

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.

¹ 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

Capítulo 1 Fundamentos de las pruebas.	25
1.1 ¿Qué son las pruebas?	27
1.1.1 Objetivos de la prueba.	28
1.1.2 Pruebas y depuración.	29
1.2 ¿Por qué son necesarias las pruebas?	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
1.3 Principios de prueba.	41
1.4 Actividades de prueba, software de prueba y funciones de prueba	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
1.5 Habilidades Esenciales y Buenas Prácticas en Pruebas.	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
Preguntas de muestra.	69
Capítulo 2 Pruebas a lo largo del ciclo de vida del desarrollo de software.	75
2.1 Pruebas en el contexto de un ciclo de desarrollo de software.	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
2.2 Niveles de prueba y tipos de prueba.	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
2.3 Pruebas de mantenimiento.	127
Preguntas de muestra.	130
Capítulo 3 Pruebas estáticas.	133
3.1 Conceptos básicos de las pruebas estáticas	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
3.2 Proceso de retroalimentación y revisión	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
Capítulo 4 Análisis y diseño de pruebas.	169
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
Capítulo 5 Gestión de las actividades de prueba.	251
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
Parte III Respuestas a preguntas y ejercicios	
Respuestas a preguntas de muestra.	313
Respuestas a las preguntas del cap. 1	313
Respuestas a las preguntas del cap. 2	317
Respuestas a las preguntas del cap. 3	321
Respuestas a las preguntas del cap. 4	323
respuestas a las preguntas del cap. 5	332
Respuestas a las preguntas del cap. 6	337
Soluciones a los ejercicios.	339
Soluciones a los ejercicios del cap. 4	339
Soluciones a los ejercicios del cap. 5	349
Parte IV Examen oficial de muestra	
Conjunto de exámenes A	355
Preguntas de muestra adicionales.	369
Conjunto de exámenes A: Respuestas.	379
Preguntas de muestra adicionales: respuestas.	391
Referencias.	399
Índice.	403

Acerca de los autores²

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), 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

²Los 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).

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). En 2002, la ASQF alemana (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:

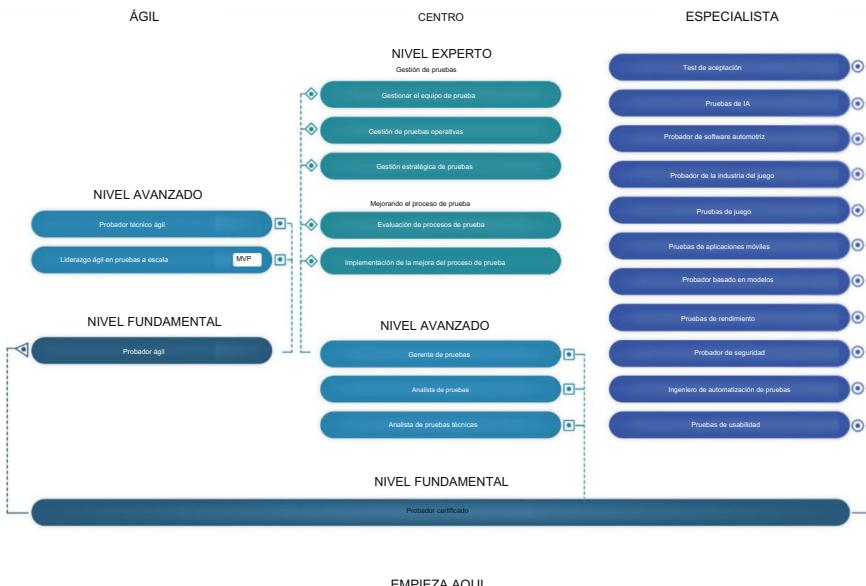
- Analista de pruebas técnicas (orientado a la tecnología, pruebas no funcionales, análisis estático, técnicas de prueba de caja blanca, trabajo con código fuente)
- Analista de pruebas (orientado al cliente, comprensión del negocio, pruebas funcionales, técnicas de prueba de caja negra y pruebas basadas 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.

Objetivos del Sistema Internacional de Cualificaciones

Fig. 1 Esquema de certificación oficial ISTQB® (fuente: 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. 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.

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. • 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	
Capítulo 1				
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	
2				
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	
Capítulo 3				
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	
Capítulo 4				
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	
Capítulo 5				
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	
Capítulo 6				
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), 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.

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 ¿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.

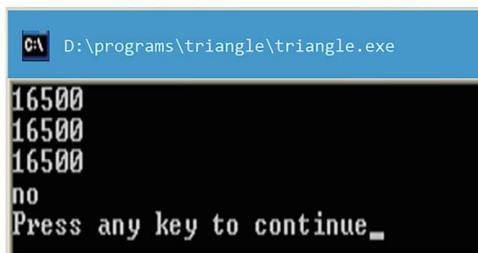
Cuando las pruebas dinámicas detectan una falla, un proceso de depuración típico consistirá en los siguientes pasos:

- 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

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.

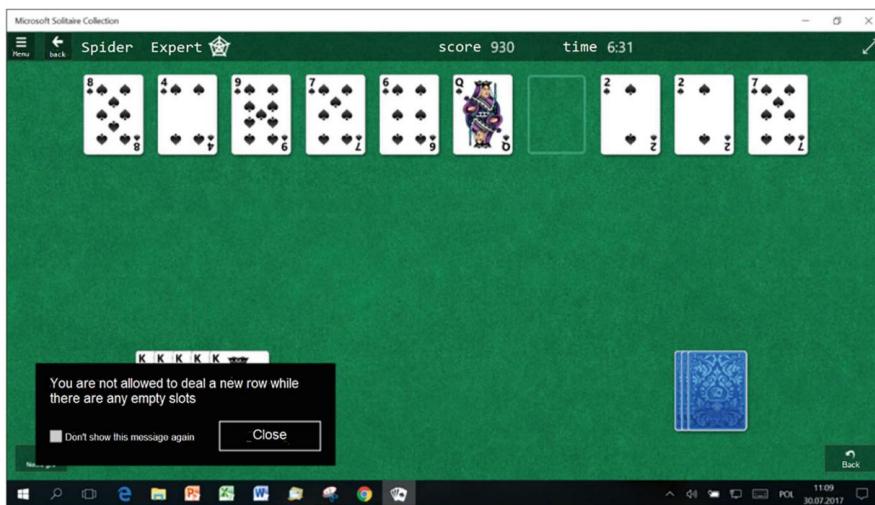


Fig. 1.2 Juego de solitario "Spider"

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 10⁻⁶.

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	Análisis 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 positivo¹ 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.

¹La 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
	{ 1.2700 9.037 847 025 1.2700 9.037 846 995 cosine 2.130476415 (23) 4.615925059 (-2) 2.130476415
	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)

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	
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

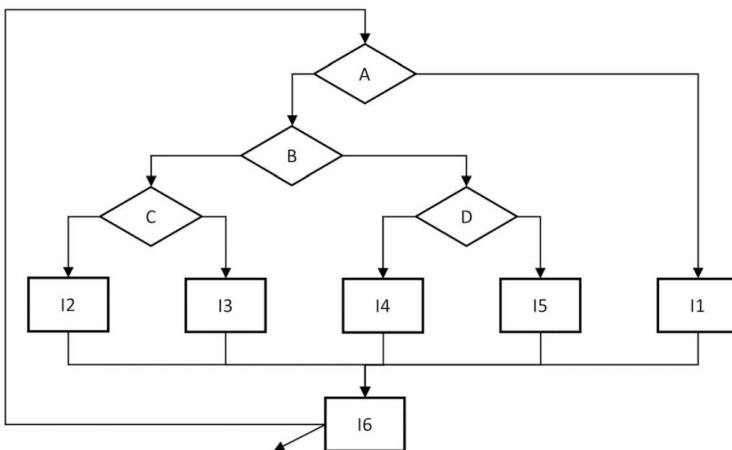


Fig. 1.6 Gráfico de flujo de control del fragmento de código a probar

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.

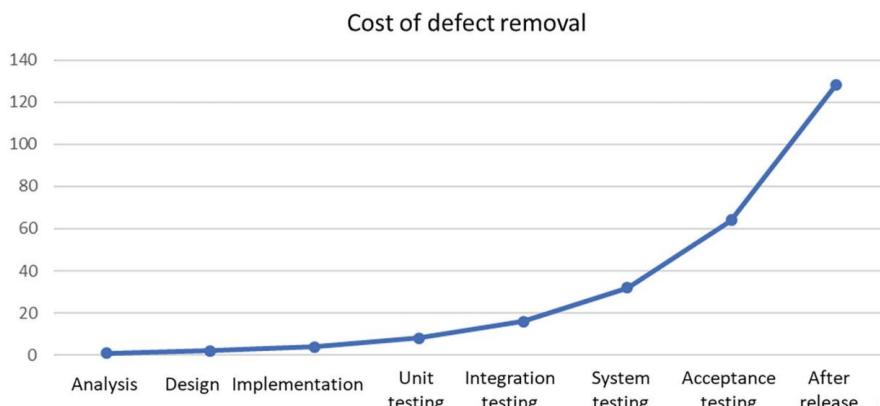


Fig. 1.7 Costo de la eliminación de defectos en función del tiempo

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

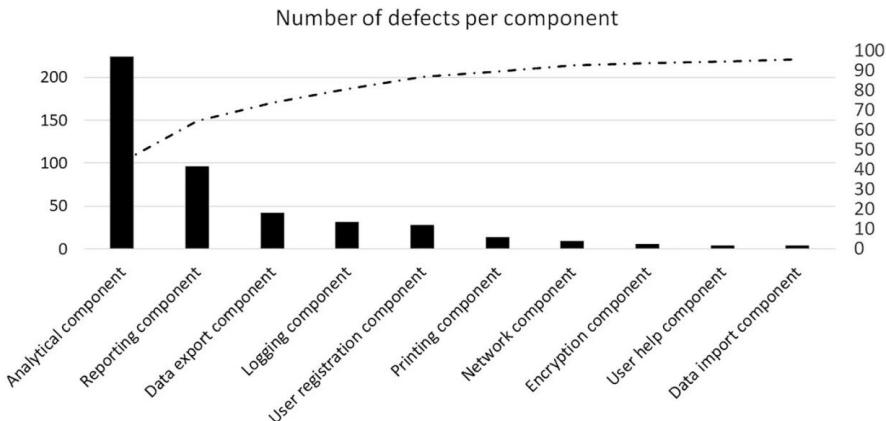


Fig. 1.8 Ejemplo de distribución tipo Pareto

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.

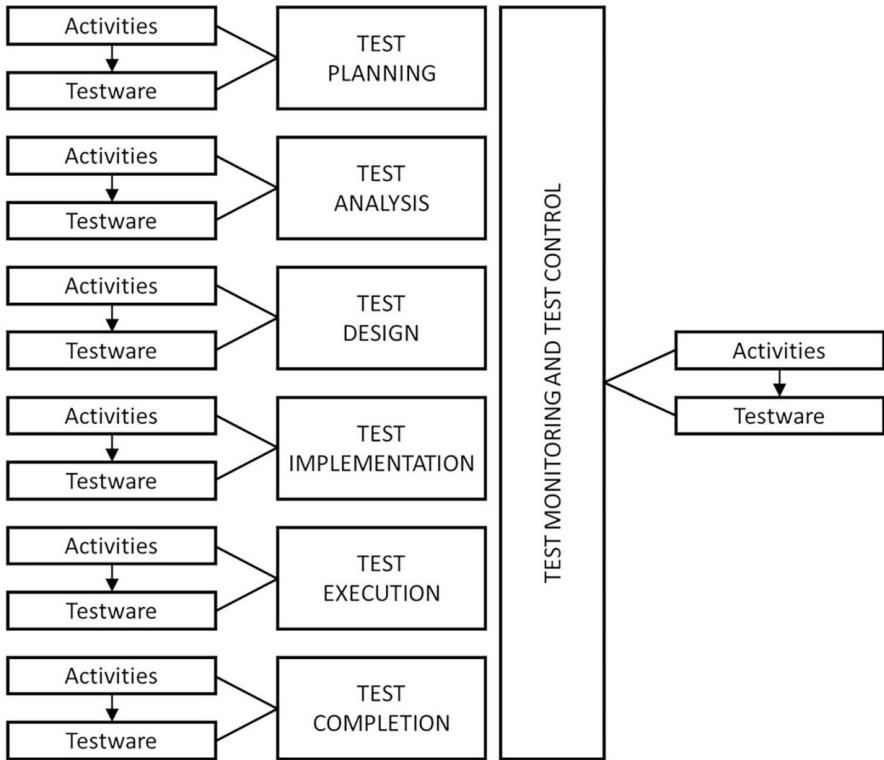


Fig. 1.10 Actividades y productos de trabajo en el proceso de prueba.

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 humo² 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.

