desarrolla el producto. La calidad del proceso se puede aumentar de varias maneras. Por ejemplo, la introducción de la automatización de pruebas mejora la eficiencia del proceso de lanzamiento del sistema. El uso de pruebas basadas en riesgos optimiza los gastos en pruebas. Los datos recopilados a través del monitoreo de pruebas (ver Sección 5.3) ayudan a identificar lugares en el proceso de desarrollo que necesitan mejoras (por ejemplo, demasiado tiempo para reparar defectos, demasiados defectos introducidos en una determinada fase del ciclo de desarrollo, etc.).

#### Las pruebas de

los objetivos del proyecto pueden ayudar a aumentar la probabilidad de alcanzar los objetivos del proyecto. Por ejemplo, el uso de pruebas estáticas al principio del proyecto reduce los costos de mantenimiento del software y mejora la eficiencia del desarrollador al reducir el tiempo dedicado a la corrección de defectos. Como resultado, existe una mayor probabilidad de que el producto se complete a tiempo y dentro del presupuesto.

Las pruebas proporcionan un medio para evaluar directamente la calidad de un objeto de prueba en varias etapas del SDLC. Estas medidas se utilizan como parte de una actividad de gestión de proyectos más amplia, contribuyendo a las decisiones para pasar a la siguiente etapa del SDLC, como la decisión de lanzamiento.

#### Habilidades

interpersonales Un "efecto secundario" de las prácticas de prueba es aumentar las habilidades de los miembros del equipo y otras partes interesadas. Por ejemplo, realizar revisiones de código (p. ej., en forma de recorridos; consulte la Sección 3.2.4) aumenta la comprensión del código y permite a los desarrolladores menos experimentados mejorar sus habilidades de programación y diseño). La estrecha cooperación entre los probadores y los arquitectos de sistemas durante la fase de diseño del trabajo permite a ambas partes comprender mejor el proyecto.

## 1.2.2 Pruebas y garantía de calidad (QA)

Las pruebas a menudo se equiparan erróneamente con el aseguramiento de la calidad (QA). Estos son dos procesos separados (aunque relacionados) que se incluyen en el término más amplio "Gestión de la calidad" (QM). La Gestión de la Calidad abarca todas las actividades destinadas a guiar y supervisar las actividades de calidad de una organización.

Los dos elementos básicos de la gestión de la calidad son:

- Seguro de Calidad /
- Control de Calidad

#### Aseguramiento de la

calidad El aseguramiento de la calidad se centra en establecer, implementar, monitorear, mejorar y adherirse a procesos relacionados con la calidad. Funciona sobre la base de que si se sigue correctamente un buen proceso, se generará un buen producto. Cuando los procesos relevantes se implementan correctamente, se contribuye a la prevención de defectos y aumenta la confianza en que se alcanzarán los niveles adecuados de calidad del producto del trabajo. El control de calidad, cuando se aplica al desarrollo y mantenimiento de software, también debe aplicarse a las pruebas de software, que forman parte de cada una de estas actividades. Además, el uso del análisis de causa raíz para detectar las causas de los defectos y la aplicación de las lecciones aprendidas en reuniones retrospectivas para mejorar los procesos también son importantes para un control de calidad eficaz.

#### Control de calidad El

control de calidad (QC) abarca una variedad de actividades, incluidas actividades de prueba, que apoyan el logro de niveles de calidad apropiados. Las actividades de prueba son una parte importante del proceso general de desarrollo o mantenimiento de software. La realización adecuada del control de calidad, especialmente las actividades de prueba, es importante para el aseguramiento de la calidad, y el aseguramiento de la calidad respalda las pruebas adecuadas. Esto se debe a que los resultados de las pruebas se utilizan tanto para el control de calidad como para el aseguramiento de la calidad. En el control de calidad, se utilizan para corregir defectos, mientras que en el control de calidad, brindar retroalimentación sobre qué tan bien se están desempeñando los procesos de desarrollo y prueba.

La Tabla 1.1 resume las diferencias clave entre los procesos de garantía de calidad y control de calidad.

### 1.2.3 Errores, defectos, fallas y causas fundamentales

El enfoque ISTQB® distingue tres etapas que conducen a un resultado anormal, relacionadas con tres conceptos muy importantes, que son:

- Error (también conocido como error): una acción humana que causa un resultado incorrecto.
- Defecto (error, falla): una imperfección o defecto en un producto de trabajo que involucra incumplimiento de los requisitos
- Fallo: un evento en el que un componente o sistema no logra realizar una función requerida dentro de un contexto específico.

Como resultado de un error humano en el código de software u otro producto de trabajo relacionado, se puede introducir un defecto en el producto de trabajo. Tenga en cuenta que es importante prestar atención a esta terminología durante el examen, ya que es contraintuitiva y se incluye en el vocabulario que se requiere que conozca el examen. Por lo general, en el lenguaje común, un defecto se denomina error; a menudo decimos: "Hay un error en este lugar del código". De

Tabla 1.1 Comparación de los procesos de aseguramiento de la calidad y control de calidad

Categoría	Seguro de calidad	Control de calidad
Descripción general	Implementar procesos, metodologías y estándares para asegurar que el producto desarrollado cumpla con los estándares de calidad requeridos.	Realizar actividades para verificar que el producto desarrollado cumpla con los estándares de calidad requeridos.
Objetivo	Mejorando el proceso de fabricación.	Mejora del producto mediante detección de fallos y defectos.
Tipo de proceso	Preventivo (prevención de defectos), proactivo Control (detección de defectos), reactivo	
Ejemplos de actividades	Implementación de procesos, por ejemplo, gestión de defectos, gestión de cambios, lanzamiento de software; auditorías de calidad; mediciones de procesos y productos; verificación de la correcta implementación y ejecución de procesos; formación de los miembros del equipo; selección de herramientas	Ánalisis estático de la documentación del proyecto; revisiones de código; análisis, diseño, implementación de casos de prueba; pruebas dinámicas; escribir y ejecutar scripts de prueba; informe de defectos; usar herramientas para apoyar las pruebas

Desde el punto de vista de la terminología ISTQB®, esto es incorrecto. Puede haber un defecto en el programa. Un error siempre se refiere a un error humano. Ejecutar un fragmento de código donde hay un defecto puede causar o no una falla.

Ejemplo Consideremos un ejemplo sencillo que muestra por qué la ejecución de una línea de programa que contiene un defecto no conduce necesariamente a un fallo. Supongamos que un fragmento de código calcula el valor promedio de las mediciones dividiendo su suma por el número de mediciones tomadas. En el código, se utiliza la siguiente declaración para hacer esto:

```
Valor Promedio := SumaDeValores / NúmeroDeMedidas
```

En esta línea hay un defecto: el programa no controla si el denominador de la fracción que se cuenta es cero. Esto se debe a que no se permite dividir por cero y realizar dicha operación puede resultar en la terminación del programa.

En una situación en la que el número de mediciones sea positivo, la ejecución de la declaración anterior no causará ningún problema: el programa funcionará perfectamente y devolverá el resultado correcto. La prueba que obliga a ejecutar esta afirmación con un número positivo de medidas pasará y no notaremos ningún signo de fallo. Sin embargo, si esta línea se ejecuta cuando hay cero mediciones (el valor de la variable NumberOfMeasurements es 0), se producirá un fallo y lo notaremos.

Ejemplo Consideremos ahora una situación un poco más sofisticada, aunque todavía relativamente simple. El programa Count que se muestra a continuación cuenta cuántas entradas de una matriz denominada T son ceros. Nos referimos a los elementos de la matriz T por sus índices; por ejemplo, T[3] denota el elemento que aparece en la matriz en la posición número 3. Supongamos además que los elementos de la matriz (como en muchos lenguajes de programación reales) están indexados desde cero, por lo que el primer elemento de la matriz es T[0], el siguiente es

Fig. 1.4 Error, defecto, fallo



$T[1]$ , y así sucesivamente. El siguiente es el pseudocódigo de nuestro programa, que contiene un defecto: el bucle que pasa por los siguientes elementos de la matriz comienza a pasar desde el elemento  $T[1]$  en lugar de  $T[0]$ :

programa Contar entrada:

una matriz T (con elementos  $T[0]$ ,  $T[1]$ , ...,  $T[n]$ ) salida: número de celdas en T que contienen cero

número := 0 para cada  $i = 1, 2, \dots, n$  hazlo si  $T[i] == 0$  entonces número :=

número + 1 devuelve

número

Tenga en cuenta que la línea con el defecto (el bucle "para cada") se ejecutará para cada ejecución del código. Sin embargo, la ocurrencia de una falla (mal resultado) dependerá de si la celda  $T[0]$  contiene cero u otro número. En el primer caso ( $T[0]=0$ ), el resultado será incorrecto: el programa contará una celda menos. Por ejemplo, si  $T = (0, 3, 2, 0, 1)$ , el programa devolverá 1 en lugar de 2. Sin embargo, en el segundo caso ( $T[0]\neq0$ ), el resultado será correcto. Por ejemplo, si  $T = (5, 2, 0, 1, 0, 0)$ , el programa devolverá 3, que es el resultado correcto a pesar de ejecutar una línea con un defecto. Entre otras cosas, el diseño de pruebas consiste en tener en cuenta este tipo de situaciones y asegurarse de que el conjunto de pruebas sea capaz de detectar defectos similares en el código.

Un error que resulte en la introducción de un defecto en un producto de trabajo puede causar un error que resulte en la introducción de un defecto en otro producto de trabajo relacionado. La ejecución de código que contiene un defecto puede provocar fallos, pero, como vimos en el ejemplo anterior, esto no sucede necesariamente con todos los defectos. Algunos defectos provocan fallos, por ejemplo, sólo después de una intervención estricta o debido a la aparición de determinadas condiciones previas, que pueden ocurrir muy raramente o nunca. La aparición consecutiva de tres factores (error, defecto, falla) causa el mal funcionamiento observado del producto bajo prueba (Fig. 1.4).

Los errores pueden ocurrir por muchas razones:

- Presión de tiempo •
- Falibilidad humana • Falta de experiencia o habilidades insuficientes de los miembros del equipo del proyecto •
- Problemas con el intercambio de información entre las partes interesadas •
- Ambigüedades con respecto a la comprensión de los requisitos y la documentación del proyecto • Complejidad del código, diseño, arquitectura, problema a resolver y/o tecnología que se utiliza

- Malentendidos sobre las interfaces dentro y entre sistemas, especialmente cuando hay un gran número de ellos • Uso de tecnologías nuevas y desconocidas

Las fallas, a su vez, pueden ser causadas no necesariamente por errores humanos sino también por factores ambientales, tales como:

- Radiación •
- Campo electromagnético •
- Contaminación

Estos factores pueden causar fallas en el software integrado o afectar el rendimiento del software al cambiar las condiciones operativas del hardware.

El primer “error” de la historia El primer error de la historia se encontró en 1947. En la Universidad de Harvard en Cambridge, Massachusetts, los investigadores descubrieron que su computadora, Mark II, experimentaba fallas constantemente. Al investigar el hardware de la computadora, hicieron un descubrimiento inesperado: una polilla había quedado atrapada en su interior. El insecto había causado averías en la electrónica del ordenador. La figura 1.5 muestra un extracto del registro original del sistema, incluido el “error” real (polilla), la causa de las fallas.

No todos los resultados inesperados de las pruebas significan fallas. Un resultado falso positivo puede ser el resultado de errores relacionados con la ejecución de la prueba, defectos en los datos de la prueba, el entorno de la prueba, otro software de prueba, etc. Los resultados falsos positivos se informan como defectos que en realidad no existen. Problemas similares pueden causar la situación opuesta: un resultado falso negativo, es decir, una situación en la que las pruebas no logran detectar un defecto que deberían detectar (consulte la Tabla 1.2).

Formalmente, un resultado falso positivo<sup>1</sup> es un resultado positivo de la prueba (una prueba fallida), cuando en realidad la prueba debería haberse aprobado. Un ejemplo de tal situación es cuando el evaluador malinterpreta la especificación y define incorrectamente el resultado esperado. Un resultado falso negativo es un resultado negativo de la prueba (una prueba aprobada), cuando en realidad la prueba debería haber fallado. Un ejemplo de tal situación es el cambio de requisitos entre ciclos de prueba. En el segundo ciclo, el requisito modificado debería hacer que la prueba falle, pero en la prueba, el resultado esperado permaneció sin cambios y la prueba aún se pasa.

Los casos de prueba deben diseñarse para evitar el enmascaramiento de defectos, es decir, situaciones en las que la aparición de un defecto impide la detección de otro defecto o la aparición de dos defectos cancela su efecto mutuo. Se utilizan varias mejores prácticas.

---

<sup>1</sup>La terminología “falso positivo” y “falso negativo” proviene del campo de la analítica médica. Un resultado negativo significa, por ejemplo, la ausencia de un virus en el organismo, y un resultado positivo, su presencia. Entonces la analogía con las pruebas es la siguiente: una prueba es positiva cuando detecta una falla (la presencia de un defecto) y negativa cuando no la detecta.

9/9	
0800	Antan started
1000	stopped - antan ✓ 13°C (033) MP-MC (033) PRO-2 cosine
	1. 12700 9.037 847 025 1. 12700 9.037 846 995 cosine 2. 13047 6415 (23) 4.615 925 059 (-2) 2. 13047 6415
	Relays 6-2 in 033 fault special speed test in relay "10.00 test." Relays changed
1100	Started Cosine Tape (Sine check)
1525	Started Multi Adder Test.
1545	 Relay #70 Panel F (moth) in relay.
1620	First actual case of bug being found. antangal started.
1700	closed down.

Fig. 1.5 El primer "error" de la historia ([www.atlasobscura.com/places/grace-hoppers-bug](http://www.atlasobscura.com/places/grace-hoppers-bug))

Tabla 1.2 Posibles resultados de las pruebas en el contexto de su corrección

Posibles resultados de la prueba	RESULTADO INTERPRETADO		
	PRUEBA APROBADA	PRUEBA FALLADA	PRUEBA FALLADA
REAL RESULTADO DE LA PRUEBA	PRUEBA APROBADA	Correcto (resultado negativo)	resultado falso positivo
	PRUEBA FALLADA	resultado falso negativo	Correcto (resultado positivo)

en el diseño de la prueba que ayudan al evaluador a evitar tales situaciones (más sobre esto en el Capítulo 4), pero en general, los defectos enmascarados son difíciles de detectar.

En relación con las consideraciones anteriores, un factor importante es el análisis de la causa raíz del defecto, la razón principal que resultó en el defecto. Este

La razón puede ser la ocurrencia de una situación específica o la ocurrencia de un error humano.

Analizar un defecto para identificar la causa raíz ayuda a reducir la aparición de defectos similares.

defectos en el futuro. Análisis de causa raíz, centrándose en las causas raíz más importantes.

de defectos, puede conducir a mejoras en el proceso, lo que a su vez puede traducirse en mayores Reducciones de defectos en el futuro.

## 1.3 Principios de prueba

Ejemplo Un defecto en el código provoca un cálculo incorrecto del descuento en compras al por mayor en la tienda electrónica, lo que genera quejas de los clientes. El código defectuoso se escribió basándose en una historia de usuario, pero el propietario del producto malinterpretó las reglas de cálculo de descuento y escribió mal la historia.

En este ejemplo:

- Las quejas de los clientes son las consecuencias. • El cálculo incorrecto de los descuentos es un fracaso. • El defecto es una fórmula de cálculo incorrecta implementada en el código. • La falta de conocimiento del propietario del producto es la causa fundamental. • La causa raíz es el resultado de un error del propietario del producto.

## 1.3 Principios de prueba

FL-1.3.1 (K2) Explicar los siete principios de prueba.

Hay muchas “leyes” o “principios” diferentes relacionados con las pruebas que son válidos independientemente del contexto del proyecto o del tipo de producto que se esté desarrollando, y han surgido en los últimos 60 años. El programa de estudios de Foundation Level describe siete de estos principios. Estos principios básicos de las pruebas son:

1. Las pruebas muestran la presencia, no la ausencia, de defectos.
2. Es imposible realizar pruebas exhaustivas.
3. Las pruebas tempranas ahorrarán tiempo y dinero.
4. Los defectos se agrupan.
5. Las pruebas se desgastan.
6. Las pruebas dependen del contexto.
7. Falacia de ausencia de defectos.

### 1. Las pruebas muestran la presencia, no la ausencia de defectos

Este famoso principio fue formulado por Edsger Dijkstra, un informático danés, en la conferencia “Técnicas de ingeniería de software” celebrada en Roma en 1969 [12]. Si bien las pruebas pueden mostrar que existen defectos, no podemos probar que no hay defectos en el programa que se está probando. Por lo tanto, las pruebas sólo reducen la probabilidad de que queden defectos no identificados en el software. El hecho de que no se detecten defectos no es una prueba de la corrección del sistema bajo prueba. Este principio tiene serias implicaciones: las pruebas son de naturaleza negativa, es decir, muestran que algo no funciona, no que todo esté bien. Esto conlleva algunas implicaciones psicológicas importantes, que se analizarán más adelante.

Es interesante observar que la afirmación de Dijkstra puede formularse como un teorema matemático estricto y preciso; esto está relacionado con el hecho de que ciertos problemas informáticos, como el llamado problema de detención (que puede tratarse como un

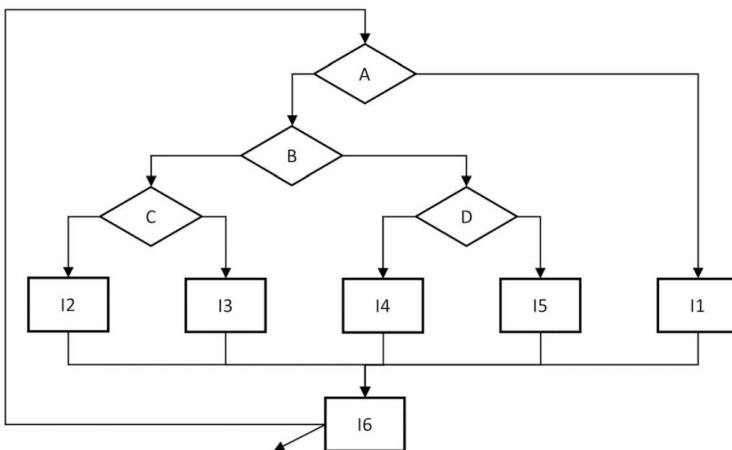


Fig. 1.6 Gráfico de flujo de control del fragmento de código a probar

defecto del algoritmo), son irresolvibles. En general, es imposible responder a la pregunta de si un programa determinado se detendrá finalmente ante cada entrada posible. Por lo tanto no somos capaces de detectar este tipo de defecto (la posibilidad de bucle del programa) en cada caso.

## 2. Las pruebas exhaustivas son imposibles

Consideremos el siguiente ejemplo [10]. Considere un fragmento de código para probar con cuatro decisiones (A, B, C, D) y seis declaraciones (I1, I2, I3, I4, I5, I6), que se muestran en la figura 1.6. El código se ejecuta en un bucle, con un máximo de 20 iteraciones de bucle. En la primera iteración, tenemos cinco caminos posibles P1-P5 (el símbolo “!” indica la falsedad de la decisión, por ejemplo, “!A” significa que la decisión A es falsa; la verdad de la condición resulta con un control flujo indicado por la flecha derecha y la falsedad por la flecha izquierda):

P1: A I1 I6

P2: !A B D I5 I6 P3: !