² 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

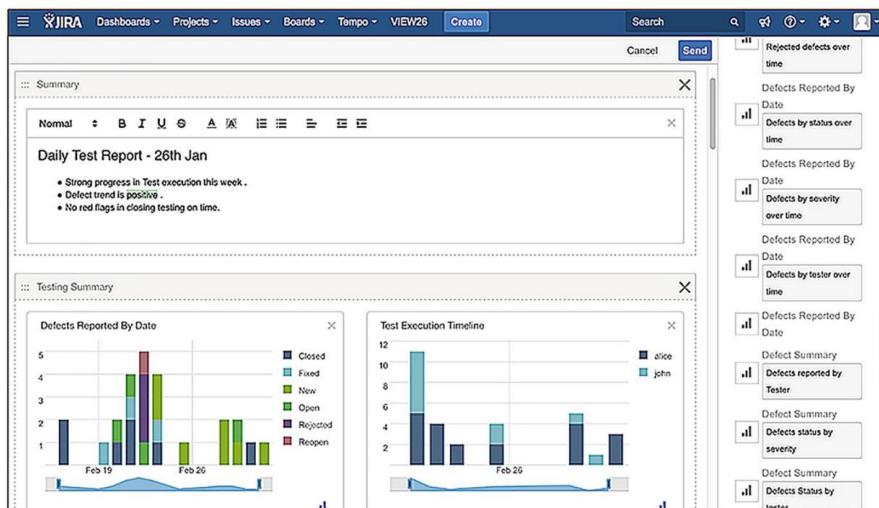


Fig. 1.11 Ejemplo de informe de prueba diario (fuente: community.atlassian.com)

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 www.our.com .	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 $10*0$, $0*10$ y $0*0$ 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.

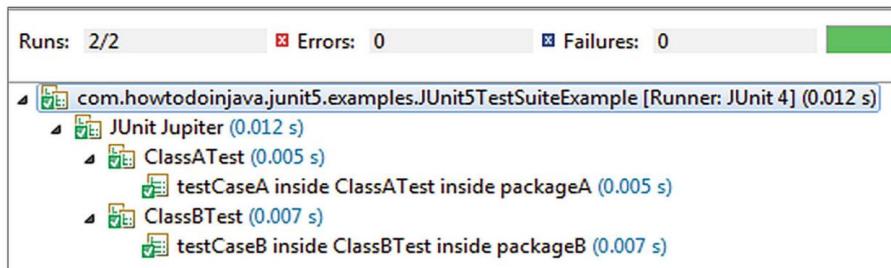


Fig. 1.12 Resultado de la ejecución de la prueba de componentes en JUnit (fuente: howtodoinjava.com)

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.

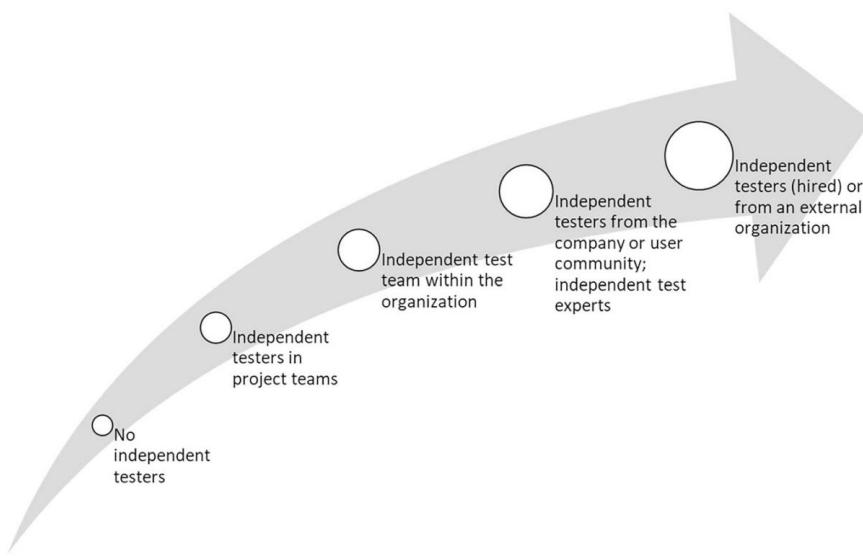


Fig. 1.13 Niveles de independencia de las pruebas

- 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 sistema en su conjunto cumple con los requisitos especificados.
Nivel de prueba	El sistema en su conjunto cumple con los requisitos especificados. una instancia específica de un proceso de prueba. Sinónimos: etapa de prueba.
Objeto de prueba	Sinónimos: etapa de prueba.
Tipo de prueba	el producto de trabajo a probar. 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.

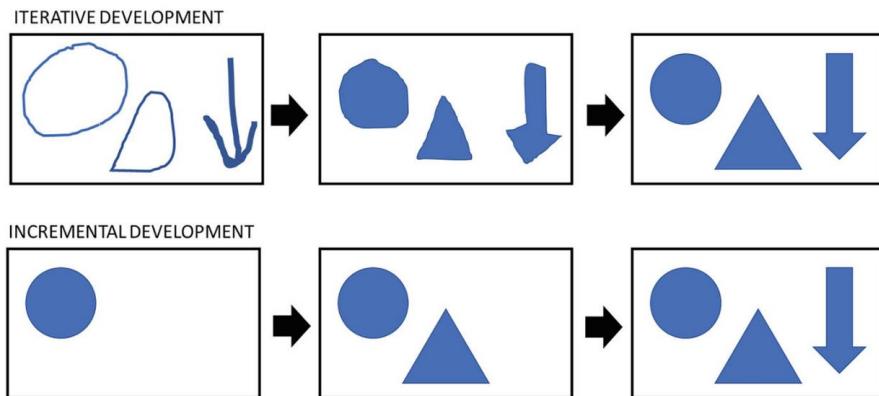


Fig. 2.1 Diferencia simbólica entre los modelos SDLC iterativos e incrementales

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.

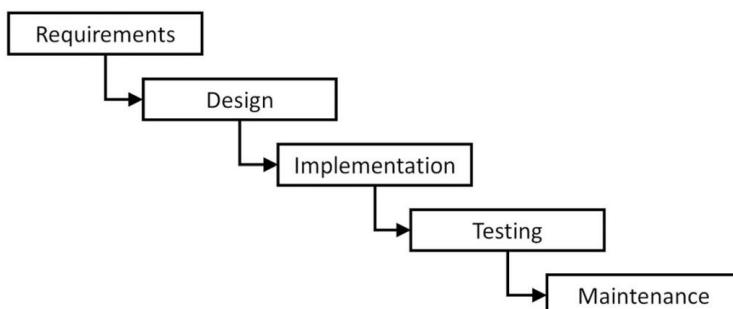


Fig. 2.2 Modelo de cascada

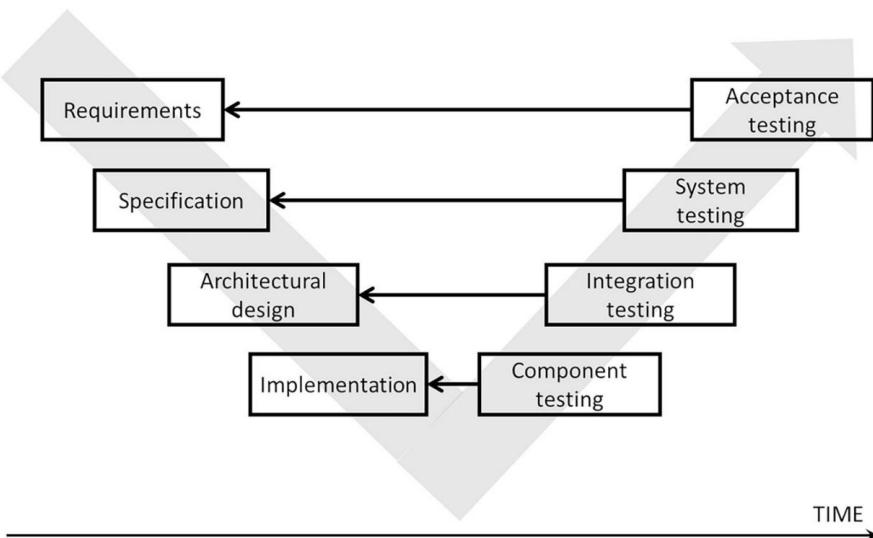


Fig. 2.3 Modelo V

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.

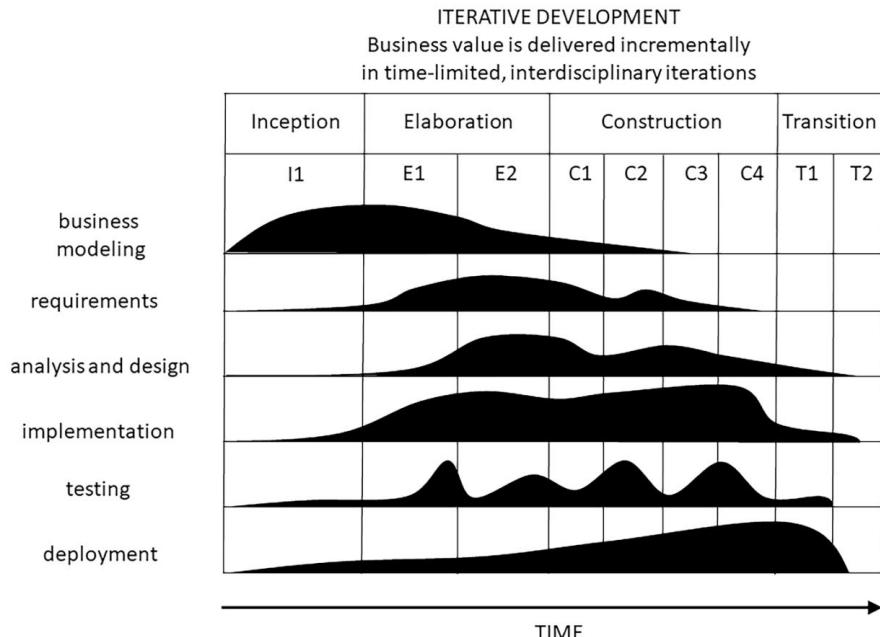


Fig. 2.4 Proceso unificado

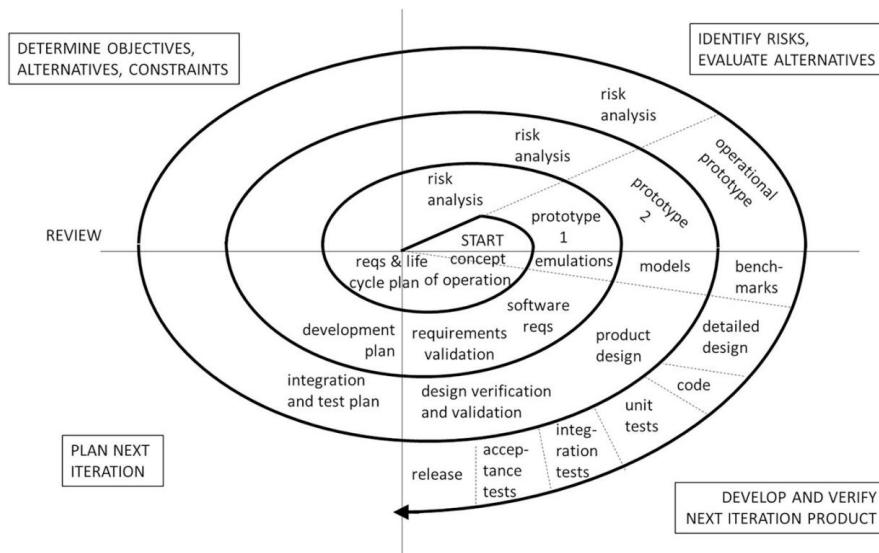


Fig. 2.5 Modelo espiral de Boehm

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

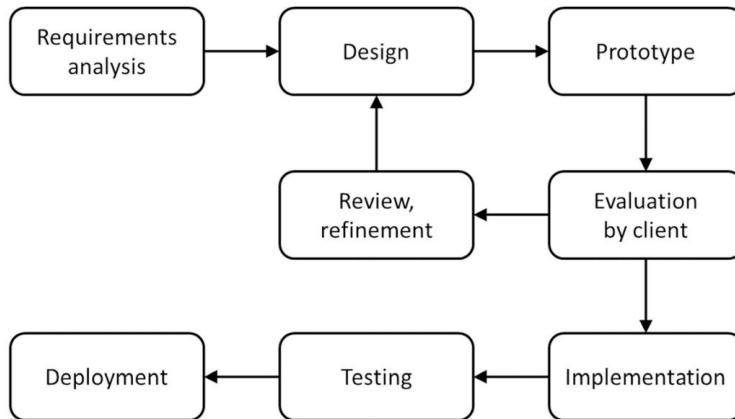


Fig. 2.6 Modelo de creación de prototipos

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.¹ 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.

¹ 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. Marca, 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.

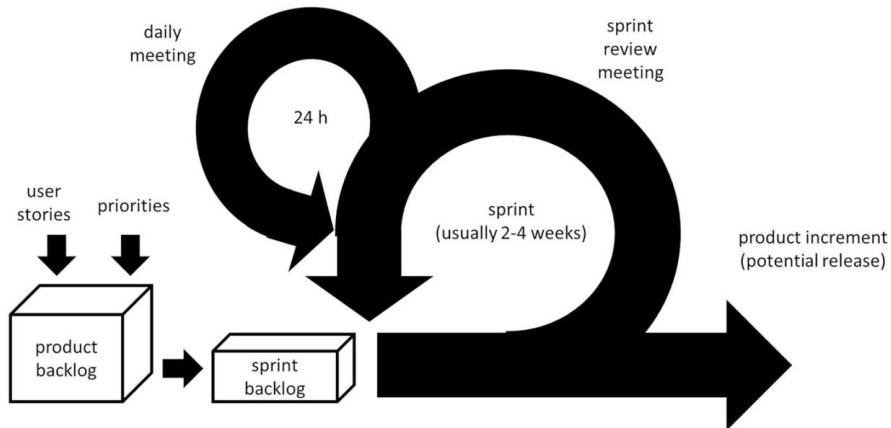


Figura 2.7 Melé

(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.