AB !D I4 I6 P4: !A !BC

I3 I6 P5: !A !B !C I2 I6

Dos iteraciones de bucle pueden realizar 25 caminos posibles:

P1 P1; P1 P2; P1 P3; P1 P4; P1 P5; P2 P1;

P2 P2; P2 P3; P2 P4; P2 P5; P3 P1; P3 P2;

P3 P3; P3 P4; P3 P5; P4 P1; P4 P2; P4 P3;

P4 P4; P4 P5; P5 P1; P5 P2; P5 P3; P5 P4;

P5 P5.

Tres iteraciones del bucle dan  $5^3 = 125$  caminos posibles. En general, si el bucle se ejecuta  $n$  veces, tenemos  $5^n$  posibles realizaciones de ruta. Como asumimos que el bucle se ejecutará como máximo 20 veces, el número de posibles realizaciones diferentes de las rutas del flujo de control será:

$$5 \times 52 \times 53 \times \dots \times 520 = 119.209.289.550.780 > 10^{14}$$

Si suponemos que necesitamos sólo 0,001 s para probar una ruta, necesitaríamos aproximadamente 3860 años para probar todos los caminos. ¡Desafortunadamente no tenemos tanto tiempo!

Observe que en el ejemplo anterior analizamos un problema muy simple (y restringido en términos del número de iteraciones del bucle). En la vida real, para probar completamente (a fondo) una aplicación determinada, sería necesario verificar:

- Cada valor de entrada posible para cada variable (incluido el resultado y el trabajo variables) •

Cada secuencia posible de ejecución del programa • Cada configuración posible de hardware/software • Todas las formas posibles, pero generalmente inimaginables, de uso del producto probado por El usuario final

Las pruebas exhaustivas deben significar que una vez completadas, todos estarán seguros de que no se producirán fallas, pero esto es imposible (excepto en casos triviales) [13]. En lugar de pruebas exhaustivas, se deben utilizar el análisis de riesgos y la priorización para guiar las pruebas. Esto significa que el papel del probador de software es esencial en el ciclo de desarrollo de software. Además, los evaluadores deben dominar determinadas técnicas de prueba.

### 3. Las pruebas tempranas ahorrarán tiempo y dinero

Las pruebas tempranas a veces se denominan desplazamiento a la izquierda. Las actividades de prueba deben comenzar lo antes posible para el software bajo prueba. Esto ahorra tiempo y dinero, porque los defectos que se solucionan en las primeras etapas del proceso no causarán defectos posteriores en los productos de trabajo derivados, como el diseño o el código. Esto, a su vez, aumenta la productividad de la programación, reduce los costos y el tiempo de desarrollo y puede tener un impacto positivo en el esfuerzo de prueba requerido [14]. El costo general de la calidad se reducirá porque habrá menos fallas más adelante en el ciclo de desarrollo. Las pruebas siempre deben estar dirigidas a alcanzar objetivos bien definidos. Incluso si no estamos preparados para realizar pruebas dinámicas (porque, por ejemplo, el software aún no se ha implementado), podemos realizar pruebas estáticas, revisiones de documentación, revisiones de diseño, etc.

La Figura 1.7 muestra la famosa curva de Boehm, que ilustra la relación entre el coste de arreglar un defecto y el tiempo transcurrido hasta su hallazgo (asumimos que el defecto se introdujo en la fase de Análisis). Esta curva es exponencial, lo que significa que cuanto más tarde se encuentre un defecto, mayor será el aumento del coste de repararlo. Las investigaciones modernas sugieren que esta curva no aumenta tan rápidamente, pero, sin embargo, su aumento en cada caso es significativo, lo que sugiere claramente que encontrar defectos temprano es muy rentable.

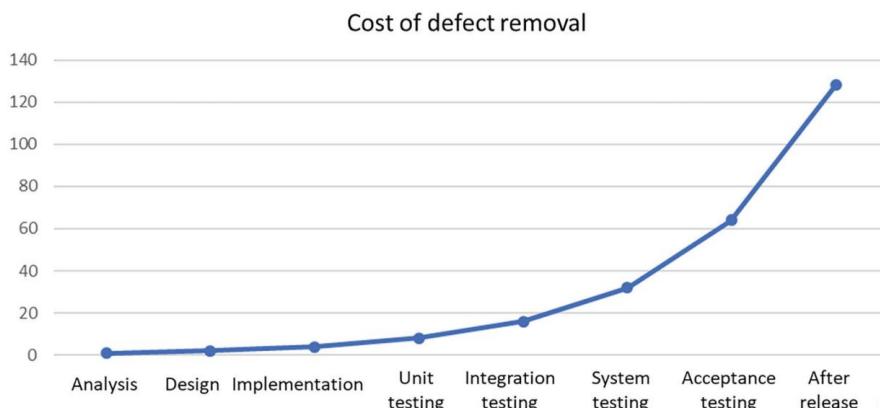


Fig. 1.7 Costo de la eliminación de defectos en función del tiempo

#### 4. Los defectos se agrupan

Los defectos no se distribuyen uniformemente ni en el software ni a lo largo del tiempo. La mayoría de los defectos encontrados en las pruebas previas al lanzamiento de software o que causan fallas de producción se encuentran en una pequeña cantidad de componentes [15]. Como resultado, los grupos de defectos previstos y los grupos de defectos realmente observados durante la fase de prueba u operativa son una parte importante del análisis de riesgos que se realiza para guiar los esfuerzos de prueba en consecuencia. Esto no significa que haya menos defectos en los otros componentes; es solo que durante las pruebas (o en producción), nos enfocamos en las rutas más relevantes para el usuario, y ahí es donde encontramos la mayoría de los defectos.

Cuando los defectos se acumulan, se aplica un principio bien conocido, la llamada regla de Pareto, que establece que un pequeño número de causas provocan un gran número de efectos. En terminología de pruebas, por ejemplo, podría traducirse así: alrededor del 20% de los componentes contienen alrededor del 80% de los defectos.

La Figura 1.8 muestra una distribución típica del número de defectos en los componentes (histograma) y el número acumulado de defectos (línea). Los componentes se clasifican en orden descendente según el número de defectos. Normalmente, sólo unos pocos componentes tienen una gran cantidad de defectos y el resto tiene pocos. El gráfico muestra un ejemplo de la aplicación de la regla de Pareto para optimizar el esfuerzo. Si conocemos (por ejemplo, mediante estimación o referencia a datos históricos) la distribución del número esperado de defectos, como en el cuadro antes mencionado, podemos preocuparnos de probar un pequeño número de los componentes más defectuosos, de modo que podamos detectar la mayoría de los defectos en poco tiempo. Por ejemplo, los componentes analíticos y de informes contienen un total de 320 defectos, es decir, el 20% de los componentes (2 de 10) contienen aproximadamente el 70% de todos los defectos previstos (320 de 460).

Si durante las pruebas vemos que con un esfuerzo de prueba comparable detectamos muchos más defectos en el componente A que en el componente B, esto significa que probablemente haya aún más defectos no detectados en el componente A. Un enfoque racional basado en la

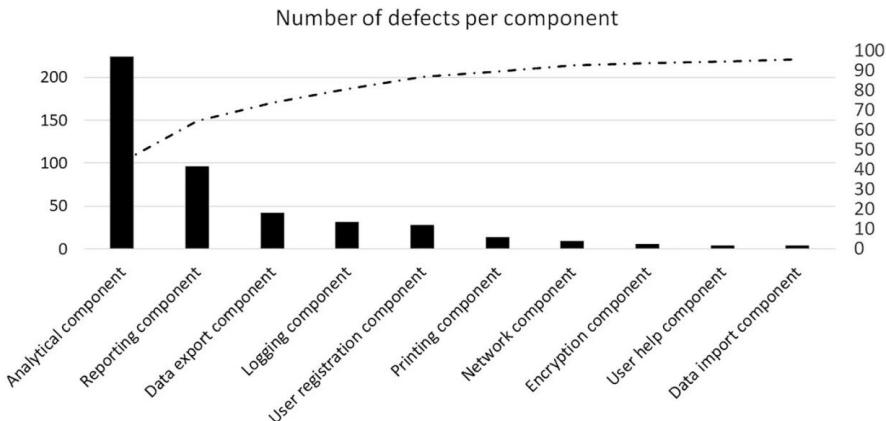


Fig. 1.8 Ejemplo de distribución tipo Pareto

El principio de “los defectos se agrupan” requeriría centrarse aún más en el componente A que en el componente B en esta situación.

##### 5. Las pruebas se desgastan (o la “paradoja de los pesticidas”)

Si las mismas pruebas se repiten una y otra vez, entonces, después de los cambios que conducen a la eliminación de los defectos detectados, no se encuentran más defectos nuevos [16]. Para superar este fenómeno, los casos de prueba deben revisarse y modificarse periódicamente. Además, para probar partes nuevas o corregidas del sistema bajo prueba, se deben crear o ejecutar nuevas pruebas con un conjunto diferente de datos de prueba.

Las pruebas no modificadas pierden su capacidad de detectar defectos con el tiempo. A veces, por ejemplo, con las pruebas de regresión automatizadas, la paradoja de los pesticidas puede ser beneficiosa porque nos permite confirmar que el número de defectos asociados con las pruebas de regresión es pequeño (en el caso de las pruebas de regresión, más bien nos importa que las pruebas siempre pasen).

##### 6. Las pruebas dependen del contexto

Este es un principio bastante obvio: las pruebas deben realizarse de manera diferente en diferentes situaciones. Prestamos atención a otra cosa cuando probamos sistemas críticos para la vida, a otra cosa cuando probamos sistemas bancarios (aquí lo más importante es la precisión funcional (por ejemplo, el cálculo correcto de los intereses sobre el capital)) y a otra cosa cuando probamos juegos de ordenador, aquí no. Los atributos de prueba funcionales como el rendimiento (fluidez del juego) o la usabilidad (interfaz de juego interesante) probablemente serán más importantes. Sin embargo, en los juegos también se debe prestar atención a la corrección funcional (por ejemplo, el caballo de dos cabezas en la figura 1.9).

El “contexto” mencionado en el principio tiene un alcance muy amplio. Se refiere, entre otras cosas, a la naturaleza del software que se está desarrollando, el dominio empresarial dentro del cual se está desarrollando el software, los riesgos del proyecto y del producto, los aspectos funcionales.



Fig. 1.9 Caballo de dos cabezas en un juego de computadora

y requisitos no funcionales, características del grupo de usuarios objetivo, todo tipo de riesgos y sus consecuencias relacionados con el incorrecto funcionamiento del software, regulaciones legales, prácticas, normas y estándares existentes en la materia, etc.

Una consecuencia de este principio es que no existe un enfoque único para las pruebas [17]. Las pruebas, por su propia naturaleza, son un proceso intelectual que requiere conocimientos, habilidades y, a menudo, mucha intuición y creatividad.

#### 7. Falacia de ausencia de defectos

Algunas organizaciones todavía esperan que los evaluadores puedan ejecutar todas las pruebas posibles y detectar todos los defectos posibles, pero los principios (1) y (2) muestran que esto es imposible. También es erróneo creer que simplemente encontrar y corregir una gran cantidad de defectos garantizará una implementación exitosa del sistema, porque incluso una aplicación libre de defectos (verificación correcta) puede no cumplir con los requisitos del usuario (validación incorrecta).

Básicamente, este principio dice que dentro del proceso de prueba, la verificación por sí sola no es suficiente; aún se necesita validación, mediante la cual nos aseguramos de que el programa cumpla con los requisitos del cliente, y no solo con los supuestos técnicos que tiene el proyecto.

equipo formado en base a los requisitos [14]. Podemos crear un producto perfecto, libre de defectos y completamente inútil desde el punto de vista del usuario.

#### 1.4 Actividades de prueba, software de prueba y funciones de prueba

FL-1.4.2 (K2) Explicar el impacto del contexto en el proceso de prueba.

FL-1.4.3 (K2) Diferenciar el software de prueba que soporta las actividades de prueba.

FL-1.4.4 (K2) Explicar el valor de mantener la trazabilidad.

FL-1.4.5 (K2) Comparar los diferentes roles en las pruebas.

##### 1.4.1 Actividades y tareas de prueba

No existe un proceso de prueba de software único para todos. Sin embargo, existen actividades de prueba típicas y esenciales necesarias para alcanzar los objetivos establecidos. Estas actividades forman el proceso de prueba. Las actividades de prueba que se incluyen en este proceso de prueba, cómo se implementan y cuándo se llevan a cabo generalmente se definen en la estrategia o enfoque de prueba de la organización como parte de la planificación de pruebas para una situación específica (consulte el Capítulo 5).

Es una buena práctica definir criterios de cobertura mensurables  la base de la prueba (para cada nivel de prueba o tipo de prueba bajo consideración). En la práctica, pueden actuar como los llamados indicadores clave de desempeño (KPI) que favorecen el desempeño de actividades específicas y permiten que el equipo de prueba demuestre el logro de los objetivos de la prueba (por ejemplo, los criterios de cobertura pueden requerir al menos una prueba para cada base de prueba). artículo).

El proceso de prueba puede estar definido formalmente o no. En situaciones típicas, consta de los siguientes grupos de actividades:

1. Planificación de pruebas
2. Monitoreo y control de pruebas
3. Análisis de pruebas
4. Diseño de pruebas
5. Implementación de pruebas
6. Ejecución de pruebas
7. Finalización de pruebas

Por ejemplo, el desarrollo de software en metodologías ágiles implica pequeñas iteraciones de diseño, construcción y prueba de software que se llevan a cabo de forma continua, respaldadas por una planificación continua. Por lo tanto, las pruebas también se realizan de forma iterativa y continua dentro de este enfoque de fabricación. Incluso en ciclos de vida de desarrollo secuenciales, grupos de actividades del proceso de prueba, presentadas anteriormente como secuenciales, en el proceso real pueden ocurrir de manera iterativa, superponerse, ocurrir simultáneamente (por ejemplo, en pruebas exploratorias) o omitirse. Las relaciones temporales entre estas actividades siempre dependen del proyecto específico. Los factores contextuales que afectan la selección del proceso de prueba de una organización incluyen:

- Ciclo de vida de desarrollo de software y metodologías de proyecto utilizadas • Niveles de prueba y tipos de prueba considerados • Riesgos del producto y riesgos del proyecto • Dominio empresarial • Requisitos contractuales y regulatorios • Limitaciones operativas

- Presupuestos y recursos
- Horarios

- Complejidad del dominio • Políticas y prácticas de prueba de la organización • Normas/estándares internos y externos requeridos

A continuación se analizarán las actividades dentro de cada grupo del proceso de prueba que se muestra en la Fig. 1.10. La Sección 1.4.3, a su vez, discutirá los productos de trabajo producidos dentro de estas actividades.

#### Planificación de pruebas: actividades

La planificación de pruebas implica definir los objetivos de la prueba y el enfoque de la prueba para lograrlos dentro de las limitaciones impuestas por el contexto. La planificación de pruebas se explica con más detalle en la Sección. 5.1.

Las actividades típicas de planificación de pruebas incluyen:

- Definir los objetivos de la prueba.
- Identificar las actividades de prueba necesarias para cumplir la misión del proyecto y cumplir con los objetivos de la prueba
- Definir un enfoque para lograr los objetivos de prueba dentro de los límites establecidos por el contexto
- Determinar técnicas de prueba y tareas de prueba apropiadas. • Formular un cronograma de ejecución de pruebas. • Definir métricas.

Los planes de prueba se pueden revisar en función de los comentarios del monitoreo y control de pruebas. actividades. La planificación de pruebas debe ser una actividad continua.

#### Monitoreo de pruebas y control de pruebas: actividades

El monitoreo de pruebas es la comparación continua del progreso de las pruebas reales y planificado utilizando métricas específicamente definidas para este propósito en el plan de pruebas. El control de pruebas es la toma proactiva de las acciones necesarias para alcanzar los objetivos marcados en el plan de pruebas (teniendo en cuenta sus posibles actualizaciones). Estas acciones se toman sobre la base de información de seguimiento. El seguimiento y control de las pruebas se explican con más detalle en la Sección. 5.3.

El progreso del plan de pruebas se comunica a las partes interesadas en informes escritos o verbales de progreso de las pruebas (consulte la Sección 5.3.2), que incluyen cualquier desviación notable del plan, así como los impedimentos de las pruebas y las soluciones alternativas relacionadas.

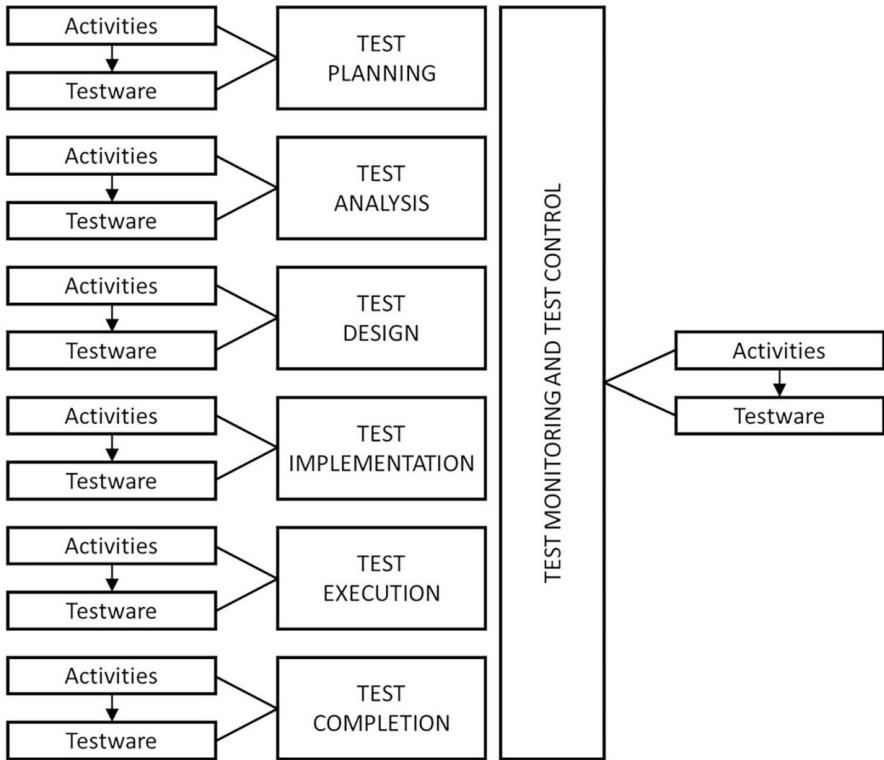


Fig. 1.10 Actividades y productos de trabajo en el proceso de prueba.

Un elemento que respalda el monitoreo y control de las pruebas es la evaluación de los criterios de salida (a menudo llamados Definición de Hecho (DoD) en un enfoque ágil) del plan de pruebas, que puede incluir:

- Verificar los resultados de las pruebas y los registros de pruebas con respecto a los criterios de cobertura especificados. • Estimar el nivel de calidad de un componente o sistema, en función de los resultados de las pruebas y registros de prueba
- Determinar si son necesarias más pruebas (si las pruebas realizadas hasta el momento no alcanzan el nivel de cobertura de riesgo del producto original; esto implica escribir y ejecutar pruebas adicionales) • Informar a las partes interesadas sobre el progreso del plan de pruebas • Escribir informes de progreso de las pruebas

#### Análisis de pruebas: actividades La

tarea del análisis de pruebas es obtener la base de la prueba y analizarla para identificar características comprobables, definir las condiciones de prueba asociadas y determinar "qué probar" (en términos de criterios de cobertura mensurables). Los objetivos generales de la prueba se transforman en condiciones de prueba específicas.