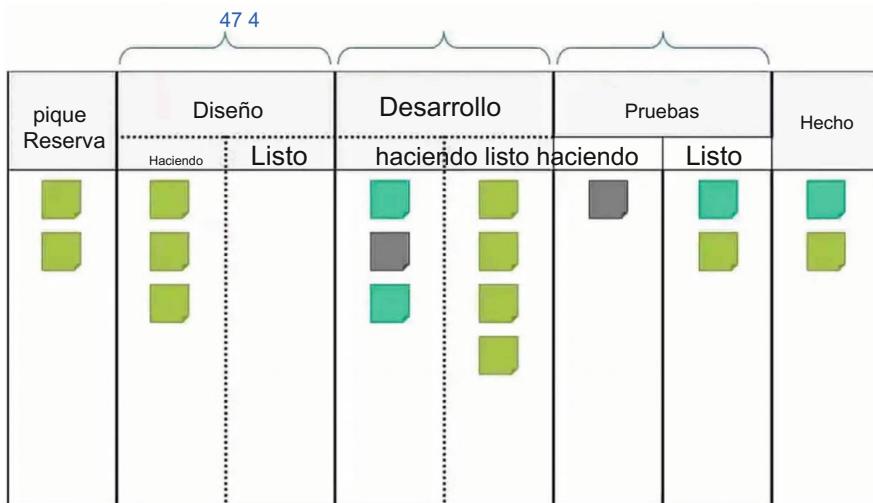


Fig. 2.8 Ejemplo de tablero Kanban (fuente: scrum.org)

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.

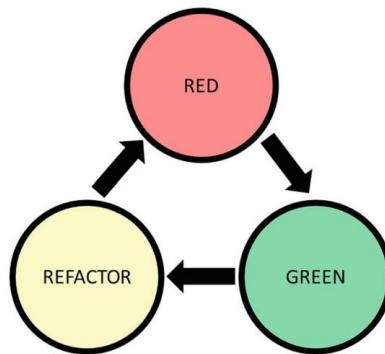
El enfoque de “probar primero” se origina en la programación extrema (XP) y es una de las prácticas centrales del desarrollo ágil de software. Este enfoque puede parecer contradictorio (normalmente creamos algo primero y lo probamos después), pero tiene algunos beneficios importantes [23]:

- 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 ahorrarán 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

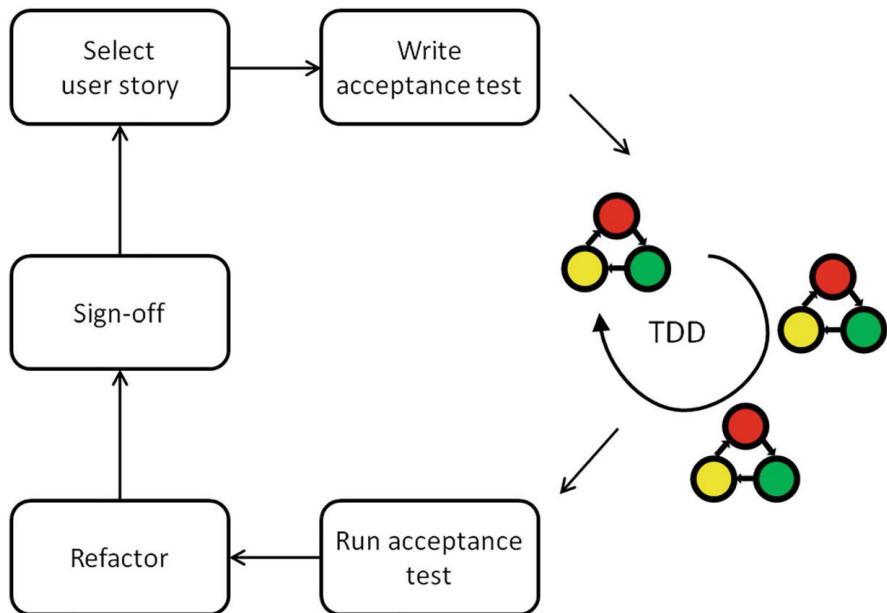
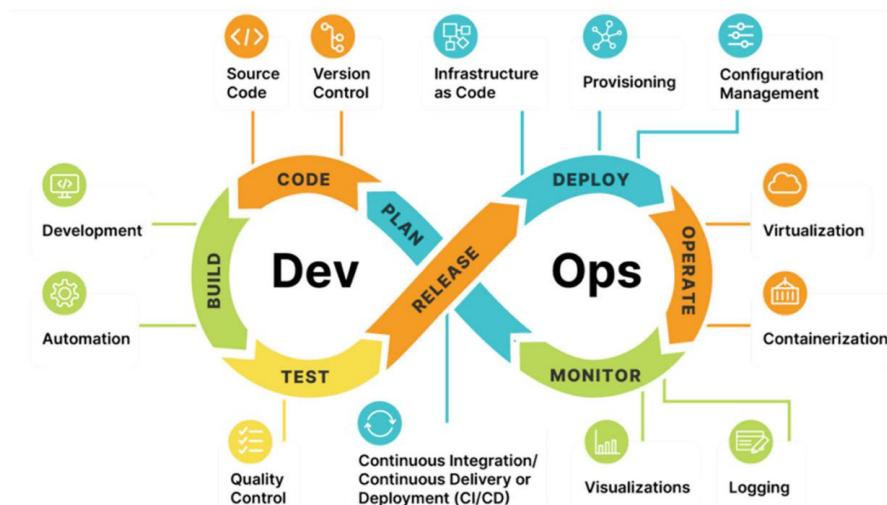


Fig. 2.10 Proceso de desarrollo basado en pruebas de aceptación

Fig. 2.11 Proceso de implementación de DevOps (fuente: orangematter.solarwinds.com)

un cambio cultural en la organización para cerrar la brecha entre desarrollo, pruebas y operaciones, tratando sus funciones con igual valor.

DevOps se basa en el uso de un modelo iterativo/incremental del SDLC respaldado por principios ágiles como autonomía del equipo, retroalimentación rápida y cadenas de herramientas integradas, así como prácticas técnicas como integración continua (CI), entrega continua de software o implementación continua de software. Estos principios y prácticas permiten a los equipos crear, probar y publicar código de calidad más rápidamente a través de lo que se conoce como canal DevOps [27]. Para ayudar a lograr una mayor calidad y una entrega más rápida, los canales de DevOps automatizan las actividades manuales siempre que sea posible. Idealmente, la gestión de la configuración, la compilación, la creación de software, la implementación y las pruebas se incluyen en un proceso de implementación único, automatizado y repetible.

En la práctica, la implementación efectiva de una canalización significa que se deben automatizar tantas pruebas como sea posible, porque de lo contrario las pruebas ralentizan la entrega de código y potencialmente permiten que los defectos se filtren en la producción. Idealmente, estas pruebas automatizadas deberían proporcionar información rápida sobre cualquier defecto encontrado, además de respaldar la integración continua, la entrega continua y la implementación continua.

A través del proceso de integración continua, el nuevo código cargado por los desarrolladores al repositorio de código se fusiona, compila y construye automáticamente. Idealmente, también se cargan pruebas de componentes (unitarios). Definitivamente esto es más fácil si se utiliza el enfoque TDD. Se ejecutan pruebas automáticas de componentes para garantizar que el código cargado pase estas pruebas y, a menudo, forman parte de conjuntos de pruebas de regresión. Además, en esta etapa se puede realizar un análisis estático del código. La retroalimentación al desarrollador es prácticamente instantánea, especialmente en situaciones en las que el código no se compila y no pasa las pruebas de humo o cuando la herramienta de análisis estático detecta anomalías. Dependiendo de la disponibilidad de los entornos de prueba y del tiempo requerido para las pruebas, también es posible ejecutar pruebas de integración de componentes en esta etapa. Estas pruebas de integración de componentes serán una combinación de nuevas pruebas para la nueva funcionalidad y pruebas de regresión para garantizar que la funcionalidad existente aún funcione como se espera. La principal ventaja de la integración continua es que proporciona a los desarrolladores información rápida si su código tiene errores. Cuando los comentarios llegan demasiado tarde, es probable que los desarrolladores continúen con el desarrollo basándose en el código defectuoso, lo que no es eficiente y requiere un cambio de contexto cuando llegan los comentarios.

La entrega continua de software puede considerarse una extensión de la integración continua e incluye otro nivel de pruebas realizadas en un entorno de prueba que es representativo del entorno de producción. Se pueden ejecutar pruebas de integración de componentes, pruebas de sistemas y pruebas no funcionales (muchas de las cuales son pruebas de regresión) como parte del proceso de entrega continua. Si todas estas pruebas pasan, el código se considera listo para la implementación, pero la decisión de implementarla la toma el equipo de DevOps. El equipo puede implementar el código de forma automática o manual.

La implementación continua puede considerarse una extensión de la entrega continua. Se diferencia de este último en que tanto la decisión de implementación como la implementación en sí están automatizadas. Con una implementación continua, el equipo de DevOps intenta lograr la mayor confianza posible en que las pruebas automatizadas detectarán cualquier defecto que no deba escapar al entorno de producción. Las organizaciones suelen pasar de la entrega continua a la implementación continua cuando obtienen suficiente

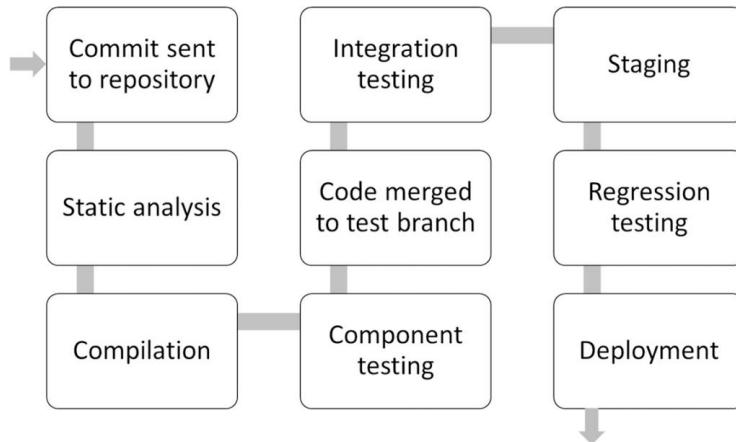


Fig. 2.12 Ejemplo de canal de implementación

confianza en el proceso de implementación automatizada. La infraestructura como código (IaC) también es una práctica común utilizada por los equipos de DevOps. Admite la configuración automatizada de entornos de prueba y producción para respaldar la entrega y la implementación continuas.

En la figura 2.12 se muestra un ejemplo de canal de implementación . Después de que el desarrollador envía el código al repositorio, se inicia un proceso automático que, si no ocurren problemas en el camino, conduce al lanzamiento automático de una nueva versión del software. Una vez enviado el código, se somete a un análisis estático (por ejemplo, se comprueba si el código tiene el formato correcto, si la denominación de las variables sigue las reglas de la organización, si el código tiene una complejidad ciclomática suficientemente baja,² etc.). Luego se compila el código y se ejecutan pruebas de componentes. Una vez aprobados, el nuevo código se fusiona con el código base existente y se pueden ejecutar pruebas automáticas de integración de componentes. Una vez aprobados, el código se implementa en un entorno de preproducción (similar o idéntico al entorno de producción) y se ejecutan pruebas de regresión para ver si el nuevo código ha causado problemas en otras partes del software. Una vez que pasan las pruebas de regresión, el código se entrega al cliente.

Si en algún momento de este proceso algo sale mal (por ejemplo, un resultado de análisis estático negativo, un error de compilación, una prueba fallida, etc.), el proceso se detiene inmediatamente y los desarrolladores reciben la retroalimentación adecuada. Luego pueden corregir el código y volver a enviarlo al repositorio reiniciando el proceso de implementación.

Una gran ventaja del proceso de implementación automática, respaldado por el proceso de gestión de código (control de versiones), es que en cualquier punto del ciclo de desarrollo, el código se compila, es ejecutable y está listo para su lanzamiento. Cualquier cambio en el código activa automáticamente el proceso de implementación y los desarrolladores reciben información inmediata.

²La complejidad ciclomática, o complejidad de McCabe, es una métrica de software desarrollada por Thomas J. McCabe en 1976 y utilizada para medir la complejidad de la estructura de un programa. La base para el cálculo es el número de puntos de decisión en el diagrama de flujo de control de un programa determinado.

Comentarios sobre el nivel de calidad del código. En caso de cualquier problema, el código vuelve a la versión (funcional) anterior a los cambios.

Desde una perspectiva de prueba, los beneficios del enfoque DevOps son los siguientes:

- Comentarios rápidos sobre la calidad del código y si los cambios afectan negativamente al código existente.
- CI promueve un enfoque de desplazamiento a la izquierda en las pruebas (ver Sección 2.1.5) al alentar a los desarrolladores a enviar código de alta calidad acompañado de pruebas de componentes y análisis estático.
- DevOps facilita entornos de prueba estables al promover la automatización continua procesos de integración y entrega continua (CI/CD).
- Aumenta la visión sobre las características de calidad no funcionales (por ejemplo, rendimiento, fiabilidad).
- La automatización a través de un proceso de entrega reduce la necesidad de realizar tareas manuales repetitivas. pruebas.
- El riesgo de regresión se minimiza debido a la escala y variedad de procesos automatizados. pruebas de regresión.

Sin embargo, DevOps no está exento de riesgos y desafíos, en particular:

- Se debe definir, establecer y mantener el proceso de implementación de DevOps. • Deben existir y mantenerse herramientas de integración continua. • La automatización de pruebas requiere recursos adicionales y puede ser difícil de establecer y mantener.
- A veces los equipos se vuelven demasiado dependientes de las pruebas de componentes y realizan muy pocas pruebas de aceptación y del sistema.

2.1.5 Enfoque de desplazamiento a la izquierda

El principio de "Las pruebas tempranas ahoran tiempo y dinero" (ver Sección 1.3) a veces se denomina "desplazamiento a la izquierda" porque es un enfoque en el que las pruebas se realizan lo antes posible en un SDLC determinado. Shift-left generalmente implica que las pruebas deben realizarse antes (por ejemplo, sin esperar la implementación del código o la integración de componentes), pero no significa que las pruebas posteriores en el ciclo de desarrollo deban descuidarse.

Existen varias prácticas recomendadas que ilustran cómo lograr el desplazamiento hacia la izquierda en las pruebas, que incluyen:

- Uso de reseñas. Las revisiones se pueden realizar en las primeras etapas del ciclo de desarrollo, incluso antes de que se escriba el código. Una revisión de la especificación a menudo encuentra defectos potenciales en la misma, como ambigüedades, falta de integridad, inconsistencias, elementos superfluos o contradicciones. • El uso de integración y entrega continuas (consulte la Sección 2.1.4) proporciona retroalimentación rápida sobre la calidad del código y obliga a la creación de pruebas de componentes automatizadas para el código fuente a medida que se envía al repositorio de código.

- Realización de análisis estáticos. El análisis estático del código fuente se puede realizar antes de las pruebas dinámicas o como parte de un proceso automatizado, como un proceso de implementación de DevOps (consulte la Sección 2.1.4). •

Realizar pruebas no funcionales lo antes posible. Esta es una forma de desplazamiento a la izquierda, ya que las pruebas no funcionales generalmente se realizan más adelante en el ciclo de desarrollo del software, cuando están disponibles un sistema completo y un entorno de prueba representativo.

- Utilizar pruebas basadas en modelos. En este enfoque, el evaluador utiliza o crea un modelo del software, y una herramienta especial basada en este modelo crea automáticamente casos de prueba que alcanzan un determinado criterio de cobertura. Por lo general, el modelo se puede crear antes de que comience el trabajo de implementación.

Se pueden ver bien ejemplos del uso del enfoque de desplazamiento a la izquierda en algunos SDLC.

Modelos o prácticas de desarrollo:

- En el modelo V, las actividades de prueba (por ejemplo, diseño de prueba o creación de prototipos) ya toman lugar cuando se inicie la correspondiente fase de desarrollo.
- En los modelos iterativos, las pruebas suelen estar presentes en cada iteración, por lo que comienzan temprano en el proyecto.
- En el desarrollo basado en pruebas (un enfoque de “prueba primero”), escribir pruebas antes de escribir código en TDD o ATDD (ver Sección 2.1.3) naturalmente mueve las actividades de prueba a una etapa temprana del ciclo de desarrollo.

2.1.6 Retrospectivas y mejora de procesos

Las retrospectivas (también conocidas como reuniones de lecciones aprendidas) se llevan a cabo al final de un proyecto o cuando se alcanza un hito en un proyecto, como al final de una iteración o al finalizar el nivel de prueba. Durante estas reuniones, los participantes discuten qué funcionó bien (qué tuvo éxito), qué no funcionó y qué se puede mejorar, y cómo realizar mejoras y mantener los éxitos en el futuro. Las reuniones cubren específicamente temas como:

- Procesos y organización (por ejemplo, ¿los procedimientos utilizados retrasaron, obstaculizaron o más bien acelerar y facilitar la realización de determinadas tareas?)
- Personas (p. ej., ¿tenían los miembros del equipo los conocimientos y habilidades adecuados, o hay deficiencias en esta área y la necesidad de capacitación?) • Relaciones (p. ej., ¿trabajó el equipo en conjunto sin problemas? ¿Fue la comunicación? ¿eficaz?)
- Herramientas (por ejemplo, ¿el uso de ciertas herramientas aumentó la efectividad o la eficiencia de las pruebas? ¿Adquirir una determinada herramienta ayudaría a aumentar la efectividad o la eficiencia?) • El producto que se desarrolló y probó (por ejemplo, ¿hay características que se pueden mejorar? ¿Existe algún nivel de deuda técnica que deba resolverse?)

Sin embargo, normalmente los participantes no se limitan a discutir áreas específicas.

Si existen acciones correctivas apropiadas contra los problemas identificados durante la

retrospectivamente, contribuyen a la autoorganización de los equipos y a la mejora continua del desarrollo y las pruebas. Los resultados de la retrospectiva deben registrarse y pueden formar parte del informe de finalización de la prueba (ver Sección 5.3.2).

Las retrospectivas pueden dar lugar a decisiones sobre mejoras relacionadas con las pruebas y, por tanto, contribuir a mejorar el proceso de prueba. En particular, las retrospectivas realizadas con éxito ofrecen los siguientes beneficios para las pruebas:

- Aumentar la efectividad de las pruebas (por ejemplo, más defectos detectados)
- Aumentar la eficiencia de las pruebas (por ejemplo, lograr el mismo objetivo con menos esfuerzo a través de el uso de herramientas)
- Mejorar la calidad de los casos de prueba (más probabilidades de detectar defectos, menos probabilidades de que ocurran defectos en las pruebas mismas) • Incrementar la satisfacción del equipo de pruebas (aumentar la moral, mejorar las relaciones del equipo, aumentar la efectividad del equipo)
- Mejorar la colaboración entre desarrolladores y evaluadores

Las retrospectivas también pueden abordar la capacidad de prueba de aplicaciones, historias de usuarios u otros requisitos, funciones o interfaces del sistema. El análisis de la causa raíz de los defectos también puede conducir a mejoras en las pruebas y el desarrollo. En general, los equipos deben implementar sólo unas pocas mejoras a la vez (por ejemplo, en una iteración determinada) para permitir una mejora continua a un ritmo constante.

El momento y la organización de la reunión dependen del modelo específico del ciclo de vida del desarrollo de software. Por ejemplo, en el desarrollo de software ágil, los representantes empresariales y el equipo ágil asisten a cada retrospectiva como participantes, mientras el facilitador organiza y dirige la reunión. En algunos casos, los equipos pueden invitar a otros participantes a la reunión.

Los evaluadores deben desempeñar un papel importante en las retrospectivas porque aportan su perspectiva única (ver Sección 1.5.3). Las pruebas se realizan en cada iteración o versión y contribuyen al éxito del proyecto. Se anima a todos los miembros del equipo a contribuir tanto en las actividades de prueba como en las que no son de prueba.

Las retrospectivas deben realizarse en una atmósfera de confianza mutua. Las características de una retrospectiva exitosa son las mismas que las de una revisión (ver Capítulo 3).

Un resultado típico de una retrospectiva es la selección y priorización de una (y sólo una) acción de mejora que se agregará al trabajo pendiente de la siguiente iteración. La acción de mejora puede mirar hacia afuera (relacionada con el producto) o hacia adentro (relacionada con el equipo).

Al comprometerse a implementar una nueva acción de mejora por iteración, el equipo pretende lograr una mejora continua.

No es deseable seleccionar múltiples acciones de mejora para la siguiente iteración, ya que reduce el valor potencial para el cliente de la siguiente iteración, pero también porque hace imposible identificar de forma única el impacto (positivo) de la acción de mejora.

2.2 Niveles de prueba y tipos de prueba

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 prueba FL-2.2.3 (K2)

Distinguir las pruebas de confirmación de las pruebas de regresión

Los niveles de prueba son grupos de actividades de prueba que se organizan y gestionan juntas.

Cada nivel de prueba es una instancia del proceso de prueba que consta de las actividades descritas en la Sección. 1.4, realizado para software en un nivel de desarrollo determinado, desde componentes individuales hasta sistemas completos o sistemas de sistemas. Estos niveles están vinculados a otras actividades realizadas como parte del ciclo de vida del desarrollo de software.

Los niveles de prueba se caracterizan en particular por atributos tales como:

- Objeto de prueba 
- Objetivo de prueba
- Base de prueba
- Defectos y fallas

Enfoques y responsabilidades específicos

Las pruebas asociadas con una característica específica de un objeto de prueba se denominan se refiere pruebas. La diferencia entre niveles de prueba y tipos de prueba es que los niveles de prueba al tipo. las fases del ciclo de vida del desarrollo y la etapa de avance en el desarrollo del producto, mientras que los tipos de prueba tienen en cuenta el objetivo de la prueba y el motivo de la prueba. Por lo tanto, los niveles y tipos de prueba son independientes entre sí, lo que significa que cada tipo de prueba se puede realizar en cualquier nivel de prueba. Volveremos sobre esta cuestión más adelante.

2.2.1 Niveles de prueba

El programa de estudios de Foundation Level describe los cinco niveles de prueba típicos y más comunes. Estos son:

- Pruebas de componentes
- Pruebas de integración de componentes
- Pruebas de sistemas
- Pruebas de integración de sistemas
- Pruebas de aceptación

Sin embargo, es importante recordar que esto es sólo un ejemplo de clasificación.

Las organizaciones de desarrollo de software pueden utilizar sólo algunos de estos niveles o tener otros niveles adicionales específicos de su organización.

Cada nivel de prueba requiere un entorno de prueba apropiado. Sin embargo, puede haber limitaciones con respecto a la cantidad de entornos de prueba disponibles. Normalmente, el

Los entornos de prueba mínimos disponibles se identifican como DTAP: entorno de desarrollo, entorno de prueba, entorno de aceptación y entorno de producción.

Para las pruebas de componentes, este entorno suele ser un entorno de pruebas unitarias, es decir, marcos de trabajo tipo xUnit (p. ej., JUnit), bibliotecas para crear los llamados objetos simulados (p. ej., EasyMock), etc. Para las pruebas de aceptación, por otro lado, un entorno de prueba similar a un entorno de producción es ideal, ya que una gran parte de los defectos de campo (es decir, los reportados por los usuarios después de que el software se entrega al cliente) son defectos creados por la interacción del sistema y el entorno.

2.2.1.1 Prueba de componentes

Objetivos de las pruebas de componentes

Las pruebas de componentes, también conocidas como pruebas unitarias o pruebas de programas, se centran en módulos que se pueden probar por separado. Los objetivos de este tipo de pruebas incluyen:

- Mitigación de riesgos de los componentes
- Comprobar la compatibilidad del comportamiento funcional y no funcional del componente con el diseño y las especificaciones
- Generar confianza en la calidad del componente
- Detección de defectos del componente
- Evitar que los defectos alcancen niveles más altos de prueba

En algunos casos, especialmente en modelos SDLC incrementales e iterativos (por ejemplo, en proyectos ágiles), donde los cambios de código son continuos, las pruebas de regresión de componentes automatizadas son un componente clave para garantizar que los cambios realizados no hayan provocado un mal funcionamiento de otros componentes existentes.

Las pruebas de componentes suelen realizarse de forma aislada del resto del sistema (esto depende principalmente del modelo SDLC y del sistema específico), en cuyo caso puede ser necesario utilizar virtualización de servicios, arneses de prueba u objetos simulados (p. ej., stubs). o conductores). Este aislamiento significa que las pruebas de componentes deberían poder ejecutarse en cualquier orden y sus resultados deberían ser los mismos independientemente del orden de ejecución.

Los objetos simulados se utilizan en las pruebas de componentes para permitir la ejecución real de lotes de código que hacen referencia a otros objetos (por ejemplo, una base de datos) que aún no existen (en el momento de la prueba de componentes). Esta es una situación común, ya que las pruebas de componentes generalmente se ejecutan temprano en el ciclo de vida de desarrollo y es posible que muchos componentes del sistema aún no estén listos. Los objetos simulados permiten que el código se ejecute y así probar su propia funcionalidad; tenga en cuenta que aquí no estamos interesados en la comunicación o interacción del componente bajo prueba con el objeto simulado (por ejemplo, una base de datos simulada). La prueba de dicha comunicación sólo se realiza mediante pruebas de integración de componentes.

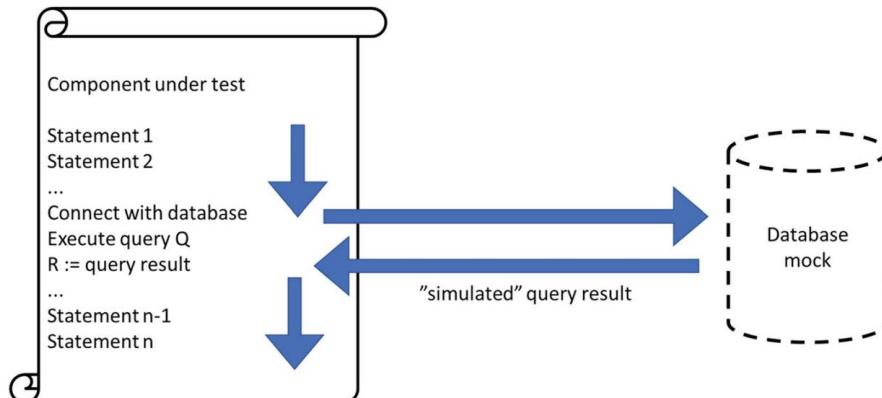


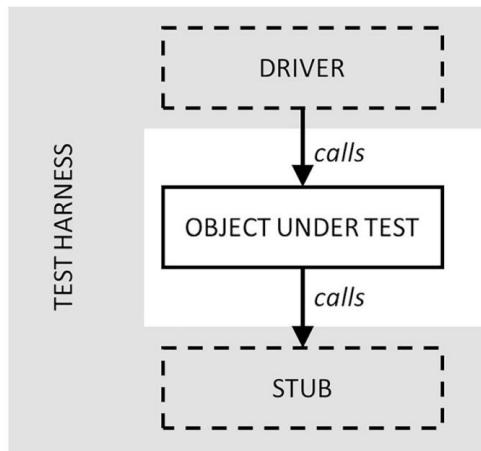
Fig. 2.13 Usando un objeto ficticio

Consideremos el ejemplo de la figura 2.13. Supongamos que estamos probando un script que en algún momento tiene que enviar una consulta a la base de datos, recibir el resultado de esa consulta y procesarla de una determinada manera. En la prueba de componentes, no nos interesaría la comunicación del script con la base de datos, si la consulta se transmitió y entendió correctamente y el resultado se entregó correctamente al script. Aquí sólo nos interesa la funcionalidad del componente en sí: si hace lo que debería hacer según la especificación. Por lo tanto, un sustituto de una base de datos podría incluso devolver siempre el mismo resultado de la consulta, ya que lo importante aquí no es la forma del resultado sino lo que hace el script con la consulta recibida.