La base de prueba se refiere a cualquier documentación o información que describa cómo debe funcionar el software y sirva como base o referencia para diseñar y ejecutar casos de prueba. Los ejemplos comunes de base de prueba incluyen:

- Especificación de requisitos • Especificación de diseño • Casos de uso

- Historias de usuario
- Código fuente • Reglas de negocio

Una condición de prueba  es cualquier tipo de propiedad, característica o atributo del software que se puede verificar mediante pruebas. Las condiciones de prueba son ideas iniciales para realizar pruebas. Por lo general, no contienen los resultados esperados. Por ejemplo, en el caso de probar un cajero automático, se pueden definir las siguientes condiciones de prueba: "verificar que el cajero automático reconozca correctamente las tarjetas de pago", "verificar que el cajero automático acepte el código PIN correcto ingresado por el usuario", "verificar que el usuario puede imprimir el saldo de la cuenta", etc. Las condiciones de prueba son la base para derivar casos de prueba y datos de prueba.

Las actividades de análisis de pruebas verifican que los requisitos:

- Son consistentes • Están expresados correctamente • Están completos • Son comprobables (son adecuados para derivar criterios de aceptación) • Están listos para comenzar a desarrollar el software (la Definición de Listo—DoR) • No necesitan preparación adicional y por lo tanto pueden usarse como fuente de estimación • Reflejar adecuadamente las necesidades de los clientes, usuarios y otras partes interesadas

Las actividades típicas de análisis de pruebas incluyen:

- Familiarizarse con la base de prueba que define el comportamiento funcional y no funcional deseado de un componente o sistema. • Análisis de información de diseño e implementación, como diagramas o documentos que describen la arquitectura del sistema o software, diagramas de flujo de control, diagramas UML [18], diagramas de relaciones entre entidades, especificaciones de interfaz o productos de trabajo similares; Estos documentos definen la estructura de un componente o sistema. • Análisis de la implementación del componente o sistema en sí: código, metadatos, consultas de bases de datos e interfaces. • Análisis de informes de análisis de riesgos, que pueden cubrir funciones funcionales, no funcionales y

aspectos estructurales de un componente o sistema

- Evaluar la capacidad de prueba de la base de la prueba para identificar tipos comunes de defectos: ambigüedades, omisiones, inconsistencias, inexactitudes, contradicciones, instrucciones redundantes (innecesarias).
- Identificar las características y conjuntos de características que se van a probar.
- Definir las condiciones de prueba para características individuales y priorizarlas basándose en el análisis de la base de prueba, teniendo en cuenta parámetros funcionales, no funcionales y estructurales, otros factores comerciales y técnicos y niveles de riesgo.

- Crear trazabilidad bidireccional entre los elementos de la base de prueba y sus asociados.

condiciones de prueba especificadas

El uso de técnicas de prueba de caja negra, caja blanca y basadas en la experiencia puede ser útil como parte del análisis de la prueba para reducir la probabilidad de pasar por alto condiciones de prueba importantes y definir condiciones de prueba más precisas y exactas (consulte el Capítulo 4). Las condiciones de prueba más formales suelen representarse como los llamados modelos de prueba (por ejemplo, diagramas de transición de estados, tablas de decisión, diagramas de flujo de control).

Identificar defectos en la base de prueba como resultado del análisis de la prueba es un beneficio importante, especialmente cuando no se realiza una revisión separada de la base de prueba. Las actividades de análisis de pruebas no sólo pueden verificar que los requisitos sean consistentes, expresados adecuadamente y completos, sino también verificar que los requisitos abordan adecuadamente las necesidades de los clientes, usuarios y otras partes interesadas.

#### Diseño de pruebas: actividades

Durante el diseño de pruebas, las condiciones de prueba se transforman en casos de prueba de alto nivel (lógico), colecciones de dichos casos de prueba y en otro software de prueba. El diseño de la prueba responde a la pregunta "cómo realizar la prueba".

Transformar las condiciones de prueba en casos de prueba a menudo implica identificar elementos de cobertura de prueba y utilizar técnicas de prueba (ver Capítulo 4). Los elementos de cobertura de prueba también sirven como pautas para determinar los datos de entrada del caso de prueba y en algunas situaciones pueden incluso definir estos datos (por ejemplo, valores identificados por el análisis de valores límite).

El diseño de la prueba precede a la implementación de la prueba y, a veces, se pueden realizar simultáneamente, pero es importante distinguir entre las dos actividades. Al diseñar pruebas antes de su implementación, puede identificar defectos en el diseño de la prueba, lo que reduce el riesgo de perder tiempo y esfuerzo en la implementación.

Las actividades de diseño de pruebas incluyen:

- Diseñar (conjuntos de) casos de prueba de alto nivel y priorizarlos • Identificar los datos de prueba necesarios • Identificar los requisitos para el entorno de prueba • Identificar las herramientas y elementos de infraestructura necesarios • Crear trazabilidad bidireccional entre la base de prueba, las condiciones de prueba, Casos de prueba, y procedimientos de prueba (ampliando la matriz de trazabilidad)
- Identificación de defectos en la base de prueba.

#### Implementación de pruebas: actividades

Durante la implementación de la prueba, el evaluador crea y/o finaliza el software de prueba necesario para la ejecución de la prueba, lo que incluye, entre otros, transformar casos de prueba de alto nivel (lógicos) en casos de prueba de bajo nivel (concretos), ensamblar pruebas casos en procedimientos de prueba, creando scripts de prueba automatizados, adquiriendo datos de prueba y, a menudo, implementando un entorno de prueba. Por lo tanto, mientras que el diseño de la prueba responde a la pregunta "cómo realizar la prueba", la implementación de la prueba responde a la pregunta "¿tenemos todo lo que necesitamos para ejecutar las pruebas?"

Las actividades de implementación de pruebas incluyen:

- Cuando sea necesario: hacer que los casos de prueba de alto nivel (lógicos) sean más concretos especificando datos de prueba detallados (por ejemplo, un caso de prueba de alto nivel que requiere "Ingresar una edad entre 18 y 65" puede resultar en múltiples casos de prueba concretos que requieren "ingresar una edad entre 18 y 65 años"). Ingrese una edad de 18", "Ingrese una edad de 42" y "Ingrese una edad de 65")
- Desarrollar procedimientos de prueba y priorizarlos. • Crear conjuntos de pruebas (basados en procedimientos de prueba) y scripts de prueba automatizados (si se realizan pruebas). se utiliza la automatización)
- Organizar conjuntos de pruebas en un cronograma de ejecución de pruebas para garantizar que todo el proceso funciona eficientemente
- Crear un entorno de prueba que incluya, si es necesario, objetos simulados (controladores, stubs), virtualización de servicios, simuladores y otros elementos de infraestructura, y verificar que se haya configurado correctamente.
  
- Preparar los datos de la prueba y verificar que se hayan cargado correctamente en la prueba ambiente
- Verificar y actualizar la trazabilidad entre bases de prueba, condiciones de prueba, casos de prueba, procedimientos de prueba y conjuntos de prueba

#### Ejecución de la prueba: actividades

Durante la ejecución de la prueba,  los conjuntos de pruebas se ejecutan de acuerdo con la ejecución de la prueba programada. Dentro de esta fase se realizan las siguientes actividades:

- Registrar los datos de identificación y versión de elementos de prueba, objetos de prueba, herramientas de prueba y otro software de prueba.
- Realizar pruebas manualmente o con herramientas, incluidas pruebas de humo<sup>2</sup> o de cordura. • Comparar los resultados reales de las pruebas con los esperados. • Analizar anomalías para determinar sus causas probables (por ejemplo, defectos en el código, falsos positivos)
- Informar defectos, basado en fallas observadas. • Registrar los resultados de la ejecución de la prueba (aprobada, fallida, prueba de bloqueo). • Repetir las actividades de prueba necesarias (pruebas de confirmación, ejecución de una prueba revisada, prueba de regresión)
- Verificar y actualizar la trazabilidad bidireccional entre la base de prueba y todos los software de prueba utilizado

#### Finalización de la prueba: actividades

En la fase de finalización de la prueba,  se recopilan los datos de las actividades de prueba completadas para consolidar las lecciones aprendidas, el software de prueba y otra información relevante. Esto se hace cuando se alcanzan los hitos del proyecto, tales como:

- Entrega del sistema de software para su funcionamiento. • Finalización (o cancelación) del proyecto.

---

<sup>2</sup> La prueba de humo es una prueba rápida y sencilla para comprobar la correcta implementación de la funcionalidad básica.

- Finalización de una iteración de un proyecto ágil (por ejemplo, después de una demostración o en una reunión retrospectiva)
- Finalización de un nivel de prueba •  
Finalización del trabajo en la versión de mantenimiento

Como parte de la realización de la prueba, se realizan las siguientes actividades:

- Comprobar que todos los informes de defectos estén cerrados y crear solicitudes de cambio o Elementos de la cartera de productos para cualquier defecto no resuelto
- Identificar y archivar cualquier caso de prueba que pueda ser útil en el futuro. • Entregar el software de prueba al equipo de operaciones, a otros equipos del proyecto u otros. partes interesadas que podrían beneficiarse de su uso
- Llevar el entorno de prueba a un estado acordado. • Analizar las actividades de prueba completadas para identificar las lecciones aprendidas e identificar mejoras para futuras iteraciones, lanzamientos o proyectos
- Crear un informe sobre la finalización de las pruebas y distribuirlo a las partes interesadas.

#### 1.4.2 Proceso de prueba en contexto

Las pruebas no se realizan de forma aislada. Es un proceso que apoya el proceso de desarrollo establecido dentro de una organización. Las pruebas también son un proceso patrocinado por las partes interesadas, cada una de las cuales tiene requisitos o expectativas del producto final.

Por lo tanto, el proceso de prueba debe estar alineado con el proceso de desarrollo de software establecido por la organización, y cómo se construye en detalle dependerá de una serie de factores contextuales. Estos factores incluyen, en particular:

- Partes interesadas (necesidades, expectativas, requisitos, incluidos los requisitos comerciales). para el producto, voluntad de cooperar con el equipo de prueba, etc.)
- Miembros del equipo (habilidades, conocimientos, nivel de experiencia, disponibilidad, necesidades de capacitación, relaciones con otros miembros del equipo, etc.) • Dominio comercial (riesgos de producto identificados, necesidades del mercado, condiciones legales específicas). ciones, etc.)
- Factores técnicos (arquitectura del proyecto, tecnología utilizada, etc.) • Restricciones del proyecto (alcance del proyecto, tiempo, presupuesto disponible, recursos disponibles, riesgos del proyecto,
- etc.) • Factores organizacionales (estructura organizacional, políticas existentes, incluidas las pruebas políticas, prácticas utilizadas, etc.)
- Ciclo de vida del desarrollo de software (prácticas de ingeniería, métodos de desarrollo, etc.)
- Herramientas (disponibilidad, usabilidad, cumplimiento, etc.) • Políticas (datos, privacidad, cookies, etc.)

Los factores anteriores afectarán significativamente cómo se organiza y llevados a cabo en el proyecto. En particular, afectarán a:

- Estrategia de prueba
- Técnicas de prueba utilizadas
- Grado de automatización de la prueba
- Nivel de cobertura de prueba requerido para los requisitos y riesgos identificados • Nivel de detalle y tipo de documentación de prueba a desarrollar • Nivel de detalle de los informes de progreso de las pruebas • Nivel de detalle de los informes de defectos

#### 1.4.3 Software de prueba

El proceso de prueba produce software de prueba  que son los productos de trabajo asociados con las pruebas (ver Fig. 1.9). Pueden tener diferentes tipos y diferentes nombres en diferentes organizaciones. Una buena referencia para probar productos de trabajo es la norma internacional ISO/IEC/IEEE 29119-3 [3]. Vale la pena señalar que muchos productos de trabajo de prueba se pueden crear y administrar utilizando herramientas de administración de pruebas y herramientas de administración de defectos.

##### Planificación de pruebas: productos de trabajo

Los productos típicos del trabajo de planificación de pruebas son:

- Plan de pruebas • Registro de riesgos • Criterios de entrada y criterios de salida

Un registro de riesgos es una lista de riesgos identificados por el equipo, junto con información sobre su probabilidad, impacto y cómo serán mitigados (ver Sección 5.2). El registro de riesgos y los criterios de entrada y salida suelen formar parte del plan de pruebas. En proyectos más grandes, puede haber más de un plan de prueba, por ejemplo, varios planes relacionados con niveles de prueba específicos (plan de prueba de integración del sistema, plan de prueba de aceptación, etc.).

El plan de prueba contiene información sobre la base de la prueba, a la que se vincularán otros productos del trabajo de prueba a través de información de trazabilidad bidireccional. Define los criterios de salida (Definición de Hecho) que se utilizarán para el monitoreo y control de la prueba.

Los planes de prueba se pueden ajustar en función de los comentarios del monitoreo y control de pruebas. Es importante recordar que la planificación es un proceso continuo; no termina en la fase inicial del proyecto. Los ajustes de los planes pueden ocurrir en cualquier punto del ciclo de vida del desarrollo de software, pero cualquier ajuste debe ser aprobado por las partes interesadas.

---

Ejemplo de plantilla de plan de prueba de aceptación (basado en [19]).

---

ID del plan de prueba de aceptación del sistema

XYZ, autor, fecha, historial de revisiones 1.

Introducción (propósito del documento, base para su desarrollo, descripción del sistema)

2 Enfoque de prueba

2.1 Supuestos preliminares 2.1.1 Criterios

de entrada para las pruebas (documentación disponible y requerida, cronograma de ejecución de pruebas aceptado, condiciones logísticas y organizativas, pruebas preliminares completadas, documentación disponible de las pruebas preliminares)

2.1.2 Criterios de entrada para iteraciones de prueba posteriores (defectos solucionados de iteraciones anteriores, posiblemente sin defectos de prioridad suficientemente alta, modificaciones necesarias en la documentación de prueba en la medida que resulte de la iteración anterior, logrando una cobertura adecuada)

2.1.3 Tipos de pruebas de aceptación (describir los tipos de pruebas de aceptación realizadas; identificar dónde se realizan; identificar quién las realizará, por ejemplo, pruebas alfa, pruebas beta, pruebas de aceptación del usuario, pruebas de aceptación operativa; identificar las funciones de las partes interesadas en la realización de estas pruebas (por ejemplo, el papel del contratista, el papel de los usuarios clave)

2.2 Organización de la prueba

2.2.1 Recursos de personal (roles requeridos, calificaciones, composición de equipos, descripciones de roles)

2.2.2 Procedimiento de prueba (procedimientos generales y específicos que describen el curso de las pruebas de aceptación: requisitos previos, cómo proporcionar información, como datos de prueba por parte del contratista)

2.2.3 Clasificación de defectos (descripción de la clasificación de defectos, por ejemplo, crítico/grave/baja prioridad, con descripción de las acciones en caso de un defecto de una categoría determinada)

2.2.4 Procedimiento de notificación de defectos (definición del proceso, herramientas utilizadas)

2.2.5 Manejo de defectos (el proceso de manejar informes de defectos y resolverlos)

2.2.6 Informes de progreso (descripción de las reglas de informes de trabajo, tipos de informes)

2.2.7 Condiciones para la aceptación de diferentes tipos de pruebas 2.3 Entornos de

prueba (descripción de entornos, infraestructura y datos de prueba, cómo se adquieren y mantienen; división de datos en diccionario y datos operativos, procedimientos para manejar datos)

2.4 Cronograma del proceso de

prueba 2.5 Tipos y niveles de pruebas realizadas (p. ej., pruebas de regresión, pruebas basadas en escenarios, pruebas exploratorias)

3 Pruebas

3.1 Casos de prueba (lista jerárquica de casos de prueba relacionados con seguimiento de escenarios de prueba, riesgos y otros elementos relevantes; puede incluirse en documentos separados)

3.2 Calendario de ejecución de pruebas (el orden de ejecución del caso de prueba)

3.3 Escenarios de prueba (especificación de escenarios; puede incluirse en un documento separado)

4 Anexos (p. ej., extractos de contratos de clientes, informes de muestra y otra documentación necesaria)

---

Para obtener más información sobre el propósito y el contenido del plan de prueba, consulte la Sección [5.1.1](#).

Monitoreo y control de pruebas: productos de trabajo

Los productos de trabajo esenciales en este grupo son:

- Informes de progreso de las pruebas (ver Sección [5.3.2](#)) • Documentación de directivas de control (ver Sección [5.3](#)) • Información de riesgos (ver Sección [5.2](#))

Los informes de progreso de las pruebas se crean de forma continua y/o a intervalos regulares, según lo acordado con las partes interesadas. Todos los informes de progreso de las pruebas deben contener detalles relevantes para la audiencia sobre el progreso actual de las pruebas, incluido un resumen de los resultados de la ejecución de las pruebas a medida que estén disponibles. Los informes siempre deben adaptarse a las necesidades de

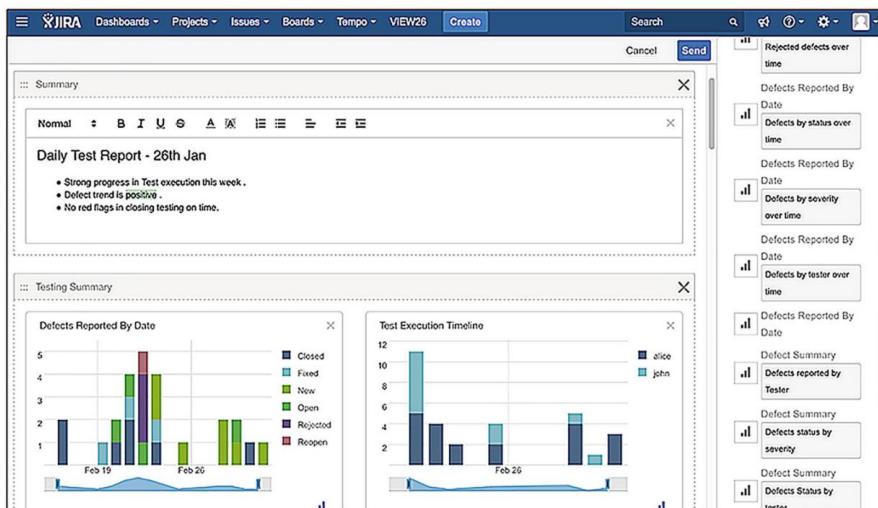


Fig. 1.11 Ejemplo de informe de prueba diario (fuente: [community.atlassian.com](http://community.atlassian.com))

audiencias específicas. Además, los informes de progreso de las pruebas deben incluir cuestiones de gestión del proyecto, información sobre las tareas completadas y la asignación y el consumo de recursos.

La información de riesgos se puede almacenar en el registro de riesgos del proyecto o localmente en el plan de prueba.

Ejemplo En la Fig. 1.11 se muestra un ejemplo de un informe de progreso de la prueba . Es un informe de prueba diario con datos comparados con los días anteriores. El gráfico de la izquierda describe la distribución del número de defectos en intervalos de tiempo específicos, desglosados por categoría (cerrados, arreglados, nuevos, abiertos, rechazados, reabiertos). El gráfico de la derecha muestra la ejecución diaria de la prueba por parte de dos evaluadores (Alice y John).

#### Análisis de pruebas: productos de trabajo

Los productos de trabajo típicos del análisis de prueba son:

- Condiciones de prueba (priorizadas)
- Criterios de aceptación (ver Sección 4.5.2)
- Informes de defectos en la base de prueba (si no se corrigen directamente)

Las condiciones de prueba suelen estar definidas y priorizadas. Los criterios de aceptación pueden considerarse como el equivalente a las condiciones de prueba en metodologías ágiles de desarrollo de software. Para pruebas exploratorias, los productos de trabajo pueden contener cartas de prueba.

El análisis de pruebas también puede dar como resultado la detección y el informe de defectos en la base de la prueba, lo que significa que los productos del trabajo de análisis de pruebas también son informes de defectos.

Ejemplo Al probar la funcionalidad de inicio de sesión de un servicio, podemos identificar las siguientes condiciones de prueba:

- Inicio de sesión correcto (usuario existente, inicio de sesión correcto y contraseña correcta) • Inicio de sesión incorrecto (usuario existente pero contraseña incorrecta) • Inicio de sesión incorrecto con bloqueo de cuenta (ingresar la contraseña incorrecta tres veces, lo que debería resultar en el bloqueo de la cuenta) • Incorrecto iniciar sesión (usuario inexistente)

#### Diseño de pruebas: productos de trabajo

Los productos del trabajo del diseño de pruebas son en particular:

- Casos de prueba de alto nivel (lógicos) •

Elementos de cobertura •

Requisitos de datos de prueba •

Diseño del entorno de prueba

A menudo es una buena práctica diseñar casos de prueba de alto nivel, que no incluyan valores de datos de entrada específicos ni resultados esperados. Los casos de prueba de alto nivel se pueden reutilizar varias veces en diferentes ciclos de prueba con diferentes datos de prueba, documentando adecuadamente el alcance del caso de prueba. Idealmente, para cada caso de prueba, existe una trazabilidad bidireccional entre ese caso de prueba y las condiciones de prueba que cubre. La ventaja de utilizar casos de prueba (lógicos) de alto nivel es que permiten flexibilidad. Por otro lado, estos casos de prueba ponen en peligro la reproducibilidad.

Entre los productos de trabajo de la fase de diseño de pruebas también puede estar el diseño y/o identificación de los datos de prueba necesarios, el diseño del entorno de prueba y la identificación de infraestructura y herramientas. El grado de documentación de estos productos de trabajo puede variar.

Las condiciones de prueba definidas en la fase de análisis de prueba se pueden refinar durante el diseño de la prueba.

Ejemplo El siguiente ejemplo muestra un caso de prueba con pasos de ejecución de prueba (pasos de prueba). Tenga en cuenta que este es un ejemplo de un caso de prueba de bajo nivel, porque contiene los datos de prueba detallados (dirección del sitio web concreto, inicio de sesión y contraseña).

TC N° 20.002.11	Autor: Joan Smith
Prioridad: media Función:	Fecha: 11.07.2020
registro Descripción:	
intentar iniciar sesión con el nombre de usuario y contraseña correctos usando la tecla Intro en lugar de hacer clic en el botón de inicio de sesión Requisitos previos: el usuario tiene un nombre de usuario y contraseña Datos de paso/prueba Abra	
la página de inicio de sesión <a href="http://www.our.com">www.our.com</a> .	Resultado Esperado
Ingrese el inicio de sesión "johnsmith".	Iniciar sesión aceptado
Ingrese la contraseña "contraseña123".	Contraseña aceptada, botón de inicio de sesión activo predeterminado
Presione Entrar.	Iniciar sesión en el servicio

Condiciones de salida: usuario que inició sesión, el hecho de iniciar sesión se guarda en la base de datos junto con la hora de inicio de sesión

Implementación de pruebas: productos de trabajo

Dentro de este grupo de actividades se crean los siguientes productos de trabajo:

- Casos de prueba de bajo nivel (concretos);
- Procedimientos de prueba (incluido el orden en que se ejecutan); • Scripts de prueba automatizados • Conjuntos de prueba
- Datos de prueba • Programa de ejecución de pruebas • Elementos del entorno de prueba

Los datos de prueba se utilizan para asignar valores específicos a los datos de entrada y los resultados esperados de los casos de prueba. Estos valores específicos, junto con pautas específicas para su uso, transforman casos de prueba de alto nivel en casos de prueba ejecutables de bajo nivel. Normalmente, un caso de prueba de alto nivel da como resultado múltiples casos de prueba de bajo nivel. El mismo caso de prueba de alto nivel se puede ejecutar utilizando diferentes datos de prueba para diferentes versiones del objeto de prueba.

Los resultados esperados específicos asociados con datos de prueba específicos se identifican utilizando el oráculo de prueba.

Ejemplos de componentes del entorno de prueba son:

- Objetos simulados (p. ej., stubs, controladores) • Simuladores • Virtualización de servicios

En las pruebas exploratorias, algunos productos de trabajo relacionados con el diseño y la implementación de las pruebas se pueden crear durante la ejecución de la prueba, aunque el grado en que las pruebas exploratorias están documentadas y rastreables hasta elementos específicos de la base de la prueba varía ampliamente.

En ocasiones, el producto del trabajo de este grupo es una descripción del uso de herramientas (por ejemplo, para la virtualización de servicios). Los productos de trabajo más típicos de esta fase son los scripts de prueba automatizados.

Ejemplo Supongamos que estamos probando la función multiplicar( $x, y$ ), que toma dos números enteros como entrada y devuelve su producto. Durante el análisis de la prueba, se definió la siguiente condición de prueba: "verificar la exactitud de la multiplicación que involucra el valor cero". El diseño de la prueba transforma esta condición de prueba en tres casos de prueba que implican la multiplicación por cero:

- TC1: el primer parámetro es cero y el segundo parámetro es diferente de cero. • TC2: el segundo parámetro es cero y el primer parámetro es diferente de cero. • TC3: ambos parámetros son iguales a cero.

Para estos casos se definieron los siguientes insumos y productos esperados:

- TC1:  $x = 0, y = 10$ , salida esperada: 0 • TC2:  $x = 10, y = 0$ , salida esperada: 0 • TC3:  $x = 0, y = 0$ , salida esperada: 0

El siguiente script de prueba automatizado es un ejemplo de implementación de tres casos de prueba, TC1, TC2 y TC3, en Java utilizando la biblioteca JUnit (utilizada para ayudar a crear pruebas de componentes). La declaración afirmarEquals comprueba si los dos primeros parámetros tienen el mismo valor. En nuestro caso, estamos comprobando si los resultados de los productos de  $10 \times 0$ ,  $0 \times 10$  y  $0 \times 0$  serán realmente iguales a 0.

```
importar estática org.junit.jupiter.api.Assertions.assertEquals; importar org.junit.jupiter.api.Test;

clase pública MisPruebas {
    @Prueba
    vacío público multiplicarValueNumbersBy0Day0() {
        Pruebas MiClase = nueva MiClase(); // pruebas de MiClase

        // aserciones que implementan los casos PT1, PT2, PT3 afirmarEquals(0,
        tests.multiply(10, 0), "10 x 0 debe ser 0"); afirmarEquals(0, tests.multiply(0, 10), "0 x
        10 debe ser 0"); afirmarEquals(0, tests.multiply(0, 0), "0 x 0 debe ser 0");

    }
}
```

#### Ejecución de pruebas: productos de trabajo

Los productos típicos del trabajo de ejecución de pruebas son:

- Registros de prueba
- Documentación del estado de los procedimientos de prueba
- Informes de defectos (ver Sección 5.5)
- Documentación que indique qué se utilizó en la prueba (por ejemplo, objetos de prueba, herramientas de prueba, y software de prueba)

En situaciones ideales, también se puede informar del estado de los elementos individuales de la base de prueba. Estos informes son factibles gracias a la trazabilidad bidireccional de los procedimientos de prueba correspondientes (ver Sección 1.4.4). Es posible, por ejemplo, indicar qué requisitos se han probado en su totalidad, cuáles no han superado las pruebas y/o implican defectos y cuáles aún no se han probado en su totalidad. Esto hace posible comprobar si se han cumplido los criterios de cobertura de las pruebas y presentar los resultados de las pruebas en informes de una manera que las partes interesadas puedan entender.

Ejemplo La Figura 1.12 muestra la pantalla de informe de la ejecución de dos pruebas unitarias, testCaseA y testCaseB, utilizando la biblioteca JUnit5. Los colores de los iconos al lado de los nombres de las pruebas simbolizan los resultados de las mismas: el color verde significa que la prueba fue aprobada y el color rojo no pasó. En nuestro caso, todos los colores son verdes, lo que significa que ambos casos de prueba pasan.

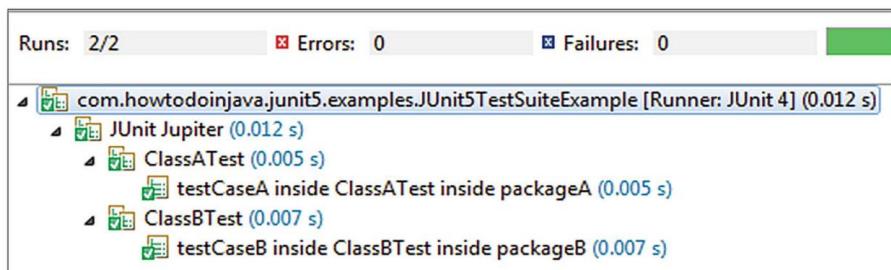


Fig. 1.12 Resultado de la ejecución de la prueba de componentes en JUnit (fuente: [howtodoinjava.com](http://howtodoinjava.com))

Finalización de la prueba: productos de trabajo

Los productos de trabajo típicos de la finalización de la prueba son:

- Informe de finalización de la prueba (consulte la Sección 5.3.2)
- Elementos de acción para mejorar proyectos o iteraciones posteriores (por ejemplo, elementos de acción retrospectivos transformados en elementos del Backlog del Producto para iteraciones futuras)
- Solicitudes de cambio (por ejemplo, como elementos de una cartera de productos)

Los informes de finalización de pruebas se crean cuando se alcanzan hitos individuales.

Estos informes deben incluir información detallada sobre el progreso del proceso de prueba hasta la fecha, un resumen de los resultados de la ejecución de la prueba e información sobre cualquier desviación del plan y acciones correctivas.

#### 1.4.4 Trazabilidad entre la base de prueba y el software de prueba

Para garantizar un monitoreo y control efectivo de las pruebas, es importante establecer y mantener un mecanismo de trazabilidad entre cada elemento de la base de la prueba y sus correspondientes productos de trabajo de prueba (p. ej., condiciones de prueba, casos de prueba, riesgos, etc.) durante todo el proceso. proceso de prueba. Una trazabilidad eficiente permite:

- Evaluación de la cobertura de las pruebas •
- Analís del impacto del cambio • Realización de auditorías de pruebas • Cumplimiento de criterios relacionados con la gestión de TI • Creación de informes de estado de pruebas y resumen de finalización de pruebas fáciles de entender informes
- Presentar el estado de los elementos básicos de las pruebas (requisitos para los cuales las pruebas han sido aprobadas, falladas o están esperando ser ejecutadas) • Proporcionar a las partes interesadas información sobre los aspectos técnicos de las pruebas en una forma que puedan entender • Proporcionar la información necesaria para evaluar calidad del producto, capacidades del proceso,
- y el progreso del proyecto en comparación con los objetivos comerciales

Tabla 1.3 Trazabilidad matriz

	Riesgo 1	Riesgo 2	Riesgo 3	Riesgo 4
TC001X	X			
TC002	X			
TC003		X		
TC004X			X	X

Los criterios de cobertura pueden funcionar como KPI para demostrar el logro de la prueba. objetivos (ver Apartado 1.1.1). Por ejemplo, aprovechando la capacidad de realizar un seguimiento desde:

- Casos de prueba a requisitos, podemos verificar que la cobertura de los casos de prueba del se cumplen los requisitos
- Los resultados del caso de prueba se relacionan con los riesgos; se puede evaluar el nivel de riesgo residual en el objeto de prueba.

Ejemplo Considere la matriz de trazabilidad para los casos de prueba y los riesgos identificados que se muestran en la Tabla 1.3.

Una "X" en la fila correspondiente al caso de prueba T y en la columna correspondiente al riesgo R indica que un caso de prueba T cubre un riesgo R. Por ejemplo, TC001 cubre los riesgos 1 y 2, TC003 cubre el Riesgo 3, y así sucesivamente. Supongamos que TC001, TC002 y TC004 pasa, mientras que TC003 falla. Esto podría significar que el Riesgo 3 no está cubierto (si queremos ese riesgo está cubierto cuando pasan todas las pruebas relacionadas) o, por ejemplo, que está cubierto en 50% (si definimos la cobertura de riesgo como el porcentaje de casos de prueba relacionados que pasan).

#### 1.4.5 Roles en las pruebas

El programa de estudios de Foundation Level distingue dos roles fundamentales en las pruebas: una prueba rol de gestión (gerencial) y un rol de prueba (técnico). Las actividades y tareas asignados a estos roles dependen de una serie de factores, como el contexto de la proyecto o producto, el modelo adoptado del ciclo de vida de desarrollo, las habilidades de personas y la organización del trabajo en equipo.

La persona (o equipo) en el rol de gestión de pruebas es responsable de implementar el proceso de prueba, organizando el trabajo del equipo de prueba y dirigiendo las actividades de prueba. Las tareas de gestión de pruebas se centran principalmente en actividades relacionadas con:

- Planificación de pruebas
- Monitoreo y control de pruebas.
- Finalización de la prueba

Una persona en el rol de prueba tiene la responsabilidad general de la ingeniería (tecnología). aspecto técnico) de las pruebas. Las tareas de prueba se centran principalmente en:

- Análisis de pruebas
- Diseño de pruebas
- Implementación de pruebas
- Ejecución de pruebas

La forma en que se define y comprende la función de gestión de pruebas depende, en particular, del modelo de ciclo de vida de desarrollo de software adoptado. A veces, esta función la desempeña una sola persona, normalmente denominada director de pruebas. En proyectos basados en metodologías ágiles, por ejemplo, algunas tareas de gestión pueden dejarse en manos de todo el equipo (autoorganizado). Por otro lado, las tareas que afectan a varios equipos o a toda la organización pueden ser manejadas por administradores de pruebas fuera del equipo de desarrollo.

También es importante distinguir entre roles y posiciones dentro de la empresa. Por lo general, un puesto se asigna permanentemente a una persona específica: cada miembro del equipo suele estar empleado en un puesto específico. Por otro lado, cada miembro del equipo puede desempeñar diferentes roles en distintos momentos. En particular, la función de gestión de pruebas en las pruebas puede ser la de líder de equipo, director de proyecto, director de calidad, etc. En proyectos más grandes, el director o coordinador de pruebas puede coordinar varios equipos de pruebas, dirigidos por líderes de pruebas o probadores principales.

La función de prueba la desempeña cualquier persona que realice cualquier actividad de prueba en un momento dado. En este sentido, por ejemplo, un desarrollador en el momento de la prueba de un componente (unitario) o un cliente que realiza una prueba de aceptación entra en el rol de prueba. También es posible que una persona desempeñe funciones de prueba y gestión al mismo tiempo.

A continuación se analizarán en detalle dos funciones.

#### Función de gestión de pruebas La

forma en que se llevan a cabo las funciones del director de pruebas depende del ciclo de desarrollo del software.

Las tareas típicas de un director de pruebas son:

- Desarrollar o revisar estrategias y políticas de pruebas. • Planificación de proyectos de pruebas sensibles al contexto (ver Sección 5.1) .
  - Creación y actualización de planes de prueba.
  - Planificación de iteraciones y lanzamientos en proyectos ágiles.
  - Elección del método de prueba
  - Definición de criterios de entrada y criterios de salida.
  - Introducir métricas apropiadas para medir el progreso de las pruebas y evaluar la calidad de las pruebas y del producto.
  - Definición de niveles de prueba y ciclos de prueba.
  - Estimar el tiempo, el esfuerzo y el coste de las pruebas.
  - Priorización de pruebas
- Gestión de riesgos (ver Sección 5.2) • Monitorear los resultados de las pruebas, verificar el estado de los criterios de salida (definición de hecho) y realizar el control de las pruebas, por ejemplo, ajustando los planes de acuerdo con los resultados y el progreso de las pruebas (ver Sección 5.3)
- Supervisión de procesos:
  - Gestión de la configuración (ver Apartado 5.4)
  - Gestión de defectos (ver Apartado 5.5)
- Adquisición de recursos • Coordinación de la estrategia de prueba y el plan de prueba con los gerentes de proyecto y otros partes interesadas

- Presentar el punto de vista de los evaluadores como parte de otras actividades del proyecto, como planificación de la integración
- Iniciar procesos para análisis de pruebas, diseño de pruebas, implementación de pruebas y pruebas. ejecución
- Informar el progreso de la prueba, crear un informe de finalización de la prueba. • Apoyar al equipo en el uso de herramientas para implementar el proceso de prueba (por ejemplo, recaudar fondos para la compra de la herramienta, comprar licencias, controlar la implementación de la herramienta en la organización). • Decidir sobre la implementación de entornos de prueba. • Promocionar a los testers y al equipo de prueba y representar su punto de vista dentro del organización
- Desarrollar las habilidades y carreras de los evaluadores a través de un plan de capacitación, evaluaciones de desempeño y entrenamiento.

#### Función de prueba

Las tareas típicas de un tester son:

- Revisar los planes de prueba y participar en su desarrollo. • Ser coautor de los requisitos (historias de usuario) mientras realiza tareas de usuario colaborativo.
  - escritura de la historia
- Derivar criterios de aceptación comprobables para cada elemento del Product Backlog • Analizar, revisar y evaluar la base de prueba (es decir, requisitos, historias de usuarios y criterios de aceptación, especificaciones y modelos) para la capacidad de prueba • Identificar y documentar las condiciones de prueba y registrar la relación
  - entre casos de prueba, condiciones de prueba y base de prueba
- Diseñar, configurar y verificar entornos de prueba (generalmente en consultoría).
  - (con administradores de sistemas y redes) • Diseño e implementación de casos de prueba, procedimientos de prueba y scripts de prueba • Preparación y adquisición de datos de prueba • Co-creación del cronograma de ejecución de pruebas • Realización de pruebas, evaluación de resultados y documentación de desviaciones de lo esperado resultados
- Usar herramientas apropiadas para agilizar el proceso de prueba (por ejemplo, automatización de pruebas).
  - herramientas)
- Evaluar y medir las características no funcionales del software (ver Sección.
  - 2.2.2**
- Colaborar dentro del equipo (p. ej., revisar pruebas diseñadas por otros) • Usar, si es necesario, herramientas para la gestión de pruebas • Automatización de pruebas (p. ej., crear, ejecutar y modificar scripts de prueba)

Es importante asegurarse de que las personas que trabajan en el equipo de pruebas (y otras personas que realizan pruebas en general), es decir, aquellas involucradas en el análisis de pruebas, el diseño de pruebas, tipos de pruebas específicos o la automatización, sean especialistas en sus funciones. Como se mencionó anteriormente, según el nivel de prueba y el riesgo del producto y proyecto, la función de prueba puede ser realizada por diferentes personas:

- A nivel de componente y de integración de componentes, generalmente por parte de desarrolladores. • A nivel de prueba del sistema, generalmente por parte de evaluadores, miembros de una prueba independiente. equipo
- A nivel de prueba de aceptación, generalmente por parte de expertos y usuarios del negocio. • A nivel de prueba de aceptación operativa, generalmente por parte de operadores y usuarios del sistema. administradores de sistemas

## 1.5 Habilidades esenciales y buenas prácticas en pruebas

FL-1.5.1 (K2) Dé ejemplos de las habilidades genéricas requeridas para las pruebas.