Debido a la relación entre el llamante y el destinatario, se pueden distinguir dos tipos de objetos simulados: stub y driver. En principio, estos objetos cumplen la misma función, es decir, simulan el funcionamiento de otros objetos que aún no existen. La única diferencia es si nuestro componente bajo prueba llama al objeto ficticio o si el objeto simulado llama a nuestro componente bajo prueba. En el primer caso, la implementación de dicho objeto simulado se llama stub, en el segundo, controlador. Esta diferencia se muestra simbólicamente en la figura 2.14. Cuando utilizamos tanto stubs como drivers para realizar pruebas, dicho entorno se denomina arnés de prueba.

Las pruebas de componentes pueden cubrir la funcionalidad (p. ej., corrección de los cálculos), pero también características no funcionales del software (p. ej., rendimiento, confiabilidad), así como propiedades estructurales (p. ej., pruebas de declaraciones o decisiones).

Fig. 2.14 Muñón, conductor y arnés de prueba



Detalles sobre las pruebas de componentes

Base de prueba

Ejemplos de productos de trabajo que se pueden utilizar como base de prueba para las pruebas de componentes incluyen:

- Diseño detallado • Código
- Modelo
- de datos •

Especificaciones de componentes

Objetos de prueba

Los objetos de prueba típicos para pruebas de componentes incluyen:

- Módulos, unidades o componentes • Código y estructuras de datos • Clases o métodos
- (en programación orientada a objetos) • Componentes de bases de datos

Defectos y fallas típicos.

Ejemplos de defectos y fallas comunes detectados mediante pruebas de componentes son:

- Funcionalidad incorrecta (p. ej., inconsistente con las especificaciones del proyecto) • Problemas de flujo de datos • Código y lógica incorrectos

(continuado)

Los defectos generalmente se solucionan tan pronto como se detectan, a menudo sin una gestión formal de los mismos. Por lo tanto, la información de las pruebas de componentes (por ejemplo, cuántas veces un desarrollador corrigió un fragmento de código, qué tipo de defectos se detectaron, cómo se solucionaron, cuánto tiempo llevó la reparación) normalmente no se recopila. Sin embargo, cabe señalar que al informar defectos, los desarrolladores proporcionan información importante para el análisis de la causa raíz y la mejora del proceso.

Enfoques y responsabilidades específicos Las

pruebas de componentes generalmente las realiza el desarrollador (el autor del código), ya que siempre requiere acceso al código bajo prueba. Por lo tanto, los desarrolladores pueden alternar entre crear componentes y detectar/eliminar defectos. Los desarrolladores suelen escribir y ejecutar pruebas después de escribir el código de un componente en particular. Sin embargo, en algunas situaciones, especialmente en el desarrollo ágil de software, también se pueden crear casos de prueba automatizados para probar componentes antes de escribir el código de la aplicación (consulte la Sección 2.1.3).

2.2.1.2 Pruebas de integración de componentes y pruebas de integración de sistemas

Las pruebas de integración de componentes y las pruebas de integración de sistemas son, en principio, muy similares, por lo que estos dos niveles de prueba se analizarán juntos.

Las pruebas de integración de componentes  centran en las interacciones e interfaces entre los componentes integrados. Un componente aquí puede entenderse como una clase, archivo, función, procedimiento, paquete, etc. Las pruebas de este tipo se realizan después de probar los componentes y generalmente están automatizadas. En el desarrollo de software iterativo, las pruebas de integración de componentes suelen ser parte del proceso de integración continua.

Pruebas de integración del sistema  Se centra en las interacciones e interfaces entre sistemas, paquetes y microservicios. Este tipo de prueba también puede implicar interacciones con interfaces proporcionadas por organizaciones externas (por ejemplo, proveedores de servicios web). En este caso, la organización de desarrollo de software no controla las interfaces externas, lo que puede crear una amplia gama de problemas de prueba (relacionados, por ejemplo, con la reparación de defectos en el código creado por la organización externa que bloquea las pruebas, o con la preparación de entornos de prueba). . Las pruebas de integración del sistema pueden realizarse después de las pruebas del sistema o en paralelo con las pruebas en curso del sistema (esto se aplica tanto a los SDLC secuenciales como a los incrementales).

Objetivos de las pruebas de integración

Pruebas de integración típicas  objetivos son:

- Mitigar los riesgos que surgen de las interacciones componente/sistema. •
- Comprobar el cumplimiento del comportamiento funcional y no funcional de los componentes.
- interfaces del sistema/hient con el diseño y las especificaciones
- Generar confianza en la calidad de las interfaces utilizadas por los componentes/ sistemas
- Detectar defectos en interfaces y protocolos de comunicación. • Evitar que los defectos alcancen niveles más altos de prueba.

Detalles sobre las pruebas de integración

Base de

prueba La base de prueba de las pruebas de integración será cualquier tipo de documentación que describa la interacción o cooperación de componentes o sistemas individuales.

Ejemplos de productos de trabajo que se pueden utilizar como base de prueba para las pruebas de integración incluyen:

- Diseño de software y sistemas •

Diagramas de secuencia •

Especificaciones de interfaz/protocolo de comunicación • Casos de uso

• Arquitectura a nivel de componente y sistema • Flujos de trabajo

- Definiciones de

interfaces externas

Mencionar casos de uso como base de prueba para las pruebas de integración puede parecer extraño. Sin embargo, dejará de serlo si nos damos cuenta de que los casos de uso describen interacciones entre los llamados actores, es decir, más a menudo entre un usuario y un sistema o entre dos sistemas. Por lo tanto, los escenarios de casos de uso son muy adecuados para ser la base de las pruebas de integración.

Objetos de prueba

Los objetos de prueba típicos incluyen:

- Interfaces de programación de aplicaciones (API) que proporcionan comunicación entre componentes. • Interfaces

que proporcionan comunicación entre sistemas (p. ej., sistema a sistema, sistema a base de datos, microservicio a microservicio, etc.)

- Protocolos de comunicación entre componentes y sistemas.

Defectos y fallas típicos.

Ejemplos de defectos y fallas comunes detectados en las pruebas de integración de componentes son:

- Datos incorrectos o faltantes o codificación de datos incorrecta •

Secuenciación incorrecta o sincronización incorrecta de llamadas de interfaz • Interfaces incompatibles • Errores de

comunicación entre componentes • Fallo en el manejo o manejo incorrecto de errores de comunicación entre componentes

- Suposiciones incorrectas sobre el significado, las unidades o los límites de los datos transferidos entre componentes.

Ejemplos de defectos y fallas comunes detectados en las pruebas de integración de sistemas son:

(continuado)

- Estructuras de mensajes inconsistentes enviadas entre sistemas • Datos incorrectos o faltantes o codificación de datos incorrecta • Interfaces incompatibles • Errores de comunicación entre sistemas • Fallo en el manejo o manejo inadecuado de la comunicación entre sistemas errores
- Suposiciones incorrectas sobre el significado, las unidades o los límites de los datos transferido entre sistemas
- Incumplimiento de las normas de seguridad obligatorias

Enfoques y responsabilidades específicos Las pruebas

de integración de componentes y las pruebas de integración de sistemas deben centrarse en la integración misma. Por ejemplo, al integrar el componente A con el componente B, las pruebas deben centrarse en la comunicación entre estos módulos, más que en la funcionalidad de cada uno de ellos (esta funcionalidad debería ser previamente objeto de prueba de componentes). De manera similar, en el caso de integrar el sistema X con el sistema Y, las pruebas deben centrarse en la comunicación entre estos sistemas, y no en la funcionalidad de cada uno de ellos (esta funcionalidad debe ser objeto de prueba del sistema). A nivel de prueba de integración, se pueden utilizar pruebas funcionales, no funcionales y estructurales.

Las pruebas de integración de componentes suelen ser responsabilidad de los desarrolladores, mientras que las pruebas de integración de sistemas (debido a su naturaleza de alto nivel) suelen ser responsabilidad de los evaluadores. Siempre que sea posible, los evaluadores que realizan pruebas de integración de sistemas deben estar familiarizados con la arquitectura del sistema y sus comentarios deben tenerse en cuenta en la etapa de planificación de la integración.

El entorno de prueba, especialmente para las pruebas de integración de sistemas, debe, en la medida de lo posible, reflejar las características específicas del entorno de destino o de producción. Esto se debe a que una gran cantidad de fallas surgen de la interacción del sistema con el entorno en el que se encuentra. Un entorno de prueba idéntico o similar al entorno de destino permite detectar la mayoría de estos fallos y los defectos que los provocan durante la fase de prueba, antes de que el software se entregue al cliente.

Al igual que con las pruebas de componentes, hay situaciones en las que las pruebas de regresión de integración automática le permiten estar seguro de que los cambios realizados no han provocado un mal funcionamiento de las interfaces existentes.

Estrategias de integración La

planificación de pruebas de integración y estrategias de integración antes de construir componentes o sistemas permite que esos módulos o sistemas se produzcan de una manera que maximice la eficiencia de las pruebas. Existen varios enfoques para las estrategias de pruebas de integración. El antiguo programa de estudios de Foundation Level (v3.1) enumera varios de ellos. Se ocupan principalmente de la integración de componentes. Estos son:

(continuado)

- Estrategias basadas en la arquitectura del sistema, es decir, estrategias top-down y bottom-up. En una estrategia de arriba hacia abajo, primero se prueba la integración de un componente central con los componentes que llama, seguido de la integración de esos componentes con los componentes a los que llama, y así sucesivamente. Así, la integración se produce por “niveles” de receso de los componentes en la jerarquía de sus convocatorias. En una estrategia ascendente, la dirección de integración es la opuesta: comenzamos desde abajo e incluimos secuencialmente componentes que se encuentran en niveles superiores, llamando a los componentes ya llamados. Al utilizar una estrategia de integración basada en la arquitectura del sistema, a menudo se utilizan stubs (estrategia descendente) y controladores (estrategia ascendente), porque en general los módulos llamados o que llaman aún no están escritos.
- Estrategias funcionales basadas en tareas. En ocasiones, por determinadas razones, nos resulta importante probar primero la integración de un grupo de componentes que en conjunto son responsables de alguna funcionalidad que es importante para nosotros en este momento. Así, la estrategia es probar primero la integración de los componentes que componen esa funcionalidad y luego probar la integración de los demás módulos.
- Estrategia basada en secuencias de procesamiento de transacciones. Si estamos principalmente interesados en probar, por ejemplo, la ruta del flujo de información a través del sistema (por ejemplo, la ruta desde una entrada específica a una salida específica, para lograr la observabilidad del sistema lo antes posible), primero llevamos a cabo pruebas de integración de los componentes que se encuentran en este camino. A continuación, realizamos pruebas de integración de los componentes restantes.
- Estrategia basada en otros aspectos del sistema. Los dos últimos ejemplos de estrategias de integración consistieron en determinar el orden de las pruebas de integración debido a algunos criterios que eran importantes para nosotros. Se pueden imaginar muchas otras estrategias similares, como las basadas en la criticidad de los componentes o la frecuencia de uso de las comunicaciones entre componentes.

Ejemplo. Veamos con un ejemplo cómo se verían en la práctica las estrategias anteriores. Supongamos que nuestro sistema tiene la arquitectura que se muestra en la figura 2.15. Si dos componentes están conectados con una línea, significa que el componente que está arriba llama al que está debajo (por lo que A es el componente principal: llama a B, F y G; el componente B llama a C y D, etc.). Además, supongamos que B, C, D y E forman una única funcionalidad en lo que respecta al manejo de entradas. El componente E es responsable de tomar datos del usuario y el componente H de informar los resultados.

En una estrategia de arriba hacia abajo, el orden de las pruebas de integración será el siguiente:

1. Integración de AB, AF, AG (si B, F, G aún no están listos, use sus talones).
2. Integración de BC, BD, FH, GH (puede ser necesario utilizar stubs para C, D y H).
3. Integración de DE, DI, HI (puede ser necesario utilizar resguardos para E e I).