FL-1.5.2 (K1) Recuerde las ventajas del enfoque de equipo completo.

FL-1.5.3 (K2) Distinguir los beneficios y desventajas de la independencia de las pruebas.

### 1.5.1 Habilidades genéricas requeridas para las pruebas

El proceso de desarrollo de software, incluidas las pruebas, lo llevan a cabo personas y, por lo tanto, las habilidades de los evaluadores individuales y los aspectos psicológicos del comportamiento humano son de gran importancia para el curso de las pruebas.

Habilidades esenciales requeridas en las pruebas

Un buen evaluador debe caracterizarse por una serie de cualidades (habilidades) que harán que su trabajo sea efectivo y eficiente. En particular, un buen probador tiene las siguientes características (ver también [20]):

- Conocimiento de las pruebas (para aumentar la eficacia de las pruebas, por ejemplo, mediante el uso de pruebas. técnicas)
- Minuciosidad, cuidado, curiosidad, atención al detalle, ser metódico (para identificar diferentes tipos de defectos, especialmente aquellos difíciles de encontrar) • Buenas habilidades de comunicación, escucha activa, ser un jugador de equipo (para interactuar efectivamente con todas las partes interesadas, comunicar información a otros, ser comprendido, poder informar y discutir defectos)
- Pensamiento analítico, pensamiento crítico, creatividad (para aumentar la eficacia de pruebas)
- Conocimiento técnico (para aumentar la eficiencia de las pruebas, por ejemplo, mediante el uso de herramientas de prueba)
- Conocimiento del dominio (para poder comprender y comunicarse con los usuarios finales y representantes comerciales)

Aspectos psicológicos en las pruebas Identificar

defectos durante las pruebas estáticas o identificar fallas durante las pruebas dinámicas puede percibirse como una crítica al producto o a su autor. Existe un fenómeno psicológico llamado sesgo de confirmación: la tendencia a preferir

información que confirma expectativas e hipótesis previas, independientemente de si esa información es cierta o no. El sesgo de confirmación puede dificultar la aceptación de información que contradiga las creencias existentes. Hace que las personas busquen información y la recuerden de forma selectiva, interpretándola de forma errónea. Este efecto es particularmente fuerte para temas que evocan emociones intensas e involucran opiniones fuertemente arraigadas. Por ejemplo, cuando leen sobre políticas de acceso a armas, las personas tienden a preferir fuentes que confirmen lo que ellos mismos piensan sobre el tema. También tienden a interpretar que la evidencia no concluyente respalda sus propias opiniones.

Una serie de experimentos realizados en la década de 1960 demostraron que las personas tienden a buscar la confirmación de sus creencias anteriores. Investigaciones posteriores explicaron esto como resultado de una tendencia a probar hipótesis de manera muy selectiva, centrándose en una posibilidad e ignorando las alternativas. Combinada con otros efectos, dicha estrategia afecta las conclusiones a las que llega la gente. Este error puede deberse a la capacidad limitada del cerebro humano para procesar información o a la optimización evolutiva, cuando los costos estimados de quedarse atrapado en el error no son mayores que los costos del análisis realizado de manera objetiva y científica. manera.

**Ejemplo** Los evaluadores están convencidos de que las fallas que informan son el resultado de defectos en el código; existe un efecto de confirmación, por el cual les resulta difícil aceptar situaciones en las que no hay ningún defecto y el fallo fue causado por una ejecución incorrecta de la prueba (un falso positivo).

También existen otros errores cognitivos que dificultan que las personas comprendan o acepten la información obtenida mediante pruebas. La gente a menudo tiende a culpar a la persona que trae las malas noticias, y estas situaciones a menudo surgen de las pruebas, ya que los evaluadores suelen ser los portadores de las malas noticias. Además, algunas personas ven las pruebas como una actividad destructiva, incluso si contribuyen significativamente al progreso del proyecto y a la calidad del producto. Para reducir reacciones similares, la información sobre defectos y fallas debe comunicarse de la manera más constructiva posible. Se deben hacer esfuerzos para reducir la tensión entre los evaluadores y los analistas de negocios, propietarios de productos, diseñadores y desarrolladores. Esto se aplica tanto a las pruebas estáticas como a las pruebas dinámicas.

**Habilidades interpersonales y comunicación eficiente** Los evaluadores y gerentes de pruebas deben tener sólidas habilidades interpersonales para comunicar de manera eficiente información sobre defectos, fallas, resultados de pruebas, progreso de las pruebas o riesgos y construir relaciones positivas con sus colegas. En este sentido, se sugieren las siguientes reglas de conducta:

- Coopere, no pelee. •
- Enfatizar los beneficios de las pruebas (los autores pueden utilizar la información sobre defectos para mejorar los productos de trabajo y desarrollar habilidades, y para las organizaciones, detectar y corregir defectos durante las pruebas significa ahorrar tiempo y dinero y reducir los riesgos generales de calidad del producto).

(continuado)

- Comunicar los resultados de las pruebas y otros hallazgos de manera neutral (centrarse en los hechos y no criticar a los autores del producto o solución de trabajo defectuoso).
- Crear informes de defectos y revisar conclusiones de manera objetiva y basada en hechos.
- Trate de entender por qué la otra persona reacciona negativamente a la información dada.
- Asegúrese de que la persona que llama comprenda la información que se transmite y viceversa. Esto es importante, especialmente si estás trabajando en un proyecto distribuido.

Definir sin ambigüedades el conjunto correcto de objetivos de prueba tiene importantes implicaciones psicológicas, porque la mayoría de las personas tienden a alinear sus planes y comportamiento con las metas establecidas por el equipo, la gerencia y otras partes interesadas. Hay que esforzarse por garantizar que los evaluadores cumplan los objetivos establecidos y que su mentalidad personal tenga el menor impacto posible en el trabajo que realizan. También es importante recordar que la mentalidad de una persona está determinada por las suposiciones que hace y la forma en que prefiere tomar decisiones y resolver problemas.

### 1.5.2 Enfoque de equipo completo

Una habilidad de prueba importante es ser un jugador de equipo: tener la capacidad de trabajar eficazmente en un equipo y hacer una contribución positiva al objetivo del equipo. Esta habilidad es la base del enfoque de "todo el equipo".

El enfoque de "equipo completo" se caracteriza por los siguientes atributos:

- Involucrar a todos aquellos con los conocimientos y habilidades necesarios para garantizar el éxito del proyecto •

Equipos relativamente pequeños de unas pocas personas • Compartir el mismo espacio de trabajo (ya sea físico o virtual) para establecer la comunicación y la interacción mucho más fácil

En el desarrollo ágil de software, el enfoque de "todo el equipo":

- Garantiza que el equipo incluya representantes del cliente y de otras empresas.  
Partes interesadas que deciden sobre las características del producto.
- Se apoya a través de reuniones diarias que involucran a todos los miembros del equipo, durante las cuales se comunica el progreso y se señala cualquier obstáculo para el progreso. • Promueve una dinámica de equipo más efectiva y eficiente. • Mejora la comunicación y la cooperación en el equipo. • Permite el uso de diferentes habilidades de los miembros del equipo en beneficio del proyecto • Hace que todos sean responsables de la calidad

La esencia del enfoque de "todo el equipo" es que los desarrolladores, los representantes comerciales y los evaluadores trabajen juntos en cada etapa del proceso de desarrollo de software.

La estrecha cooperación de estos tres grupos tiene como objetivo garantizar que se alcancen los niveles de calidad deseados. Esto incluye trabajar con representantes comerciales para ayudarlos a crear pruebas de aceptación apropiadas (ver Sección 4.5) y trabajar con desarrolladores para acordar una estrategia de prueba y decidir un enfoque de automatización de pruebas.

Los evaluadores también pueden transferir conocimientos sobre pruebas a otros miembros del equipo e influir en el desarrollo de productos.

Todo el equipo participa en cualquier consulta o reunión donde se determinan, analizan o estiman las características del producto. El concepto de involucrar a evaluadores, desarrolladores y representantes comerciales en todas las discusiones sobre las características del producto en desarrollo se conoce como el "poder de tres" [21] o los "Tres Amigos".

### 1.5.3 Independencia de las pruebas

Las tareas relacionadas con las pruebas pueden ser realizadas tanto por personas con roles específicos en el proceso de prueba como por personas con otros roles (como clientes). Por un lado, un cierto grado de independencia a menudo aumenta la eficacia de la detección de defectos, ya que el autor y el evaluador pueden estar sujetos a diferentes errores cognitivos (ver Sección.

[1.5.1\)](#). Por otro lado, la independencia no sustituye el conocimiento del producto y los desarrolladores pueden detectar eficazmente muchos defectos en el código que desarrollan.

Sin embargo, conviene recordar que el autor muchas veces no logra ver sus propios errores.

Cuando los desarrolladores crean código, hacen ciertas suposiciones al respecto. Cuando luego tengan que probar ese código, es posible que, incluso inconscientemente, escriban pruebas que verifiquen sus suposiciones.

Como resultado, las pruebas escritas por los desarrolladores generalmente detectarán pocos problemas en su propio código.

La independencia de las pruebas realizadas por un equipo de pruebas independiente tiene varios ventajas, en particular:

- Aumenta el énfasis en las pruebas •

Aumenta la eficiencia en la búsqueda de defectos y fallas • Proporciona beneficios adicionales, como la visión independiente de un equipo de pruebas capacitado y profesional

La independencia de las pruebas puede tener lugar en cualquier nivel de prueba. Hay diferentes niveles de independencia (ordenados a continuación de menor a mayor; consulte también la figura 1.13):

- No hay evaluadores independientes: los desarrolladores prueban su propio código. • Baja independencia: desarrolladores o evaluadores independientes que trabajan como parte de un equipo de desarrollo; los desarrolladores pueden probar productos desarrollados por colegas en lo que se conoce como prueba entre pares.

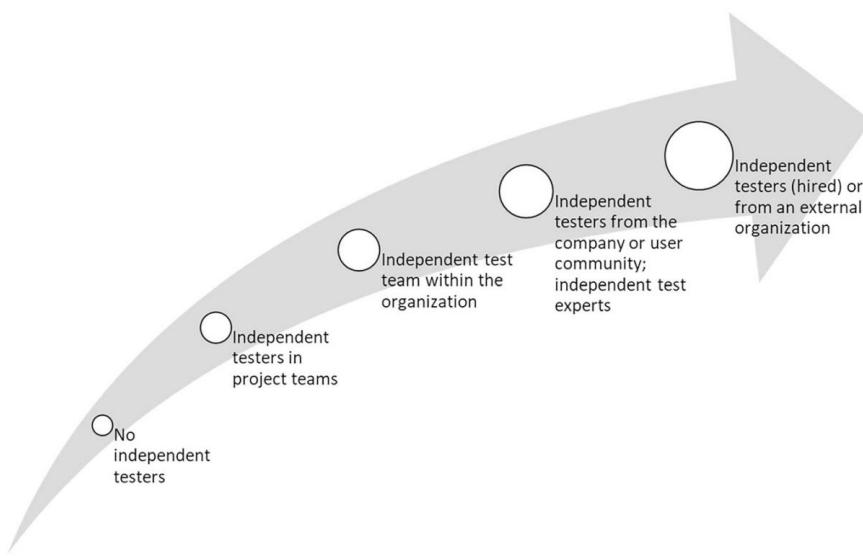


Fig. 1.13 Niveles de independencia de las pruebas

- Alta independencia: pruebas realizadas por un equipo de pruebas independiente que opera dentro de la organización (que informa a la dirección del proyecto o a la junta directiva) o por representantes de otros departamentos de la organización. • Muy alta independencia: evaluadores independientes externos a la organización, trabajando en sitio o fuera de sitio (outsourcing).

La solución óptima (para proyectos grandes y complejos o en equipos que producen productos críticos para la seguridad) son las pruebas en múltiples niveles:

- Desarrolladores de niveles inferiores: responsables de las pruebas de componentes y de integración de componentes (de este modo pueden controlar la calidad de su trabajo; sin embargo, debe recordarse que la falta de objetividad limita su eficacia). • Probadores independientes de niveles superiores: responsables del sistema. pruebas de integración, pruebas de sistemas y pruebas de aceptación

Al considerar los niveles de independencia, también se debe tener en cuenta el modelo del ciclo de vida del desarrollo de software. En el enfoque ágil, se acepta que los evaluadores puedan ser asignados para trabajar como parte del equipo de desarrollo, pero a veces pueden considerarse miembros de un equipo de prueba independiente más amplio. A menudo, los propietarios de productos realizan, al final de cada iteración, pruebas de aceptación para validar las historias de los usuarios.

Los evaluadores independientes tienen una perspectiva diferente sobre el producto bajo prueba; a menudo ven errores distintos de los notados por los desarrolladores debido a diferentes experiencias, puntos de vista de revisión técnica y errores cognitivos; pueden verificar la exactitud (revisiones) y cuestionar o refutar las suposiciones hechas por las partes interesadas tanto en las especificaciones como en la arquitectura del sistema. También se acercan al objeto de prueba sin expectativas ni sesgos preestablecidos.

La independencia de las pruebas conlleva no sólo beneficios sino también algunos riesgos. En particular:

- Aislamiento de los testers del equipo, lo que puede causar falta de comunicación, retrasos en la retroalimentación o malas relaciones con el equipo de producción (formamos un solo equipo): esto ocurre más a menudo en el caso de total independencia de las pruebas del equipo.
- Pérdida del sentido de responsabilidad de los desarrolladores por la calidad ("dado que el producto está siendo probado por expertos independientes, ya no necesito preocuparme por la calidad"). • La posibilidad de un cuello de botella al final del proyecto y responsabilizar a los probadores por la puesta en marcha inoportuna del producto ("juego de culpas"). • El riesgo de que los evaluadores independientes no tengan información importante (por ejemplo, sobre el objeto de prueba).

## Preguntas de muestra

### Pregunta 1.1

(FL-1.1.1, K1)

¿Cuál de los siguientes NO es un objetivo de prueba típico?

- A. Verificar que el objeto de prueba esté completo.
- B. Desencadenar tantas fallas como sea posible durante las pruebas de aceptación.
- C. Reducir el nivel de riesgo de fallas no detectadas previamente durante el software operación.
- D. Generar confianza en el nivel de calidad del sistema que se está probando.

Elija una respuesta.

### Pregunta 1.2

(FL-1.1.2, K2)

La siguiente es una lista de actividades relacionadas con defectos y fallas:

- i. Encontrar defectos en el código.
  - ii. Fallas desencadenantes.
- III. Analizando los defectos encontrados. IV.

Realización de retests.

¿Cuáles son las actividades de prueba y cuáles son las actividades de depuración?

- A. (ii) y (iv) son las actividades de prueba; (i) y (iii) son las actividades de depuración.
- B. (i) y (iv) son las actividades de prueba; (ii) y (iii) son las actividades de depuración.
- C. (ii) y (iii) son las actividades de prueba; (i) y (iv) son las actividades de depuración.
- D. (i), (iii) y (iv) son las actividades de prueba; (ii) es la actividad de depuración.

Elija una respuesta.

**Pregunta 1.3**

(FL-1.2.1, K2)

Eres un tester en un proyecto que desarrolla un juego basado en los mitos griegos. Eres

Actualmente estamos creando un criterio de aceptación para la siguiente historia de usuario:

Como jugador de nivel 4  
Quiero poder usar la varita de Midas.  
Para poder convertir el objeto que está frente a mí en oro y aumentar  
mis recursos financieros

Notaste que no hay información sobre el momento en que se convierte un artículo en oro.

¿Cuál de las siguientes afirmaciones ilustra MEJOR la contribución de las pruebas al éxito del proyecto?

- A. Informar al equipo que el autor de la historia del usuario realizó su tarea incorrectamente.
- B. Reducir el riesgo de producir una característica incorrecta o no comprobable.
- C. Obligar al propietario del producto a completar los datos faltantes de inmediato.
- D. Mejorar el comportamiento del tiempo, una subcaracterística del desempeño.

Elija una respuesta.

**Pregunta 1.4**

(FL-1.2.2, K1)

Considere las siguientes afirmaciones sobre pruebas y garantía de calidad.

- i. El aseguramiento de la calidad se centra en prevenir fallas verificando que la calidad  
Se cumplen los requisitos
- ii. El aseguramiento de la calidad se centra en controlar la calidad del producto creado iii. Las pruebas  
se centran en evaluar el software y los productos relacionados para determinar  
si cumplen con los requisitos específicos iv. Las  
pruebas se centran en eliminar defectos del software.

¿Cuáles de estas son ciertas?

- R. (i) y (iii) son verdaderos; (ii) y (iv) son falsos.
- B. (ii) y (iv) son verdaderos; (i) y (iii) son falsos.
- C. (i) y (iv) son verdaderos; (ii) y (iii) son falsos.
- D. (ii) y (iii) son verdaderos; (i) y (iv) son falsos.

Elija una respuesta.

**Pregunta 1.5**

(FL-1.2.3, K1)

¿Cuál de los siguientes es el ejemplo correcto de un defecto?

- A. Una acción humana que provoca un resultado incorrecto.
- B. Una materialización en el código del error del desarrollador del software.
- C. Una desviación del comportamiento esperado del software.
- D. Un caso de prueba para comprobar la respuesta del sistema a datos erróneos.

Elija una respuesta.

**Pregunta 1.6**

(FL-1.3.1, K2)

Según el principio de Pareto, la mayoría de los problemas son causados por un pequeño número de causas. Ésta es la base de uno de los principios de las pruebas. ¿Cuál?

- R. Las pruebas tempranas ahorran tiempo y dinero.
- B. Las pruebas dependen del contexto.
- C. Las pruebas se desgastan.
- D. Los defectos se agrupan.

Elija una respuesta.

**Pregunta 1.7**

(FL-1.4.1, K2)

¿Durante qué fase del proceso de prueba se verifica la capacidad de prueba de la base de prueba?

- A. Planificación de pruebas.
- B. Diseño de prueba.
- C. Análisis de pruebas.
- D. Implementación de pruebas.

Elija una respuesta.

**Pregunta 1.8**

(FL-1.4.2, K2)

¿Cuál de los siguientes tiene el MENOR impacto en el proceso de prueba de una organización?

- A. Presupuesto del proyecto.
- B. Normas y estándares externos.
- C. Número de probadores certificados empleados.
- D. Conocimiento de los evaluadores del ámbito empresarial.

Elija una respuesta.

Pregunta 1.9

(FL-1.4.3, K2)

¿Cuál de los siguientes NO es un producto de trabajo resultante del monitoreo y control de pruebas?

- A. Informe de progreso de la prueba.
- B. Información sobre el nivel de riesgo actual en el producto.
- C. Documentación que describa las acciones de control realizadas.
- D. Informe de finalización de la prueba.

Elija una respuesta.

Pregunta 1.10

(FL-1.4.4, K2)

¿Cuál de las siguientes opciones es posible gracias a un mecanismo para rastrear la relación entre los elementos de la base de prueba y sus correspondientes productos de trabajo de prueba?

- A. Cálculo del nivel de riesgo en el producto en base a los resultados de las pruebas.
- B. Definir un nivel aceptable de cobertura de código para cada componente.
- C. Usar un oráculo de prueba para determinar automáticamente el resultado esperado para un caso de prueba.
- D. Derivar datos de prueba que logren una cobertura de partición de equivalencia total.

Elija una respuesta.

Pregunta 1.11 (FL

1.4.5, K2)

¿Cuáles de las siguientes son actividades típicas de gestión de pruebas y cuáles son actividades de prueba típicas?

i. Coordinar la implementación de la estrategia de prueba y el plan de prueba. ii. Definición de condiciones de prueba. III.

Creación de un informe de finalización de la prueba.

IV. Implementación de scripts de prueba automatizados.

v. Decidir sobre la implementación de entornos de prueba. vi. Verificación de entornos de prueba.

A. (iv), (v) y (vi) son las actividades de gestión de pruebas; (i), (ii) y (iii) son las actividades de prueba.

B. (ii), (iii) y (vi) son las actividades de gestión de pruebas; (i), (iv) y (v) son las actividades de prueba.

C. (i), (ii) y (v) son las actividades de gestión de pruebas; (iii), (iv) y (vi) son las actividades de prueba.

D. (i), (iii) y (v) son las actividades de gestión de pruebas; (ii), (iv) y (vi) son las actividades de prueba.

Elija una respuesta.

Pregunta 1.12

(FL-1.5.1, K2)

¿Cuál de las siguientes habilidades es MENOS crítica para el evaluador?

- A. Pensamiento analítico.
- B. Conocimiento del dominio.
- C. Habilidades de programación.
- D. Habilidades de comunicación.

Elija una respuesta.

Pregunta 1.13

(FL-1.5.2, K1)

¿Cuál de las siguientes DOS actividades coincide MÁS con las responsabilidades?

¿Está relacionado con el enfoque de "todo el equipo"?

- R. Los evaluadores son responsables de la implementación de las pruebas de componentes, que pasan a los desarrolladores para su ejecución.
- B. Los evaluadores trabajan con representantes comerciales y desarrolladores para crear aceptaciones.  
pruebas de distanciamiento.
- C. Los representantes comerciales seleccionan las herramientas que los desarrolladores y evaluadores utilizarán durante el proyecto.
- D. Las pruebas no funcionales son diseñadas por el cliente y ejecutadas por evaluadores y desarrolladores.
- E. No sólo los evaluadores sino también los desarrolladores y representantes comerciales son responsables de la calidad del producto.

Seleccione DOS respuestas.

Pregunta 1.14 (FL

1.5.3, K2)

¿Por qué las pruebas suelen ser realizadas por evaluadores independientes?

- R. Ayuda a aumentar el énfasis en las pruebas y permite la opinión independiente de evaluadores profesionales.
- B. Debido a que los desarrolladores no tienen la capacidad de probar su código debido a sesgo cognitivo.
- C. Porque las fallas detectadas se informan de manera constructiva.
- D. Porque encontrar defectos no se considera una crítica a los desarrolladores.

Elija una respuesta.

## Capítulo 2 Pruebas en todo el software

### Ciclo de vida del desarrollo



#### Palabras clave

Test de aceptación	un nivel de prueba que se centra en determinar si se acepta el sistema. pruebas basadas en un análisis de la especificación del componente o sistema.
Pruebas de caja negra	Sinónimos: pruebas basadas en especificaciones.
Pruebas de integración de componentes	Pruebas de integración de componentes. Sinónimos: prueba de integración de módulos, prueba de integración de unidades. un nivel de prueba que se centra en componentes individuales de hardware o software. Sinónimos: prueba de módulo, prueba unitaria. un tipo de prueba relacionada con cambios que se realiza después de corregir un defecto para confirmar que una falla causada por ese defecto no vuelve a ocurrir.
Pruebas de componentes	Sinónimos: prueba de integración de unidades. un nivel de prueba que se centra en componentes individuales de hardware o software. Sinónimos: prueba de módulo, prueba unitaria. un tipo de prueba relacionada con cambios que se realiza después de corregir un defecto para confirmar que una falla causada por ese defecto no vuelve a ocurrir.
Prueba de confirmación	Sinónimos: volver a probar. Pruebas realizadas para evaluar si un componente o sistema satisface los requisitos funcionales.
Pruebas funcionales	Referencias: ISO 24765. un nivel de prueba que se centra en las interacciones entre componentes o sistemas. probar los cambios en un sistema operativo o el impacto de un entorno modificado en un sistema operativo. Pruebas realizadas para evaluar que un componente o sistema cumple con requisitos no funcionales.
Pruebas de integración	
Pruebas de mantenimiento	
Pruebas no funcionales	

Pruebas de regresión	un tipo de prueba relacionada con el cambio para detectar si se han introducido defectos o descubiertos en áreas sin cambios del software.
Mayús-izquierda	un enfoque para realizar pruebas y calidad actividades de aseguramiento lo antes posible en el Ciclo de vida del desarrollo de programas.
Pruebas de integración del sistema	las pruebas de integración de sistemas.
Pruebas del sistema	un nivel de prueba que se centra en verificar que un El sistema en su conjunto cumple con los requisitos especificados.
Nivel de prueba	una instancia específica de un proceso de prueba. Sinónimos: etapa de prueba.
Objeto de prueba	el producto de trabajo a probar.
Tipo de prueba	un grupo de actividades de prueba basadas en una prueba específica objetivos dirigidos a características específicas de un componente o sistema. Después de TMap.
Pruebas de caja blanca	pruebas basadas en un análisis de la situación interna estructura del componente o sistema. Sinónimos: prueba de caja clara, basada en código pruebas, pruebas de caja de vidrio, cobertura lógica pruebas, pruebas basadas en lógica, pruebas estructurales, pruebas basadas en estructuras.

## 2.1 Pruebas en el contexto de un ciclo de desarrollo de software

FL-2.1.1 (K2) Explicar el impacto del ciclo de vida de desarrollo de software elegido en pruebas

FL-2.1.2 (K1) Recordar buenas prácticas de prueba que se aplican a todo el software. ciclos de vida de desarrollo

FL-2.1.3 (K1) Recordar los ejemplos de enfoques de desarrollo basados en las pruebas.

FL-2.1.4 (K2) Resumir cómo DevOps podría tener un impacto en las pruebas

FL-2.1.5 (K2) Explicar el enfoque de desplazamiento a la izquierda

FL-2.1.6 (K2) Explicar cómo se pueden utilizar las retrospectivas como mecanismo para el proceso. mejora

Un modelo de ciclo de vida de desarrollo de software (SDLC) es una representación abstracta y de alto nivel del proceso de desarrollo de software. El modelo SDLC describe la actividades incluidas en el proceso de desarrollo de software y las relaciones entre ellos, tanto lógicos como temporales. Los modelos SDLC se utilizan para determinar cómo se desarrollará el software. También facilitan la comprensión de la sucesión de las diferentes fases del proceso de fabricación.

Ejemplos de tipos de modelos SDLC clásicos incluyen:

- Modelos secuenciales (p. ej., modelo en cascada, modelo V) •
- Modelos iterativos (p. ej., modelo en espiral de Boehm, creación de prototipos)
- Modelos incrementales (p. ej., proceso unificado)

Algunas actividades dentro del proceso de desarrollo de software también pueden describirse mediante modelos, metodologías de desarrollo o prácticas ágiles más detalladas. Ejemplos de tales enfoques incluyen:

- Scrum •
- Kanban •
- Lean IT (desarrollo basado en la llamada gestión lean) • Programación extrema (XP) • Desarrollo basado en pruebas (TDD) • Desarrollo basado en pruebas de aceptación (ATDD) • Desarrollo basado en el comportamiento (BDD) • Dominio -Diseño impulsado (DDD) • Desarrollo impulsado por funciones (FDD)

Para obtener más información sobre los modelos SDLC en el contexto de la ingeniería de software, consulte, por ejemplo, [22].

### 2.1.1 Impacto del ciclo de vida del desarrollo de software en las pruebas

Las pruebas, para que sean efectivas, deben integrarse con el modelo SDLC del proyecto. La elección del modelo SDLC afecta las siguientes cuestiones de prueba:

- Alcance y cronograma de las actividades de prueba • Selección y cronograma de niveles y tipos de prueba • Nivel de detalle de la documentación de prueba •
- Selección de técnicas y prácticas de prueba • Alcance de la automatización de pruebas

**Modelos SDLC secuenciales** Los modelos SDLC secuenciales suponen la ejecución de actividades de desarrollo individuales, una tras otra, de forma lineal. La consecuencia de adoptar tal modelo de desarrollo de software es que una fase determinada no puede comenzar hasta que se complete la fase anterior. En la práctica, a veces puede ocurrir que las fases se superpongan. Sin embargo, los modelos secuenciales suponen, al menos en teoría, que antes de que comience la siguiente fase, hay un momento de verificación de que todas las actividades de la fase anterior se han realizado correctamente.

En las primeras fases de un proyecto ejecutado en modelos secuenciales, el evaluador suele participar en pruebas estáticas: revisiones de requisitos y diseño de pruebas. El producto en forma ejecutable generalmente se entrega en fases posteriores, por lo que, en la mayoría de los casos, las pruebas dinámicas no se pueden realizar en las primeras etapas del ciclo de desarrollo.

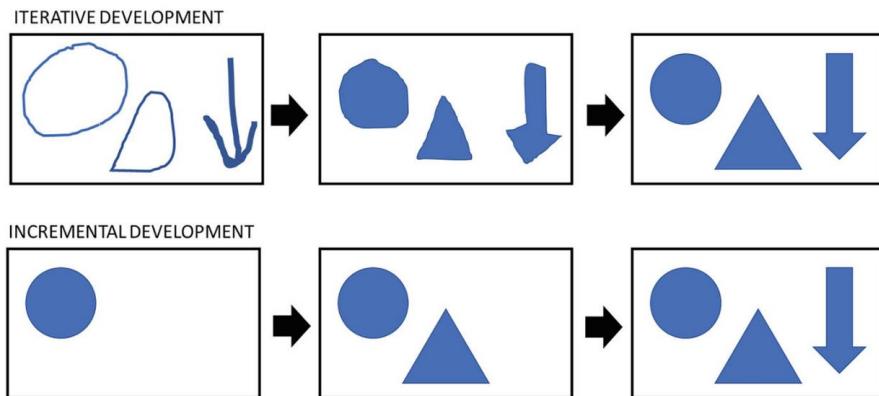


Fig. 2.1 Diferencia simbólica entre los modelos SDLC iterativos e incrementales

A continuación se analizan ejemplos de los modelos secuenciales más importantes.

**Modelos SDLC iterativos e incrementales** Todos los modelos SDLC iterativos e incrementales se basan en el mismo principio fundamental que establece que el desarrollo de software se realiza en ciclos. Existen algunas diferencias entre estos dos tipos de modelos. Estos se muestran simbólicamente en la Fig. 2.1.

El desarrollo incremental de software es el proceso de especificar requisitos y diseñar, construir y probar el sistema en partes, lo que significa que la funcionalidad del software crece incrementalmente. El tamaño de los incrementos individuales de funcionalidad depende del modelo específico del ciclo de desarrollo. Algunos modelos prevén la división en trozos más grandes, mientras que otros prevén trozos más pequeños. Un único incremento de funcionalidad puede incluso limitarse a un único cambio en una pantalla de interfaz de usuario o una nueva opción de consulta.

La parte inferior de la Fig. 2.1 muestra este enfoque. Supongamos que nuestro producto es una imagen compuesta por tres elementos: un círculo, un triángulo y una flecha. En el modelo incremental, creamos partes de esta imagen una a la vez, como cada figura en incrementos sucesivos. Específicamente, esto significa que la funcionalidad agregada incrementalmente ya debería tener la forma que tendrá en el producto objetivo final. Por supuesto, los cambios son inevitables, pero éste es el principio general que distingue los modelos incrementales de los iterativos. Se puede realizar un solo incremento en un modelo secuencial o iterativo.

El desarrollo iterativo, por otro lado, implica especificar, diseñar, construir y probar juntos grupos de funcionalidades en una serie de ciclos, a menudo de una duración estricta. Las iteraciones pueden incluir cambios en la funcionalidad producida en iteraciones anteriores, junto con cambios en el alcance del proyecto. Cada iteración entrega software funcional que es un subconjunto creciente del conjunto total de funcionalidades hasta que se entrega la versión final del software o se detiene el desarrollo.

La parte superior de la Fig. 2.1 presenta un enfoque iterativo. Realizamos la construcción de nuestra imagen, por así decirlo, en bocetos, cada uno de los cuales abarca la totalidad del

idea final, pero todo este conjunto se presenta en diferentes niveles de detalle: primero tenemos un dibujo inicial, un esquema del concepto de la imagen. En iteraciones posteriores, refinamos este concepto, "dando cuerpo" a más y más detalles. Por supuesto, este enfoque no se aplica necesariamente a todo el producto sino, por ejemplo, a algún grupo destacado de funcionalidades, como se describió anteriormente.

En algunos modelos iterativos e incrementales, se supone que cada iteración o incremento termina con un producto funcional. Esto significa que en cada iteración (incremento), se pueden realizar pruebas estáticas y dinámicas en todos los niveles de prueba.

La entrega frecuente de piezas de software que funcionen requiere una retroalimentación rápida y pruebas de regresión exhaustivas.

#### Prácticas ágiles Los

métodos ágiles de desarrollo de software suponen que el cambio puede ocurrir en cualquier punto del proyecto. Esta suposición lleva a estos métodos a favorecer la documentación liviana y la automatización extensa de las pruebas para hacer que las pruebas de regresión, en particular, sean más fáciles de realizar. Además, una gran parte de las pruebas manuales se lleva a cabo utilizando técnicas de prueba basadas en la experiencia que no requieren una planificación previa larga y exhaustiva (consulte la Sección 4.4).

Ahora discutiremos algunos de los modelos SDLC secuenciales e iterativos más importantes.

### Modelos secuenciales

#### Modelo en cascada

En el modelo en cascada (ver Fig. 2.2), las actividades de desarrollo de software (como análisis de requisitos, diseño, desarrollo de código y pruebas) se realizan una tras otra. Según los supuestos de este modelo, las actividades de prueba ocurren sólo después de que se hayan completado todas las demás actividades de desarrollo. Esta es una situación problemática desde el punto de vista del evaluador. En el cap. 1, notamos que las pruebas deben comenzar lo antes posible, porque cuanto más tarde suceda, más costoso será corregir los defectos encontrados.

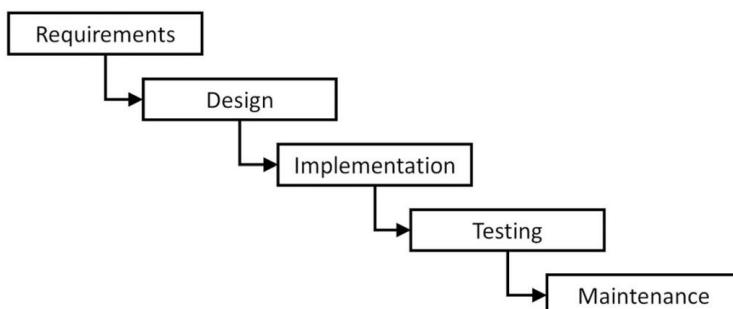


Fig. 2.2 Modelo de cascada

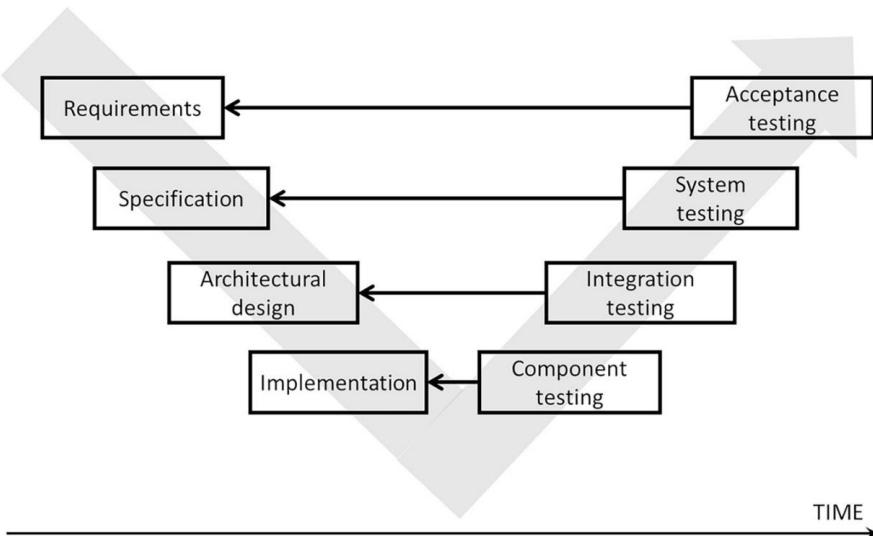


Fig. 2.3 Modelo V

El modelo en cascada nos impide comenzar a realizar pruebas antes de tiempo. Entonces, ¿cuál es el beneficio de utilizar este enfoque? Bueno, funciona bien en proyectos donde los requisitos son bien conocidos y están bien desarrollados, y el equipo puede estar seguro (o casi seguro) de que no cambiarán durante las actividades de desarrollo. El modelo en cascada es una buena solución en proyectos típicos “por lotes” (por ejemplo, proyectos de implementación que requieren una configuración más compleja) o en aquellos donde hay requisitos muy bien identificados y hay poco o ningún riesgo de que cambien (por ejemplo, requisitos que resultan directamente de la legislación).

#### Modelo V

El modelo V es un modelo en cascada modificado que intenta abordar los inconvenientes del modelo en cascada, asociados con las pruebas tardías. El modelo (ver Fig. 2.3) asume la integración del proceso de prueba en el proceso de desarrollo de software, implementando así el principio de prueba temprana. Además, el modelo V incluye niveles de prueba vinculados a cada fase correspondiente del desarrollo de software, lo que promueve aún más las pruebas tempranas (ver Sección 2.2). En este modelo, la ejecución de las pruebas asociadas con todos los niveles de prueba se produce de forma secuencial, pero en algunos casos puede haber fases superpuestas.