(continuado)

La estrategia ascendente es análoga; sólo se invierte el orden:

1. Integración de DE, DI, HI (es posible que necesite utilizar controladores para D y H)
2. Integración de BC, BD, FH, GH (puede ser necesario utilizar controladores para B, F y G)
3. Integración de AB, AF, AG (puede ser necesario utilizar un controlador para A)

En una estrategia basada en la funcionalidad, asumiendo que la función de manejo de entradas La realidad es una prioridad para nosotros; un ejemplo de orden de integración podría ser el siguiente:

1. Integración BC, BD, DE (funcionalidad de manejo de entrada)
2. Integración AB, DI (pruebas de integración del manejo de insumos con el ambiente)
3. Integración AF, AG, FH, GH, HI (otras conexiones)

A su vez, una estrategia basada en la secuencia de la transacción que se procesa puede tomar la siguiente forma:

1. Integración de ED, DB, BA, AF, FH (ruta de entrada-salida)
2. Integración de AG, BC, GH, DI, HI (otras conexiones)

Los métodos de integración incremental, como los descritos anteriormente, permiten, en el momento de la falla, localizar rápidamente el defecto. Si, en nuestro ejemplo, utilizamos una estrategia de arriba hacia abajo y la falla ocurre en el momento de la prueba FH, podemos estar casi seguros de que el defecto se relaciona con la comunicación FH y, por lo tanto, está localizado en uno de estos dos componentes, en lugar de en A o B, por ejemplo.

No se recomienda la integración big bang, que implica la integración de todos los componentes o sistemas a la vez, porque cuando ocurre una falla, generalmente no tenemos idea de qué pudo haber sido causada. El análisis de riesgos de las interfaces más complejas también puede ayudar a guiar las pruebas de integración en consecuencia.

Cuanto más amplio sea el alcance de la integración, más difícil será identificar un componente o sistema específico con defectos, lo que puede generar un mayor riesgo y más tiempo para diagnosticar los problemas. Esta es una de las razones por las que se utiliza comúnmente el método de Integración Continua (CI) para integrar software componente por componente (por ejemplo, integración funcional). Las pruebas de regresión automatizadas, que deben realizarse en múltiples niveles de prueba siempre que sea posible, suelen ser parte de la integración continua.

2.2.1.3 Prueba del sistema

Objetivos de las pruebas del sistema El

nivel de prueba del sistema es el nivel típico de la actividad de un evaluador (las pruebas de componentes y las pruebas de integración generalmente las realiza el desarrollador y las pruebas de aceptación, las realiza el cliente). Las pruebas del sistema se centran en el comportamiento y las capacidades de un sistema completo,

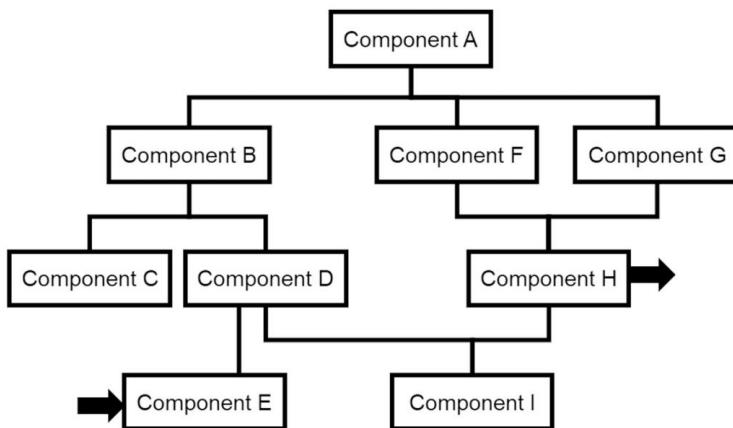


Fig. 2.15 Ejemplo de arquitectura de sistema

Sistema o producto ya integrado, a menudo teniendo en cuenta la totalidad de las tareas que puede realizar y los comportamientos no funcionales que exhibe mientras realiza esas tareas. Los objetivos de las pruebas del sistema son principalmente:

- Reducir el riesgo de mal funcionamiento del sistema. •
- Comprobar el cumplimiento del comportamiento funcional y no funcional del sistema con el diseño y especificaciones
- Comprobar la integridad del sistema y la corrección de su funcionamiento • Generar confianza en la calidad del sistema en su conjunto • Detectar defectos en el sistema • Evitar que los defectos alcancen el nivel de prueba de aceptación o producción

Para algunos sistemas, el propósito de las pruebas del sistema también puede ser verificar la calidad de los datos.

Como en el caso de las pruebas de integración de componentes o sistemas, las pruebas de regresión del sistema automatizadas brindan garantía de que los cambios realizados no han causado un mal funcionamiento de las funciones heredadas o de la funcionalidad general. Las pruebas del sistema a menudo dan como resultado información sobre la base de la cual las partes interesadas toman decisiones para transferir el sistema al uso de producción. Además, puede ser necesario realizar pruebas del sistema para cumplir con los requisitos de las regulaciones o normas/estándares aplicables.

Al igual que con el nivel de prueba de integración del sistema, el entorno de prueba debe, en la medida de lo posible, posible, reflejar las características específicas del entorno de destino o de producción.

En el nivel de prueba del sistema, la mayor parte de la retroalimentación se genera a partir del proceso de prueba. Esto incluye el número y tipo de defectos encontrados en el programa, el tiempo necesario para solucionarlos, cuándo se introdujo el defecto, etc.

Normalmente, las pruebas del sistema se centran en la verificación del sistema, comparando el comportamiento del sistema como se describe en los requisitos con el comportamiento real.

Las pruebas del sistema confirman que "el sistema se construyó correctamente".

Detalles sobre las pruebas del sistema**Bases de las**

pruebas Las pruebas del sistema se refieren a un sistema totalmente integrado, por lo que las bases de las pruebas deben referirse precisamente al sistema en su conjunto. Ejemplos de productos de trabajo que se pueden utilizar como base de prueba para las pruebas del sistema incluyen:

- Especificaciones de requisitos (funcionales y no funcionales) para el sistema.
- Tema y software
- Informes de análisis de riesgos
- Casos de uso
- Historias de usuarios y epopeyas
- Modelos de comportamiento del sistema
- Diagramas de estado
- Declaraciones de operación del sistema y manuales de usuario

Objetos de prueba

Los objetos de prueba típicos para las pruebas de sistemas incluyen:

- Sistema bajo prueba (el sistema en su conjunto)

Configuración del sistema y datos de configuración**Defectos y fallas típicos.**

Ejemplos de defectos y fallas comunes detectados por las pruebas del sistema incluyen:

- Cálculos incorrectos

Comportamiento funcional o no funcional incorrecto o inesperado del sistema • Flujos de control y/o flujos de datos incorrectos en el sistema • Problemas con el desempeño correcto y completo de las funciones funcionales generales tareas

- Problemas con el correcto funcionamiento del sistema en una producción ambiente

- Inconsistencia del funcionamiento del sistema con las descripciones contenidas en las declaraciones del sistema y manuales de usuario

Enfoques y responsabilidades específicos

Las pruebas del sistema deben centrarse en el comportamiento general del sistema en su conjunto, tanto en los aspectos funcionales como en los no funcionales. Las pruebas del sistema a menudo utilizan técnicas de prueba (consulte el Capítulo 4) que son más apropiadas para los aspectos particulares del sistema bajo prueba. Por ejemplo, se puede utilizar una tabla de decisiones para probar si el comportamiento funcional es coherente con la descripción de las reglas comerciales.

Las pruebas del sistema suelen ser realizadas por evaluadores independientes. Los defectos en las especificaciones (p. ej., historias de usuario faltantes o requisitos comerciales expresados incorrectamente) pueden dar lugar a malentendidos o desacuerdos sobre las

comportamiento esperado del sistema. Como resultado, pueden producirse resultados falsos positivos o falsos negativos (consulte la Tabla 1.2), lo que resulta en una pérdida de tiempo o una reducción del rendimiento de la detección de defectos, respectivamente. Por esta razón, entre otras cosas (para reducir la ocurrencia de las situaciones anteriores), los evaluadores deben participar en revisiones, refinamiento de historias de usuarios y otras actividades de pruebas estáticas lo antes posible en el SDLC.

Algunas pruebas de sistemas no funcionales pueden requerir que se realicen en un entorno de producción representativo. Ejemplos de tales pruebas podrían ser:

- Pruebas de rendimiento (p. ej., el entorno debe reflejar el rendimiento real de la red y los componentes no deben reemplazarse con objetos simulados, ya que esto puede interrumpir, por ejemplo, el informe del tiempo de respuesta de un módulo)
- Pruebas de seguridad (p. ej., los ataques de seguridad dependen de la configuración específica del sistema y del entorno en el que se encuentra el sistema)
- Pruebas de usabilidad (por ejemplo, las pruebas de usabilidad de la interfaz final deben realizarse en la interfaz real, no una simulación de esta interfaz)

2.2.1.4 Pruebas de aceptación

Objetivos de las pruebas de aceptación

Las pruebas de aceptación, igual que las pruebas del sistema, generalmente se centran en el comportamiento y las capacidades de todo el sistema o producto. Sin embargo, se lleva a cabo desde la perspectiva del usuario, no del equipo de desarrollo. Por tanto, tiene carácter de validación más que de verificación.

Los objetivos de las pruebas de aceptación son principalmente:

- Desarrollar la confianza del usuario en el sistema.
- Comprobar la integridad del sistema y su correcto funcionamiento desde el punto de vista del logro de objetivos comerciales

Las pruebas de aceptación pueden producir información para evaluar la preparación del sistema para su implementación y uso por parte del cliente (usuario). Los defectos también pueden detectarse durante las pruebas de aceptación, pero su detección no suele ser el objetivo principal de la prueba. Más bien, en esta etapa lo que nos preocupa es la validación del sistema, es decir, verificar si el sistema realmente satisface las necesidades comerciales del cliente. Las pruebas de aceptación confirmarán que "se construyó el sistema correcto".

En algunos casos, encontrar una gran cantidad de defectos en el nivel de las pruebas de aceptación puede incluso considerarse un riesgo importante para el proyecto. Después de todo, si se supone que el cliente debe verificar si el software resuelve su problema y el sistema "falla cada dos clics", tales pruebas son bastante inútiles: son sólo una señal de que se cometieron algunos errores fundamentales en el proceso de control de calidad, a través de en el que se filtraron muchos defectos a lo largo de todas las fases de fabricación hasta que el software se entrega al cliente.

Además, pueden ser necesarias pruebas de aceptación para cumplir con los requisitos establecidos en leyes o normas/regulaciones aplicables.

Las formas más comunes de pruebas de aceptación son:

- Pruebas de aceptación del usuario (UAT) •

Pruebas de aceptación operativa (OAT) • Pruebas de aceptación para el cumplimiento contractual y legal

Las pruebas de aceptación también se pueden dividir según el lugar donde se realizan:

- Prueba alfa: prueba realizada por el cliente en el sitio del productor, en una prueba ambiente
- Pruebas beta (también conocidas como pruebas de campo): pruebas realizadas por los clientes por su cuenta. entornos objetivo

Las diversas formas de pruebas de aceptación se describen en las siguientes subsecciones.

Pruebas de aceptación del usuario

Las pruebas de aceptación del usuario se llevan a cabo en un entorno de producción (simulado). El objetivo principal es generar confianza en que el sistema permitirá a los usuarios satisfacer sus necesidades, los requisitos que se le imponen y realizar los procesos de negocio con un mínimo de problemas, costos y riesgos.

Pruebas de aceptación operativa

Las pruebas de aceptación del sistema por parte de operadores o administradores generalmente se llevan a cabo en un entorno de producción (simulado). Las pruebas se centran en aspectos operativos y pueden incluir:

- Prueba de mecanismos de copia de seguridad y recuperación • Instalación, desinstalación y actualización de software • Recuperación de fallas • Gestión de usuarios • Actividades de mantenimiento • Actividades de carga y migración de datos • Comprobación de vulnerabilidades de seguridad • Realización de pruebas de rendimiento

El principal objetivo de las pruebas de aceptación operativa es ganar confianza en que los operadores o administradores del sistema podrán garantizar que los usuarios podrán operar el sistema correctamente en un entorno de producción, incluso en condiciones excepcionales y difíciles.

Pruebas de aceptación del cumplimiento contractual y legal Las pruebas de aceptación del cumplimiento del contrato se realizan de acuerdo con los criterios de aceptación escritos en el contrato para el desarrollo de software contratado. Estos criterios de aceptación deberán especificarse cuando las partes acuerden el contenido del contrato. Este tipo de prueba de aceptación suele ser realizada por usuarios o evaluadores independientes.

Tabla 2.2 Requisitos del estándar DO-178C para cobertura de pruebas por nivel de riesgo

Nivel de criticidad	tipo de falla	Requisitos de cobertura
A	Se requiere cobertura catastrófica completa de MC/DC (condición/decisión modificada)	
B	Peligroso/severo	Se requiere cobertura de decisión completa
C	Importante	Se requiere cobertura de requisitos de bajo nivel; pruebas del manejo y control adecuados de los datos; se requiere cobertura completa del estado de cuenta
D	Menor	Se requiere cobertura de requisitos de alto nivel
mi	Sin efecto	Sin requisitos

Las pruebas de aceptación del cumplimiento legal se realizan en el contexto de la legislación aplicable, como leyes, reglamentos o estándares de seguridad. Este tipo de prueba de aceptación suele ser realizada por usuarios o evaluadores independientes, y los reguladores pueden observar o auditar los resultados. El objetivo principal de las pruebas de aceptación para el cumplimiento contractual y legal es obtener seguridad de que se ha logrado el cumplimiento de los requisitos establecidos en los contratos o regulaciones aplicables. Un buen ejemplo de norma que impone criterios de cobertura específicos es la norma DO-178C aplicable al software de aviación. En este estándar, el software se clasifica por nivel de riesgo (niveles A a E) y luego, para cada clasificación, se definen explícitamente los requisitos para cumplir con criterios de cobertura específicos (consulte la Tabla 2.2).

Criterios de cobertura: La cobertura de MC/DC, decisiones y declaraciones que aparecen en este estándar se refieren a criterios de cobertura de caja blanca específicos. Este último, así como el criterio de cobertura de sucursales, similar a la cobertura de decisiones, se analizan en el programa de estudios del nivel básico (ver Secciones 4.3.1 y 4.3.2). El criterio MC/DC se analiza en el programa de estudios de nivel avanzado: Analista de pruebas técnicas.

Pruebas Alfa y Beta Los

desarrolladores del llamado software comercial disponible (COTS), es decir, software para venta general, a menudo quieren comentarios de clientes potenciales o existentes antes de que el software llegue al mercado. Para ello se utilizan pruebas alfa y beta.

Las pruebas alfa se realizan en las instalaciones de la organización de desarrollo de software, pero en lugar del equipo de desarrollo, las pruebas las realizan clientes y/u operadores actuales o potenciales o evaluadores independientes. Las pruebas beta, por otro lado, las realizan clientes actuales o potenciales en sus propias ubicaciones. Las pruebas beta pueden ir precedidas o no de una prueba alfa.

Uno de los objetivos de las pruebas alfa y beta es generar confianza en los clientes y/u operadores potenciales y existentes en que pueden utilizar el sistema en condiciones normales, en un entorno de producción, para lograr sus objetivos con un esfuerzo, costo y riesgo. Otro propósito puede ser detectar errores relacionados con las condiciones y entornos en los que se utilizará el sistema, especialmente cuando dichas condiciones son difíciles de reproducir para el equipo del proyecto. Si las pruebas beta se realizan en un grupo representativo de usuarios, entonces, debido a que se realizan en el

entornos utilizados por los probadores individuales, las pruebas cubrirán de forma representativa los entornos en los que funcionará el sistema. Por ejemplo, si el 80% de los usuarios tiene Windows 11 y el 20% tiene Windows 10, entonces en un grupo aleatorio y representativo de probadores beta, aproximadamente el 80% de ellos debería estar probando un producto instalado en Windows 11 y aproximadamente el 20% en Windows 10..

Detalles sobre las pruebas de aceptación

Base de prueba

Ejemplos de productos de trabajo que pueden usarse como base de prueba para cualquier tipo de

Las pruebas de aceptación incluyen:

- Procesos de negocio •

Requisitos de usuario o requisitos de negocio • Regulaciones, acuerdos, normas y estándares • Casos de uso • Documentación del sistema o

manuales de usuario • Procedimientos de instalación •

Informes de análisis de riesgos

Además, la base de prueba a partir de la cual se derivan los casos de prueba para las pruebas de aceptación operativa pueden ser los siguientes productos de trabajo:

- Procedimientos de respaldo y restauración •

Procedimientos de recuperación de fallas •

Documentación operativa • Documentación

de implementación e instalación • Supuestos de desempeño • Normas, estándares o regulaciones en el campo de la seguridad

Objetos de prueba

Los objetos de prueba típicos de cualquier tipo de prueba de aceptación incluyen:

- Sistema bajo prueba •

Configuración del sistema y datos de configuración • Procesos de negocio realizados en un sistema totalmente integrado • Sistemas de respaldo y centros de reemplazo de sitios activos (para probar el negocio

mecanismos de continuidad y recuperación de fallas) •

Procesos relacionados con el uso operativo y el mantenimiento •

Formularios

- Informes •

Datos de producción existentes y convertidos

Defectos y fallas típicas.

(continuado)

Ejemplos de defectos comunes detectados mediante diversas formas de pruebas de aceptación incluyen:

- Flujos de trabajo del sistema que son incompatibles con los requisitos comerciales o del usuario • Reglas comerciales implementadas incorrectamente • Incumplimiento del sistema para cumplir con los requisitos contractuales o legales • Fallos no funcionales, como vulnerabilidades de seguridad, rendimiento insuficiente bajo carga pesada o mal funcionamiento en una plataforma compatible

Enfoques y responsabilidades específicos Las pruebas de aceptación a menudo recaen en los clientes, usuarios comerciales, propietarios de productos u operadores de sistemas, pero otras partes interesadas también pueden participar en el proceso. Las pruebas de aceptación a menudo se consideran el último nivel del ciclo secuencial de desarrollo de software, pero también pueden tener lugar en otras etapas, por ejemplo:

- Las pruebas de aceptación del software para la venta general pueden realizarse durante instalación o integración.
- Las pruebas de aceptación de una nueva mejora funcional pueden realizarse antes de que comiencen las pruebas del sistema.

En los modelos SDLC iterativos, los equipos de proyecto pueden utilizar varias formas de pruebas de aceptación al final de cada iteración, como pruebas centradas en la verificación de que la nueva funcionalidad cumple con los criterios de aceptación o pruebas centradas en la validación de la nueva funcionalidad frente a las necesidades del usuario. Además, al final de cada iteración, se pueden realizar pruebas alfa y beta, así como pruebas de aceptación del usuario, pruebas de aceptación de producción y pruebas de aceptación de cumplimiento contractual y legal, después de completar cada iteración o una serie de iteraciones.

Ejemplo Una empresa desarrolla un software basado en web para que la oficina del IRS de Polonia ingrese datos de los formularios de impuestos PIT-11 por parte de un funcionario para generar un formulario de impuestos PIT-37 final para un contribuyente en particular.

Un ejemplo de prueba de componentes sería probar la exactitud de un formulario de entrada de datos, por ejemplo, en términos de cómo se comportarán los campos de moneda cuando se ingrese en ellos un valor de carácter o una cantidad con precisión incorrecta.

Un ejemplo de una prueba de integración de componentes sería la interacción entre los la función de inicio de sesión del sistema y una función que otorga ciertos permisos a un usuario.

Un ejemplo de una prueba del sistema sería que el evaluador ejecutara todo el proceso de ingreso de datos de varios formularios PIT-11 y luego verificará que el sistema genera el PIT-37 correcto.

Un ejemplo de una prueba de integración de sistemas sería probar la exactitud de extraer datos del contribuyente de la base de datos PESEL (número de identificación personal utilizado en Polonia) basándose en el número PESEL del contribuyente ingresado por el usuario.

Un ejemplo de prueba de aceptación (beta) sería la validación del sistema por parte de los funcionarios de la oficina tributaria en la oficina, en sus propias computadoras y en el entorno de producción. La validación puede verificar la corrección sustancial (por ejemplo, el cumplimiento del programa con la ley), pero también cuestiones de usabilidad (por ejemplo, si las interfaces son inteligibles) o qué tan bien se integra el sistema con la infraestructura de TI de la oficina tributaria.

2.2.2 Tipos de prueba

Un tipo de prueba es un grupo de actividades de prueba dinámicas realizadas para probar características específicas de un sistema de software (o parte de él) de acuerdo con objetivos de prueba específicos. Así, a diferencia de los niveles de prueba, el tipo de prueba no está relacionado con fases de un proceso de desarrollo, sino con el objetivo que tenemos en mente al realizar una determinada prueba.

Los objetivos de la prueba pueden incluir:

- Evaluación de características de calidad funcional como integridad, corrección, y adecuación
- Evaluación de características de calidad no funcionales, incluidos parámetros como confiabilidad, rendimiento, seguridad, compatibilidad o usabilidad
- Determinar si la estructura o arquitectura de un componente o sistema es correcto, completo y de acuerdo con las especificaciones

El programa de estudios de Foundation Level distingue cuatro tipos de pruebas básicas:

- Pruebas funcionales •
- Pruebas no funcionales •
- Pruebas de caja blanca •
- Pruebas de caja negra

Sin embargo, cabe señalar que se pueden distinguir muchos más tipos de pruebas. Un ejemplo serían, por ejemplo, las denominadas pruebas de humo, cuya finalidad es verificar la corrección de la funcionalidad básica del sistema, antes de comenzar a realizar pruebas más detalladas.

Ahora analizaremos cada uno de los cuatro tipos de pruebas anteriores.

2.2.2.1 Pruebas funcionales

Las pruebas funcionales de ~~el~~ sistema implican la ejecución de pruebas que evalúan las funciones que debe realizar el sistema. Por lo tanto, las pruebas funcionales prueban "qué" debe hacer el sistema. Los requisitos funcionales se pueden describir en productos de trabajo, como especificaciones de requisitos comerciales, historias de usuarios, casos de uso o especificaciones de requisitos del sistema, pero también existen en forma no documentada.

El principal objetivo de las pruebas funcionales es verificar la corrección funcional, la idoneidad funcional y la integridad funcional. Estos son los tres

Tabla 2.3 Niveles de prueba versus pruebas funcionales

Nivel de prueba	Base de prueba de muestra	prueba de muestra
Componente	Especificaciones del componente	Verificación de la corrección del componente.
		cálculo del importe del impuesto
Integración de componentes	Diagramas de componentes (diseño de arquitectura)	Verificación de la transferencia de datos entre el componente de inicio de sesión y el componente de autorización
Sistema	Historias de usuarios, casos de uso	Verificación de la correcta implementación del proceso de negocio "pagar impuestos"
Sistema	Especificación de la interfaz sistema de integración-base de datos	Verificación de la exactitud del envío de una consulta a la base de datos y la recepción de su resultado.
Aceptación Manual	de usuario	Validar que la descripción de la implementación de la funcionalidad contenida en el manual sea coherente con el proceso de negocio real realizado por el sistema.

subcaracterísticas de funcionalidad definidas en el modelo de calidad ISO/IEC/IEEE 25010 [5].

Las pruebas funcionales son el tipo de prueba más comúnmente asociado con las pruebas, porque es común equiparar las pruebas con verificar que el sistema hace lo que se supone que debe hacer para el usuario. Sin embargo, no se debe olvidar que otros tipos de pruebas, incluidas las que se analizan en las siguientes secciones, son igualmente importantes.

Las pruebas funcionales pueden (y deben) realizarse en todos los niveles de prueba (por ejemplo, las pruebas de componentes pueden basarse en las especificaciones de los componentes), pero con la salvedad de que las pruebas realizadas en cada nivel se centran en cuestiones diferentes (consulte la Sección 2.2.1). La Tabla 2.3 ofrece un ejemplo de actividades realizadas mediante pruebas funcionales en diferentes niveles de prueba para el sistema de ingreso de datos de la oficina de impuestos antes mencionado.

Las pruebas funcionales tienen en cuenta el comportamiento del software, que suele describirse en documentos externos al sistema: especificaciones de requisitos, historias de usuarios, etc. En consecuencia, las pruebas funcionales suelen utilizar técnicas de caja negra para derivar condiciones de prueba y casos de prueba que comprueban la funcionalidad de un componente o sistema (ver Sección 4.2), pero no se limita a estas técnicas.

La minuciosidad de las pruebas funcionales se puede medir mediante la cobertura funcional. El término "cobertura funcional" se refiere al grado en que se ha probado un tipo específico de elemento funcional, expresado como porcentaje de elementos de un tipo determinado cubiertos por la prueba. Al rastrear la relación entre los casos de prueba, sus resultados y los requisitos funcionales, por ejemplo, es posible calcular qué porcentaje de los requisitos han sido cubiertos por las pruebas y, como resultado, identificar cualquier brecha en la cobertura.

Ejemplo La Tabla 2.4 muestra la trazabilidad entre casos de prueba y requisitos funcionales. A cada caso de prueba se le ha asignado su prioridad (relacionada con el riesgo que cubre) en una escala de 1 (menor) a 5 (crítico).

Ahora supongamos que se han realizado los seis casos de prueba. Supongamos también que las pruebas 1, 2 y 4 pasan y las 3, 5 y 6 fallan. Supongamos la siguiente definición de cobertura de requisitos: la cobertura del requisito R es la suma de las prioridades de

Tabla 2.4 Trazabilidad entre requisitos y prueba casos

Prueba \ Requisito	Requisito 1	Requisito 2	Requisito 3
Prueba1 (prioridad 2)	X		
Prueba2 (prioridad 5)	X		X
Prueba3 (prioridad 1)		X	X
Prueba4 (prioridad 1)		X	
Prueba5 (prioridad 4)		X	
Prueba6 (prioridad 2)			X

los casos de prueba asociados con R que pasan, en relación con la suma de las prioridades de todos los casos de prueba asociados con R.

Por ejemplo, asociados con el requisito Req 1 están los casos de prueba 1 y 2 con un suma de prioridad de $2 + 5 = 7$. Dado que se pasan ambas pruebas, la cobertura del requisito El requisito 1 es $7/7 = 100\%$. De manera similar, podemos calcular la cobertura de requisitos para todos los requisitos:

$$\text{Requisito 1: cobertura} = \frac{\text{# P 2 p 5}}{\text{# P 27}} = \frac{2}{7} = 100\%$$

$$\text{Requisito 2: cobertura} = \frac{\text{# P 1}}{\text{# P 1}} = \frac{1}{1} = 100\%$$

$$\text{Requisito 3: cobertura} = \frac{\text{# P 5}}{\text{# P 5}} = \frac{5}{5} = 100\%$$

Por supuesto, ¿cómo será exactamente el resultado de la cobertura de la prueba de requisitos? depende de la definición de cobertura adoptada. También podríamos, por ejemplo, definirlo como la proporción de pruebas aprobadas (relacionadas con el requisito W) con respecto a todas las pruebas relacionadas con requisito W. Entonces la cobertura de los requisitos Req 1, Req 2 y Req 3 sería ser, respectivamente, $2/2 = 100\%$, $1/3 = 33,3\%$ y $1/3 = 33,3\%$.