El modelo V es, por tanto, un intento de abordar los problemas planteados por el modelo en cascada en el contexto de las pruebas y el control de calidad. Llama la atención sobre las actividades de prueba que ocurren en cada fase, no solo en la fase de prueba dinámica (la rama derecha del modelo) sino también en las fases de desarrollo (la rama izquierda del modelo). En este último caso, los evaluadores no siempre pueden realizar pruebas (por ejemplo, en la fase de diseño aún no hay ningún producto comprobable para ejecutar), pero siempre pueden revisar la base de la prueba y diseñar las pruebas relacionadas. Una versión del modelo V, el llamado

El modelo W propone que las actividades de prueba en las fases de fabricación también deberían incluir pruebas estáticas, es decir, que se deberían realizar revisiones de los productos de trabajo (por ejemplo, revisión de requisitos, revisión de arquitectura, revisión de código, etc.).

#### Modelos iterativos e incrementales

##### Modelo de Proceso Unificado (UP)

El modelo UP (ver Fig. 2.4) es un modelo iterativo e incremental. En esencia, UP no es un proceso único sino un marco de proceso flexible dentro del cual se definen procesos individuales, como el modelado de negocios, los requisitos, el análisis y diseño, la implementación, las pruebas o el despliegue. Estos procesos pueden ser seleccionados y adaptados a las necesidades del proyecto. Las iteraciones individuales suelen llevar un tiempo relativamente largo (por ejemplo, 2 o 3 meses) y las partes incrementales del sistema son correspondientemente grandes y cubren, por ejemplo, dos o tres grupos de funcionalidades relacionadas.

##### El modelo en espiral de Boehm

Boehm El modelo en espiral es un enfoque en el que se crean componentes experimentales incrementales que luego pueden reconstruirse ampliamente o incluso abandonarse en etapas posteriores del desarrollo del software (ver Fig. 2.5). Los componentes o sistemas que utilizan los modelos anteriores a menudo incluyen niveles de prueba repetidos y superpuestos en el software.

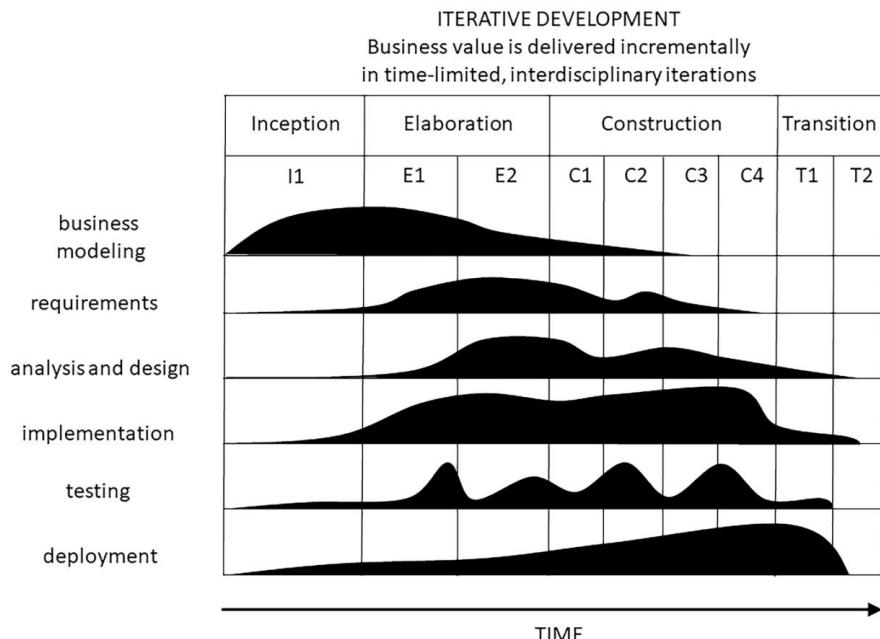


Fig. 2.4 Proceso unificado

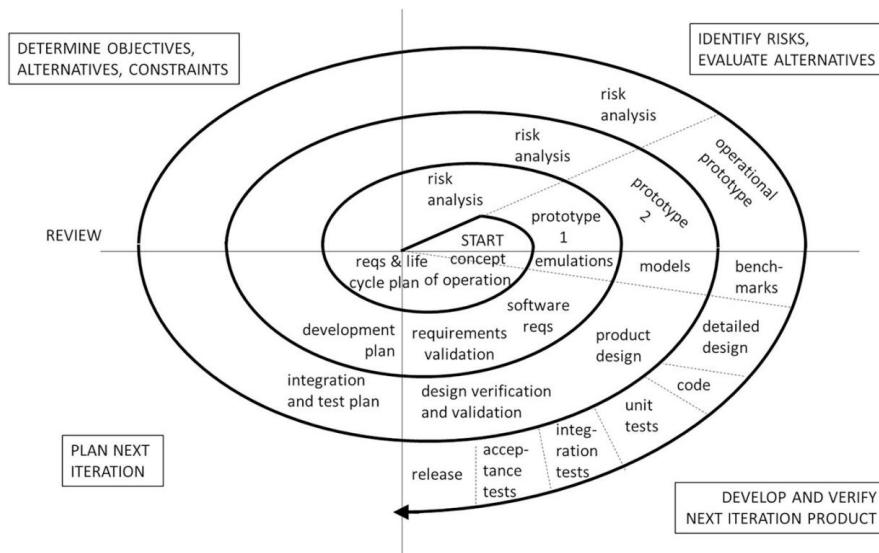


Fig. 2.5 Modelo espiral de Boehm

ciclo de desarrollo. Lo ideal es que cada funcionalidad se pruebe en múltiples niveles, acercándose a la versión final. En algunos casos, los equipos utilizan la entrega continua o la implementación continua, enfoques que hacen un uso extensivo de la automatización en múltiples niveles de pruebas como parte del proceso de entrega de software. Además, muchos de estos modelos incorporan el concepto de equipos autoorganizados, que pueden cambiar la forma en que se organiza el trabajo en relación con las pruebas y la relación entre evaluadores y desarrolladores.

El modelo espiral fue propuesto por Barry Boehm y, en palabras del propio Boehm, es esencialmente un “modelo de modelos”, ya que puede configurarse para obtener prácticamente cualquier otro modelo del ciclo de desarrollo (por ejemplo, restringiéndolo al último trimestre). De la espiral exterior, se obtiene un modelo de cascada). Su característica y uno de sus rasgos más importantes es el análisis de riesgos que se realiza antes del inicio de cada ciclo de desarrollo sucesivo.

#### Modelo de creación de prototipos

El modelo de creación de prototipos se utiliza a menudo cuando los objetivos de diseño no están estrictamente definidos o especificados de antemano. Por ejemplo, el cliente define un conjunto de objetivos generales para el software, pero no especifica requisitos funcionales detallados, el desarrollador puede tener dudas sobre la efectividad del algoritmo implementado o la forma de interacción del programa con el usuario, etc. En distintos tipos de situaciones, el modelo de creación de prototipos puede ofrecer el mejor enfoque.

La creación de prototipos ayuda a las partes interesadas a comprender mejor qué se debe construir cuando los requisitos son confusos. El proceso (ver Fig. 2.6) comienza con la comunicación que define los objetivos (requisitos) generales del software. A la fase de planificación le sigue una serie de iteraciones rápidas de creación de prototipos. En cada iteración, el equipo

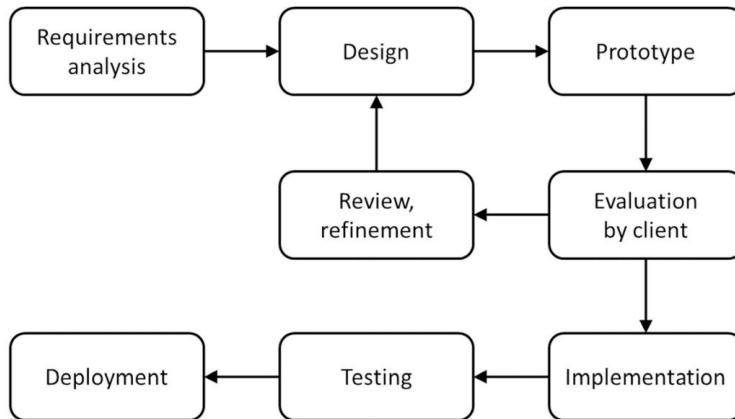


Fig. 2.6 Modelo de creación de prototipos

se centra en la representación de aquellos aspectos del software que serán visibles para los usuarios finales y finaliza con la construcción de un prototipo funcional. El prototipo es implementado y evaluado por las partes interesadas, quienes brindan retroalimentación. Esta información se utiliza para refinar aún más los requisitos.

Idealmente, el prototipo sirve como mecanismo para identificar los requisitos de software. Si se va a construir un prototipo funcional, se pueden utilizar fragmentos de programas existentes o se pueden emplear herramientas que permitan una generación rápida de programas funcionales (por ejemplo, maquetas de GUI “dinámicas”).

### Metodologías de Desarrollo y Prácticas Ágiles

#### Scrum

Scrum (ver Fig. 2.7) es actualmente uno de los métodos de desarrollo de software más comunes.<sup>1</sup> Scrum divide el desarrollo de software en iteraciones cortas de la misma duración (generalmente de 1 a 4 semanas), y las partes iterativas del sistema son correspondientemente pequeñas, es decir, incluyen, por ejemplo, algunas mejoras o nuevas funcionalidades.

En un modelo como Scrum, las pruebas se vuelven desafiantes debido a iteraciones cortas y cambios frecuentes. Por lo tanto, en lugar de un proceso de diseño de casos de prueba largo y exhaustivo, en Scrum, es más común ver prácticas como las pruebas exploratorias.

<sup>1</sup> A menudo uno puede encontrarse con la opinión de que Scrum no es un método sino un marco. A veces también se utiliza la palabra “metodología” en lugar de “método”. Estos malentendidos se deben a la falta común, lamentablemente, de distinguir el significado de las palabras: “método”, “metodología” y “marco”. Un método es un conjunto de reglas para realizar un trabajo. Marco, por otra parte, es la disposición e interrelación de los elementos que conforman un todo. La metodología, por otra parte, es la ciencia que estudia los métodos. Por lo tanto, Scrum es un método y, contra toda apariencia, es bastante prescriptivo (es decir, impone una determinada forma de hacer las cosas) sobre muchos elementos del modelo, como la duración de los sprints o la duración del llamado stand-up diario. reuniones.

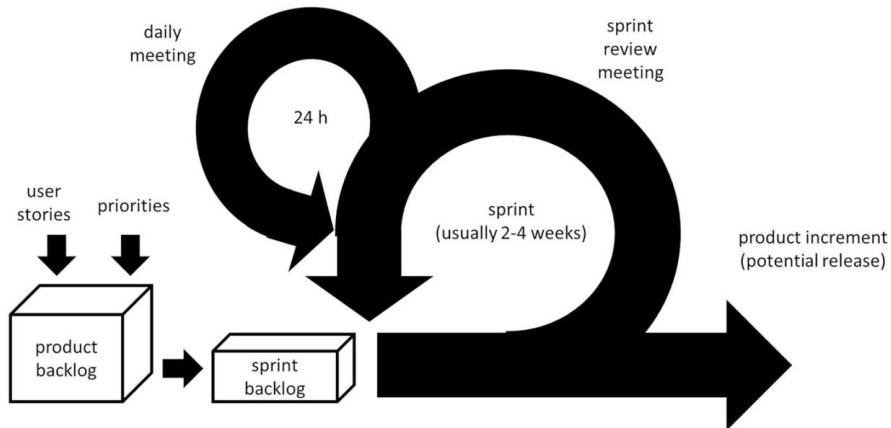


Figura 2.7 Melé

(ver Sección 4.4.2) o un énfasis en la automatización extensiva de las pruebas; esto es especialmente cierto para las pruebas de regresión, cuyo número tiende a crecer rápidamente con cada iteración.

Scrum es una combinación de un sistema push y pull, porque antes de cada iteración, el equipo selecciona ("empuja") del backlog del producto al backlog del sprint un conjunto predeterminado de tareas que se completarán en esa iteración, y dentro de un sprint, cada miembro del equipo comienza a trabajar en la siguiente tarea cuando la anterior ha terminado ("tirar").

#### Kanban

Kanban permite ofrecer una mejora o funcionalidad a la vez (inmediatamente después de la preparación) o agrupar más funcionalidades para transferirlas simultáneamente al entorno de producción.

Originalmente, el método se utilizaba en empresas manufactureras como Toyota, donde los productos se desarrollaban en serie.

Sin embargo, también se puede adaptar a proyectos de TI. Kanban utiliza el llamado tablero Kanban para visualizar, planificar y supervisar el proceso de desarrollo (ver Fig. 2.8).

El método Kanban se basa en las llamadas tarjetas de producto y organiza el proceso de desarrollo de tal manera que cada estación de producción produzca exactamente lo que se necesita en cada momento. Además, cada estación de trabajo tiene un límite descendente de trabajo que puede realizar en una determinada unidad de tiempo (el llamado límite de trabajo en proceso, WIP). Esta organización del trabajo permite gestionar la producción de manera eficiente, de modo que no se crean elementos innecesarios y la "masa" de tareas a realizar "fluye" sin problemas a través del sistema de producción, optimizando así el consumo de recursos y el tiempo de producción.

Kanban es un sistema de extracción, porque un trabajador en un trabajo determinado "extrae" una tarea de la estación de trabajo anterior donde un producto ya se ha completado y está listo para ser transferido.

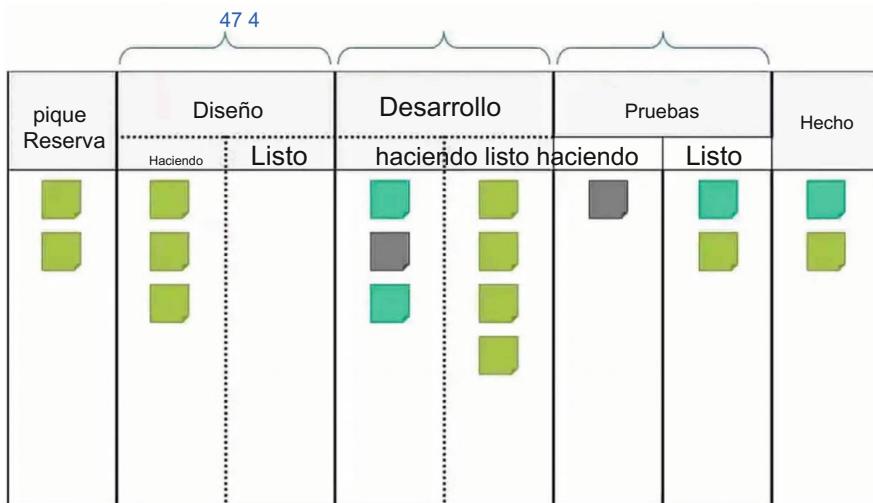


Fig. 2.8 Ejemplo de tablero Kanban (fuente: [scrum.org](http://scrum.org))

Principios para seleccionar un modelo de ciclo de vida de desarrollo de software Los modelos SDLC deben seleccionarse y adaptarse al contexto resultante de las características del proyecto y producto. En consecuencia, a la hora de seleccionar y ajustar el modelo adecuado se debe tener en cuenta lo siguiente:

- El propósito del proyecto • El tipo de producto que se está desarrollando • Prioridades comerciales (como el tiempo de comercialización) • Riesgos identificados del producto y del proyecto

Por ejemplo, el desarrollo y prueba de un sistema administrativo interno menor debe realizarse de manera diferente que el desarrollo y prueba de un sistema crítico para la seguridad, como el sistema de control de frenos de un automóvil. Como otro ejemplo, en algunos casos, los problemas organizacionales y culturales pueden obstaculizar la comunicación entre los miembros del equipo, lo que puede resultar en una desaceleración del desarrollo de software en un modelo iterativo.

Existen muchos modelos SDLC en la industria de TI y, a menudo, elegir el correcto puede resultar difícil. Sin embargo, conviene tener en cuenta los criterios de selección antes mencionados. También debéis ser conscientes de que el modelo no es una santidad inviolable. Es sólo una propuesta que nos dice cómo organizar el proceso de desarrollo de software. Un modelo de este tipo puede e incluso debe adaptarse a las necesidades y requisitos de la organización en la que se desarrolla el software.

Desde el punto de vista del evaluador, el aspecto más relevante del SDLC será, por supuesto, el proceso de prueba. Dependiendo del contexto del proyecto, puede ser necesario combinar o reorganizar algunos niveles de prueba y/o actividades de prueba. Por ejemplo, en el caso de la integración de software comercial listo para usar (COTS) con un sistema más grande, el comprador puede realizar pruebas de interoperabilidad en el nivel de prueba de integración del sistema (por ejemplo,

para la integración con infraestructura y otros sistemas) y a nivel de pruebas de aceptación (pruebas funcionales y pruebas no funcionales junto con pruebas de aceptación del usuario y pruebas de aceptación operativa). Los niveles y tipos de pruebas se describen en detalle en las Secciones. [2.2.1](#) y [2.2.2](#).

Además, se pueden combinar diferentes modelos de SDLC. Un ejemplo sería utilizar el modelo V para el desarrollo y prueba de la parte back-end del sistema y el modelo ágil para el desarrollo y prueba del front-end (interfaz de usuario).

Otra variante es utilizar un modelo de creación de prototipos al principio del proyecto y luego reemplazarlo con un modelo incremental después de la fase experimental.

Para los sistemas relacionados con Internet de las cosas (IoT), que constan de muchos objetos diferentes (como dispositivos, productos y servicios), generalmente se aplican modelos SDLC separados a partes individuales del sistema. Esto plantea un gran desafío, especialmente para el desarrollo de versiones individuales de sistemas IoT. Además, en el caso de los objetos anteriores, se pone más énfasis en las últimas etapas del SDLC, después de que los objetos ya estén en funcionamiento (por ejemplo, las fases de producción, actualización y desmantelamiento).

El proceso de selección del modelo SDLC correcto puede realizarse de la siguiente manera. En el primer paso definimos los criterios con los que evaluaremos la idoneidad de cada modelo en nuestro proyecto. Ejemplos de criterios que se pueden considerar son:

- Tamaño y experiencia del equipo •

Tamaño, tipo y nivel de complejidad del proyecto • Relaciones con los clientes, estabilidad de los requisitos, frecuencia de sus cambios • Dispersión geográfica del equipo • Compatibilidad del modelo con la estrategia empresarial/organizacional de la empresa

En el paso dos, comparamos posibles modelos SDLC, considerando tanto sus ventajas como sus desventajas. Tenga en cuenta que no existen soluciones perfectas ni universales. Cada modelo SDLC tendrá algunos aspectos positivos, pero también nos presentará algunos desafíos. Por ejemplo, utilizar modelos ágiles, especialmente en un equipo que anteriormente utilizaba modelos secuenciales, requiere cambiar el pensamiento de los miembros del equipo, lo que suele ser una tarea muy difícil.

En el paso tres, evaluamos las necesidades de nuestra organización. El modelo SDLC debe reflejar la naturaleza y las operaciones de la empresa o equipo para el que trabajamos. Producir software para las industrias aeroespacial, médica o militar requerirá procedimientos y enfoques completamente diferentes a, por ejemplo, crear un juego para un dispositivo móvil en una pequeña empresa recién creada.

En el paso final, aplicamos los criterios previamente seleccionados en el contexto de nuestra las necesidades de la organización.

### 2.1.2 Ciclo de vida del desarrollo de software y buenas prácticas de prueba

El conocimiento de los modelos SDLC es importante desde el punto de vista del evaluador, porque la selección del modelo que se adopta en el proceso de desarrollo afecta cuándo y cómo se realizarán las actividades de prueba. Insecto. 2.1.1, describimos varios modelos SDLC de ejemplo. Sin embargo, independientemente del modelo en el que trabajará un evaluador, existen varios principios de buenas prácticas de prueba que se deben seguir:

- Para cada actividad de desarrollo de software, existe una actividad de prueba correspondiente. Este principio llama la atención sobre la “totalidad” de las actividades de prueba: cada producto de trabajo que se produjo durante el desarrollo del software también debe estar sujeto a control de calidad.
- Cada nivel de prueba corresponde a objetivos de prueba apropiados para la fase o grupo de actividades dentro del ciclo de desarrollo de software. Este principio llama la atención sobre el hecho de que los objetivos de las pruebas pueden diferir significativamente (ver Sección 1.1); en un nivel, estaremos principalmente interesados en detectar tantos defectos como sea posible, mientras que en otro nivel, estaremos más interesados en validar que el sistema cumple con los requisitos y necesidades del cliente.
- El análisis y el diseño de pruebas para un nivel de prueba determinado deben iniciarse ya durante la ejecución de la actividad de desarrollo de software correspondiente. Esta regla está relacionada con el principio de pruebas tempranas (Sección 1.3), que supone que las actividades de prueba comienzan lo antes posible para detectar problemas lo antes posible. Los defectos suelen ser baratos de solucionar en las primeras fases y, a menudo, muy caros en las fases posteriores. Tenga en cuenta que a veces los defectos no se encuentran mediante pruebas físicas de la aplicación sino mediante actividades de diseño de pruebas (consulte el Capítulo 4). • Los evaluadores deben participar en revisiones de productos de trabajo (por ejemplo, requisitos, diseño o historias de usuarios) tan pronto como estén disponibles las versiones preliminares de los documentos relevantes. Este principio resalta el papel del evaluador como especialista en el control de calidad del software y es un ejemplo de la implementación del principio de “desplazamiento a la izquierda” (ver Sección 2.1.5). Durante estas revisiones, los evaluadores suelen encontrar muchas ambigüedades, errores o contradicciones que, si pasan desapercibidas, podrían provocar problemas graves en el futuro.

Ejemplo Una organización en nombre del Ministerio de Justicia está desarrollando el producto JudgeLottery, un sistema para la asignación aleatoria de casos a los jueces. La Tabla 2.1 recopila ejemplos de actividades de desarrollo y actividades de prueba correspondientes.

### 2.1.3 Las pruebas como impulsor del desarrollo de software

El desarrollo basado en pruebas (TDD), el desarrollo basado en pruebas de aceptación (ATDD) y el desarrollo basado en el comportamiento (BDD) son tres métodos populares de desarrollo de software basados en el enfoque de prueba primero. Este enfoque supone que antes de la

Tabla 2.1 Ejemplos de actividades de fabricación y actividades de prueba correspondientes

Fase	Actividad manufacturera	Actividad del probador
Requisitos	Recopilar requisitos del sistema, crear documentación de requisitos.	Inspección de los requisitos de comprobabilidad y su cumplimiento con los requisitos comerciales, como los requisitos de la Ley de Tribunales Generales.
Diseño	Diseño de base de datos	Inspección del diseño de la base de datos para comprobar el cumplimiento de los requisitos del sistema.
Implementación	Implementación del front-end (aplicación basada en web)	Pruebas de usabilidad, pruebas de integración sistema-base de datos, pruebas de sistemas
Despliegue	Instalación del sistema en el ministerio.	Pruebas de aceptación (beta) del sistema en el entorno de destino por parte de los presidentes de los tribunales

Cuando se escribe el código fuente que implementa una función específica, se crea una prueba para esa función. Es un ejemplo de cómo las pruebas pueden ser un factor rector en el desarrollo de software.

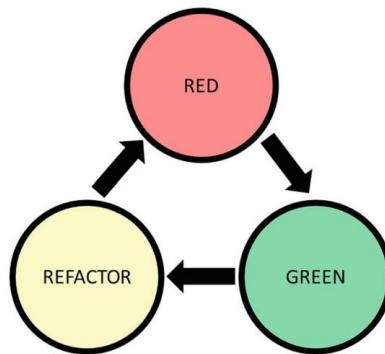
El enfoque de “probar primero” se origina en la programación extrema (XP) y es una de las prácticas centrales del desarrollo ágil de software. Este enfoque puede parecer contradictorio (normalmente creamos algo primero y lo probamos después), pero tiene algunos beneficios importantes [23]:

- Las pruebas reemplazan el método de prueba y error. En lugar de, por ejemplo, comprobar mediante prueba y error si una función implementada funciona correctamente, se crea un conjunto de casos de prueba significativos con valores de entrada y salida estrictamente definidos y luego se escribe código para pasar estas pruebas.
- Los casos de prueba proporcionan retroalimentación objetiva sobre el progreso. Cada prueba superada es una evidencia objetiva de que el proyecto está avanzando.
- Los casos de prueba reemplazan las especificaciones. Dado que las pruebas se escriben antes de crear el código, no son solo una colección de procedimientos de control, sino que también definen el comportamiento de muestra del objeto de prueba, convirtiéndose en un ejemplo de “documentación viva”. [24]. Un cambio en los requisitos requiere un cambio en las pruebas, por lo que obliga a un cambio en la documentación, haciendo que la documentación sea relevante en cualquier punto del proyecto. • El enfoque de “prueba primero” mejora la calidad de las interfaces públicas (API).

Las pruebas de componentes y las pruebas de integración de componentes utilizan los métodos públicos de las clases bajo prueba y, dado que las pruebas se escriben antes de que se cree el código, las pruebas definen los nombres de los métodos públicos de la clase bajo prueba, sus parámetros y ejemplos de método. uso. El diseño de las pruebas prácticamente se convierte en la definición de la interfaz. • El enfoque de “prueba

primero” mejora la capacidad de prueba del código que se está desarrollando. Una vez implementadas las pruebas, el desarrollador debe escribir código que las supere. Esto implica que el código debe invocar las interfaces requeridas por las pruebas. Entonces, en lugar de escribir pruebas que verifiquen el código existente, el desarrollador debe escribir código que sea “compatible” con las pruebas que se escribieron anteriormente. Al mismo tiempo, debido a que el código está escrito para pasar pruebas específicas, el programador tiene un mejor control y logra más fácilmente un mayor nivel de cobertura del código con las pruebas.

Fig. 2.9 Proceso de desarrollo basado en pruebas



La principal diferencia entre TDD, BDD y ADD es que TDD se centra en las pruebas de componentes, BDD se centra en las pruebas del comportamiento del sistema y ATDD se centra en las pruebas de aceptación del sistema. Por lo tanto, se puede decir que TDD funciona en los niveles de prueba de componente y de integración de componentes, BDD funciona en el nivel de prueba de sistema y de integración de sistema, y ATDD funciona en el nivel de prueba de aceptación (ver Sección 2.2.1). Los casos de prueba resultantes deberían ser adecuados para su uso en pruebas de regresión automatizadas. Cada uno de estos enfoques implementa el principio de prueba "Las pruebas tempranas ahorran tiempo y dinero" (ver Sección 1.3) y sigue el enfoque de "desplazamiento hacia la izquierda" (ver Sección 2.1.5) en el sentido de que las pruebas se definen antes de escribir el código.

#### Desarrollo basado en pruebas (TDD)

TDD utiliza casos de prueba de componentes automatizados para guiar el desarrollo del código y generalmente lo realiza un desarrollador. El proceso TDD es el siguiente:

- El desarrollador crea un nuevo caso de prueba, correspondiente a un requisito específico para el código.
  - Se ejecutan todas las pruebas existentes. La nueva prueba debería fallar porque aún no existe el código correspondiente (el paso "ROJO" en la Fig. 2.9). La primera ejecución de la nueva prueba es verificar si la prueba se compila. A su vez, si la prueba pasa en la primera ejecución, lo más probable es que haya un problema con la prueba en sí, porque debería fallar, ya que aún no hay un código correspondiente escrito. La ejecución de las pruebas restantes actúa como pruebas de regresión (consulte la Sección 2.2.3).
  - El desarrollador escribe una cantidad mínima de código suficiente para pasar la nueva prueba. También deben pasar todas las pruebas escritas previamente (el paso "VERDE" en la Fig. 2.9).
  - El desarrollador refactoriza el código (si es necesario) para mantener alta la calidad del código, ya que TDD se lleva a cabo en muchos ciclos cortos de "código de prueba", lo que puede degradar la calidad del código en sí (el paso "REFACTOR" en la Fig. 2.9). Después de la refactorización, el desarrollador vuelve a ejecutar todas las pruebas para asegurarse de que la refactorización no haya roto nada. •
- Los pasos anteriores
- se repiten siempre que todavía quede algo de código por escribir.

El enfoque TDD normalmente utiliza un marco de prueba como xUnit para admitir pruebas automatizadas. Un único ciclo de "código de prueba" suele ser muy corto y puede durar incluso menos de 1 o 2 minutos.

Al utilizar el enfoque TDD, los desarrolladores obtienen cierta independencia en las pruebas. No tienen una conexión emocional con el código porque aún no lo han creado. Por lo tanto, sólo a través del diseño de pruebas, los desarrolladores pueden probar o establecer supuestos de implementación para un componente. Por lo tanto, sus pruebas serán más sólidas que si se escribieran después de que se haya implementado el componente.

#### Desarrollo impulsado por el comportamiento (BDD)

En TDD (arriba), las pruebas pueden incluso hacer referencia a unas pocas líneas de código individuales. En BDD, desarrollo impulsado por el comportamiento, las pruebas se centran en el comportamiento esperado del sistema [25], por lo que operan a un nivel ligeramente superior que las pruebas de componentes en el enfoque TDD. BDD utiliza un enfoque colaborativo para generar criterios de aceptación en lenguaje sencillo, generalmente como parte de una historia de usuario que pueden entender todas las partes interesadas (ver Sección 4.5).

Los criterios de aceptación generalmente se escriben utilizando marcos. El formato típico para los criterios de aceptación es Dado/Cuándo/Entonces. BDD se puede automatizar. El marco, basado en la historia del usuario y los criterios de aceptación asociados, genera automáticamente el código del caso de prueba y lo ejecuta.

El principio del marco BDD se muestra a continuación. La prueba se puede registrar en forma de un documento de texto legible y comprensible escrito en un lenguaje como Gherkin (la prueba junto con el código siguiente es un ejemplo ligeramente modificado de <https://automationrhapsody.com/introduction-to-cucumber-and-bdd-con-ejemplos/>):

Característica: buscar una entrada en Wikipedia

Escenario: búsqueda directa de artículos

Término de búsqueda de entrada dado 'prueba'

Cuando haga clic en buscar

Luego aparece la página que contiene la frase 'pruebas'.

Esta prueba verifica que si un usuario escribe el término de búsqueda "Prueba" en el motor de búsqueda de Wikipedia, cuando el usuario hace clic en el botón "Buscar", el sistema devolverá una página que contiene el texto "prueba". Estos pasos del caso de prueba se implementan físicamente mediante código utilizando el marco BDD. Un fragmento de dicho código se muestra en el siguiente listado. En la línea anotada con @Given, el desarrollador le dice al script Gherkin anterior que busque la frase "Ingrese el término de búsqueda X" en la línea que comienza con la palabra "Given", donde el valor de X se asigna automáticamente a la variable searchTerm que es un parámetro del método searchFor asociado con la línea "Dada". El marco identifica este fragmento mediante el uso de la llamada expresión regular, que define un patrón y encuentra la cadena que coincide con él.

Si la cadena coincide con un patrón, el método searchFor se ejecuta con el parámetro X (en nuestro caso, la cadena "prueba"). Físicamente hace lo que necesita hacer: busca un cuadro de búsqueda de texto en la página web y le envía el valor de la variable X (es decir, la palabra "prueba"), utilizando las declaraciones correspondientes de la biblioteca Selenium WebDriver. una herramienta para la prueba automática de páginas web. De manera similar, el

Los métodos asociados con las anotaciones @When y @Then realizan las acciones correspondientes: hacer clic en el botón “Buscar” y verificar que el texto especificado aparece en la página.

```
paquete com.automationrhapsody.cucumber.parallel.tests.wikipedia;

importar org.openqa.selenium.By; importar
org.openqa.selenium.WebDriver; importar
org.openqa.selenium.WebElement; importar
org.openqa.selenium.firefox.FirefoxDriver;

importar pepino.api.java.After; importar
pepino.api.java.Antes; importar pepino.api.java.en.Dado;
importar pepino.api.java.en.Entonces; importar
pepino.api.java.en.Cuando;

importar estático junit.framework.Assert.assertTrue; importar estático
junit.framework.TestCase.assertFalse; importar org.junit.Assert.assertEquals estático;

clase pública WikipediaSteps {controlador
    WebDriver privado;

    @Antes
    public void before() { controlador =
        nuevo FirefoxDriver(); driver.navigate().to("http://
        en.wikipedia.org");
    }

    @Después
    vacío público después()
    { driver.quit();
    }

    @Given("^Ingrese el término de búsqueda '(.*?)'$")
    public
    void searchFor(String searchTerm) { WebElement searchField =
        driver.findElement(By.id("searchInput")); searchField.sendKeys(término de búsqueda); }

    @When("^Haga clic en buscar$")
    public void clickSearchButton() { WebElement
        searchButton = driver.findElement(By.id ("searchButton")); botónbuscar.click(); }
```

```

@Then("^Aparece la página que contiene la frase '(.*?)'$") public void
afirmarSingleResult(String searchResult) {
    Resultados de WebElement = controlador
        .findElement(By.cssSelector("div#mw-content-text.mw-content-ltr
            pag"));
    afirmarFalse(resultados.getText().contains(searchResult + "puede
        Referirse a:"));
    afirmarTrue(resultados.getText().startsWith(searchResult));
}
}

```

BDD sigue el concepto de "documentación viva", ya que la especificación es ejecutable en pruebas automatizadas. La especificación suele seguir el principio de "Especificación por ejemplo" [24], que describe el requisito mediante ejemplos típicos (comprobables) en lugar de explicar todas las posibles necesidades y excepciones.

#### Desarrollo basado en pruebas de aceptación (ATDD)

ATDD sigue el mismo procedimiento definido para TDD y BDD, pero con casos de prueba automatizados que se encuentran en el nivel apropiado para las pruebas de aceptación [26]. Estos casos de prueba de aceptación se derivan de criterios de aceptación generados conjuntamente por desarrolladores, evaluadores y representantes comerciales. Los criterios de aceptación se escriben desde la perspectiva del usuario y normalmente están en un formato fácil de entender (por ejemplo, Dado/Cuándo/Entonces conocido desde el enfoque BDD). La ATDD se explica en detalle más adelante (ver Sección 4.5).

El esquema de implementación de ATDD (ver Fig. 2.10) es el siguiente:

- Seleccionar una historia de usuario
- Escribir una prueba de aceptación
- Implementar una historia de usuario (codificación)
- Ejecutar una prueba de aceptación
- Refactorizar el código, si es necesario
- Obtener la aprobación de la historia de usuario (aprobación)

ATDD normalmente utiliza un marco de pruebas de aceptación automatizadas (como FitNesse) para respaldar las pruebas de aceptación automatizadas. ATDD, al igual que BDD, se guía por el concepto de "documentación viva".

### 2.1.4 DevOps y pruebas

Tradicionalmente, las áreas de desarrollo, pruebas y operaciones de software permanecían en silos separados, con prioridades completamente diferentes. El desarrollo se centró en la creación y entrega de software, las pruebas se ocuparon de controlar la calidad del software producido y las operaciones se centraron en implementar y respaldar el software. La separación de estas tres áreas provocó a menudo conflictos entre ellas.

DevOps es un enfoque para crear sinergia haciendo que el desarrollo, las pruebas y las operaciones trabajen juntos para lograr objetivos comunes (ver Fig. 2.11). DevOps requiere

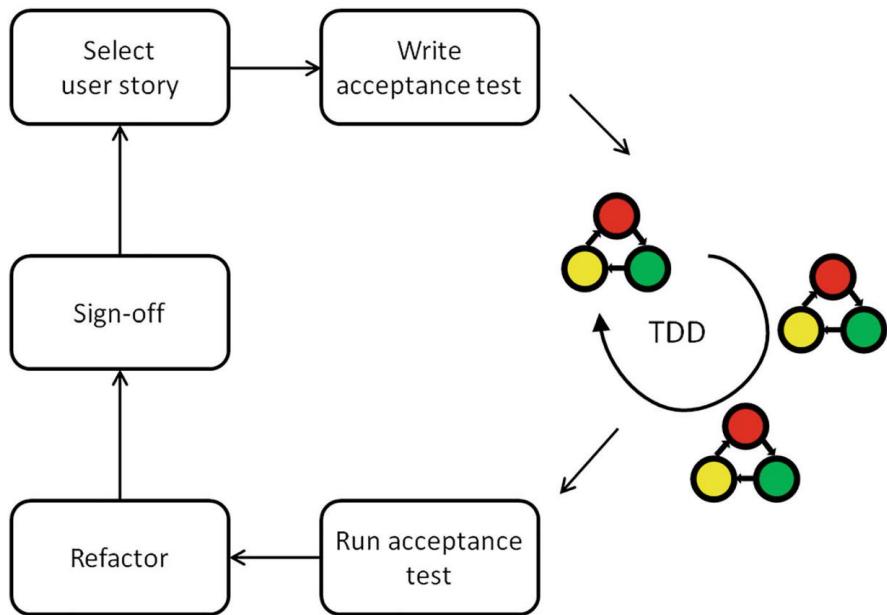
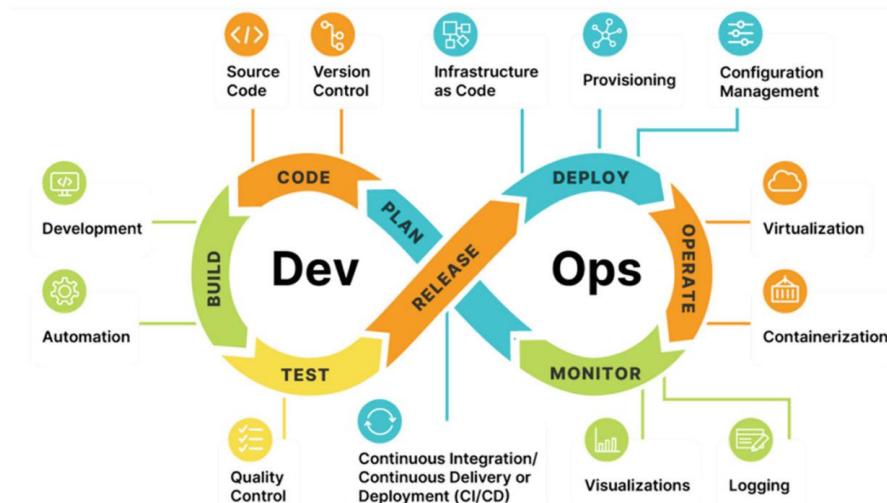


Fig. 2.10 Proceso de desarrollo basado en pruebas de aceptación

Fig. 2.11 Proceso de implementación de DevOps (fuente: [orangematter.solarwinds.com](http://orangematter.solarwinds.com))