Es posible que se necesiten habilidades o experiencia especiales para diseñar y ejecutar pruebas funcionales. como el conocimiento del problema empresarial específico que resuelve el software (por ejemplo, software de modelado geológico para la industria del petróleo y el gas) o el rol específico realiza el software (por ejemplo, un juego de computadora que proporciona entretenimiento interactivo). Esto se debe a que las pruebas funcionales se ocupan de las funciones que el proporciona el sistema, y éstos suelen estar integrados en un contexto empresarial específico. Por lo tanto, por ejemplo, los evaluadores de aplicaciones financieras deberían al menos conocer el conceptos básicos de las finanzas.

2.2.2.2 Pruebas no funcionales

El propósito de las pruebas no funcionales es evaluar las características de sistemas y software, como usabilidad, rendimiento y seguridad. una clasificación de las características de calidad del software se proporciona en el estándar ISO/IEC 25010 (Evaluación y requisitos de calidad de sistemas y software (SQuaRE) [5]).

Las pruebas no funcionales verifican "cómo" se comporta un sistema.

La Figura 2.16 muestra el modelo de calidad según ISO/IEC 25010. Distingue ocho características de calidad (rectángulos grises) y asigna a cada una de ellas

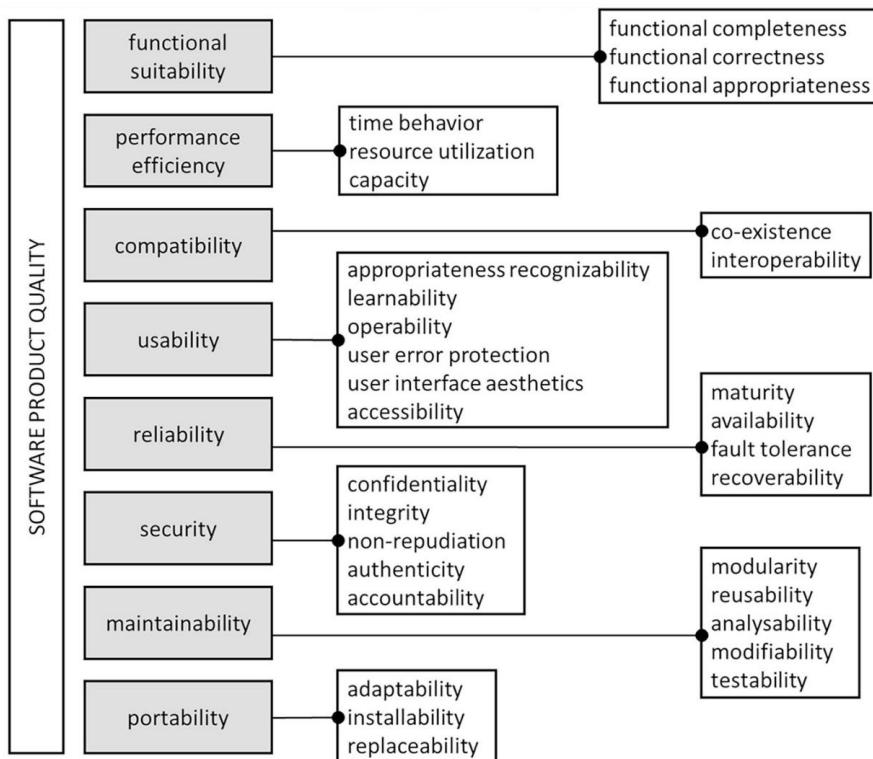


Fig. 2.16 ISO/IEC 25010—Modelos de calidad de sistemas y software (SQuaRE)

subcaracterísticas correspondientes. Desde el punto de vista del programa de estudios del nivel básico, las características no funcionales son prácticamente idoneidad funcional. Los modelos de calidad, como ISO/IEC 25010, pueden ser útiles para determinar qué parámetros no funcionales de nuestro sistema deben probarse.

Contrariamente a la idea errónea común, las pruebas no funcionales pueden, y a menudo deben, realizarse en todos los niveles de prueba. Además, debe hacerse en la fase más temprana posible, porque detectar defectos no funcionales demasiado tarde puede suponer una gran amenaza para el éxito del proyecto. Muchas pruebas no funcionales se derivan de pruebas funcionales, ya que utilizan las mismas pruebas funcionales, pero verifican que mientras se realiza la función, se cumple una restricción no funcional (por ejemplo, verificar que una función se realiza dentro de un tiempo específico o que una función se puede trasladar a un nuevo plataforma).

La Tabla 2.5 muestra ejemplos de pruebas no funcionales del sistema antes mencionado.

para la entrada de datos por parte de las oficinas tributarias, que se puede realizar en cada nivel de prueba.

Se pueden utilizar técnicas de prueba de caja negra para derivar condiciones de prueba y casos de prueba para pruebas no funcionales (ver Sección 4.2). Un ejemplo es el uso del análisis de valores límite para definir las condiciones de tensión para las pruebas de rendimiento.

La diligencia de las pruebas no funcionales se puede medir mediante la cobertura no funcional. El término “cobertura no funcional” significa el grado en que un tipo específico de

Tabla 2.5 Niveles de prueba versus pruebas no funcionales

Nivel de prueba	prueba de muestra
Componente	Verifique que el componente pueda ser reemplazado fácilmente por otro con equivalente funcionalidad (portabilidad—reemplazabilidad)
Sistema integración	Verifique que la comunicación entre el sistema cliente y el servidor central están debidamente protegidos y no se ven afectados por posibles ataques de piratería (seguridad—confidencialidad)
Sistema	Verifique que el sistema genere un informe resumido lo suficientemente rápido según 100.000 formularios de impuestos (eficiencia en el desempeño, comportamiento en el tiempo)
Aceptación	Validar que el sistema se adapta a las necesidades de los usuarios con discapacidad visual (usabilidad—accesibilidad)

Se ha probado un elemento no funcional expresado como porcentaje de elementos de un tipo determinado, cubiertos por pruebas. Al rastrear la relación entre las pruebas y los tipos de dispositivos apoyado por una aplicación móvil, por ejemplo, es posible calcular qué porcentaje de dispositivos fueron cubiertos por pruebas de compatibilidad y, en consecuencia, identificar cualquier brecha en la cobertura.

Objetivos SMART para pruebas no funcionales

Los resultados de las pruebas no funcionales deben ser precisos, es decir, deben poder expresarse en términos de algunas métricas bien definidas. A menudo, el acrónimo SMART es:

Se utiliza para especificar las características de requisitos y pruebas adecuadamente definidos: específicos, mensurables, alcanzables, realistas y con plazos determinados. Esto es porque Durante la ejecución de la prueba, necesitamos hacer una comparación entre el resultado real y el resultado esperado y declarar inequívocamente si la prueba fue aprobada o fallido. En la Tabla 2.6, damos ejemplos de métricas para varias características de calidad según el modelo ISO/IEC 25010.

Habilidades o conocimientos especiales, como el conocimiento de vulnerabilidades específicas de un proyecto o tecnología en particular (por ejemplo, vulnerabilidades de seguridad asociadas con ciertos lenguajes de programación) o un grupo particular de usuarios (por ejemplo, usuario perfiles para sistemas de gestión sanitaria): pueden ser necesarios para diseñar y ejecutar pruebas no funcionales. Por eso es una práctica común, por ejemplo, contratar organizaciones de terceros que se especialicen en realizar pruebas de un tipo específico, como pruebas de seguridad o rendimiento.

Para obtener información detallada sobre pruebas no funcionales de características de calidad, consulte el programa de estudios Analista de pruebas [28], Analista de pruebas técnicas [29], Probador de seguridad [30] y otros programas de estudios especializados de ISTQB®.

Tabla 2.6 Ejemplos de métricas para características no funcionales

Métricas de características		Modelo	Descripción
Fiabilidad	Densidad de fallas por número de pruebas	$M = A/B$	A = número de fallos detectados B = número de pruebas ejecutadas
	Disponibilidad $M = A/(A + B)$	$A = \text{tiempo medio hasta el fallo (MTTF)}$ $B = \text{tiempo medio de reparación (MTTR)}$	
Usabilidad	Facilidad de aprendizaje la función	$METRO = T$	T = tiempo de aprendizaje
	Disponibilidad de ayuda	$M = A/B$	A = el número de tareas para las cuales hay es un tema de ayuda válido B = número total de tareas probadas
	Atractivo de interacción	cuestionario	Métricas medidas por cuestionario después usando software
Actuación	Tiempo de respuesta $M = T_1 - T_2$	$T_1 = \text{tiempo de finalización de la tarea}$ $T_2 = \text{tiempo de finalización para emitir una tarea pedido}$	
	Rendimiento $M = A$		A = número de tareas completadas en un unidad fija de tiempo
Análisis de fallas de mantenibilidad	capacidad	$M = 1 - A/B$	A = número de fallas, la causa de que aun se desconoce B = número de todas las fallas observadas
Portabilidad	Posibilidad de interoperabilidad	$M = A/T$	A = número de problemas inesperados o fallos durante el uso paralelo de otros software T = tiempo total durante el cual se utilizó otro software en paralelo

2.2.2.3 Prueba de caja blanca

En el caso de las pruebas de caja blanca, las pruebas se derivan de la base de datos internos. estructura o implementación de un sistema determinado. La estructura interna puede incluir código, arquitectura, flujos de trabajo y/o flujos de datos dentro del sistema (ver Sección 4.3, donde describimos en detalle las técnicas y criterios de cobertura de caja blanca aplicables al examen Foundation Level).

La minuciosidad de las pruebas de caja blanca se puede medir mediante la cobertura estructural. El término “cobertura estructural” significa el grado en que un tipo específico de elemento estructural ha sido probado expresado como el porcentaje de elementos de un determinado tipo cubiertos por las pruebas. La métrica de cobertura general se expresa así por la fórmula:

$$\text{cobertura} = A=B,$$

donde A = número de elementos estructurales cubiertos por las pruebas y B = número de todos elementos estructurales identificados. Un ejemplo es la métrica de cobertura de declaraciones, que es

Tabla 2.7 Pruebas de caja blanca en diferentes niveles de prueba

Nivel de prueba	Ejemplo de estructura a cubrir Código
Componente	fuente (p. ej., declaraciones, ramas, decisiones, rutas)
Integración de componentes	Gráfico de llamadas (es decir, eventos de llamada de una función por otra), un conjunto de funciones API
Sistema	modelo de proceso de negocio
Integración de sistema	Gráfico de llamadas a nivel de servicios ofrecidos por los sistemas comunicantes.
Aceptación	Modelo de árbol de la estructura del menú.

el número de declaraciones ejecutables cubiertas por las pruebas dividido por todas las declaraciones ejecutables en un código determinado.

La métrica de cobertura definida anteriormente toma valores entre 0 y 1 (porque tanto A como B son positivos y A siempre es menor o igual a B). La forma más común de presentación de esta métrica es su forma porcentual:

$$\text{cobertura} = \frac{\text{A}}{\text{B}} \times 100\%:$$

Por ejemplo, si cubrimos 5 de 20 declaraciones de código ejecutable con nuestras pruebas, entonces logramos la cobertura $5/20 = 0,25$ o, en otras palabras, $(5/20) * 100\% = 25\%$.

En el nivel de prueba de componentes, la cobertura del código está determinada por el porcentaje del código de un componente que ha sido probado. Este valor se puede medir en términos de varios aspectos del código (elementos de cobertura), como el porcentaje de declaraciones ejecutables o el porcentaje de resultados de decisiones probados en un módulo determinado. Los tipos de cobertura anteriores se denominan colectivamente cobertura de código. En el nivel de prueba de integración de componentes, las pruebas de caja blanca se pueden realizar sobre la base de la arquitectura del sistema (por ejemplo, interfaces entre módulos) y la cobertura estructural se puede medir en términos del porcentaje de interfaces probadas. La Tabla 2.7 muestra ejemplos de estructuras que se pueden utilizar para realizar pruebas de caja blanca en diferentes niveles de prueba.

Es posible que se necesiten habilidades o conocimientos especiales para diseñar y ejecutar pruebas de caja blanca, como conocimiento de la construcción de código (que permite, por ejemplo, el uso de herramientas para medir la cobertura del código), conocimiento de cómo almacenar datos (que permite, por ejemplo, la evaluación de consultas a bases de datos), o cómo utilizar herramientas para medir la cobertura e interpretar correctamente los resultados que generan.

Es fundamental resaltar que lograr un nivel de cobertura arbitrariamente alto no garantiza una calidad arbitrariamente alta. Incluso una cobertura del 100% de todas las declaraciones ejecutables no garantiza que cada declaración ejecutable realmente funcione como se espera. Es un medio para medir la integridad de la prueba, en lugar de la calidad del sistema bajo prueba.

2.2.2.4 Pruebas de caja negra

Las pruebas de caja negra ~~se basan~~³ en especificaciones (es decir, información externa al objeto de prueba). El objetivo principal de las pruebas de caja negra es verificar que el comportamiento del sistema se ajuste al comportamiento descrito en la especificación.

Para obtener más información sobre las pruebas de caja negra, consulte la Sección [4.2](#), donde analizamos en detalle las técnicas de prueba de caja negra.

2.2.2.5 Niveles de prueba versus tipos de prueba

Los niveles y tipos de pruebas son independientes entre sí. Aunque en la práctica será cierto que ciertos tipos de pruebas tienden a ocurrir en ciertos niveles de prueba específicos, en general, cualquier tipo de prueba se puede ejecutar en cualquier nivel de prueba. Esto se muestra en el siguiente ejemplo.

Ejemplo Estamos probando una aplicación web de una tienda electrónica de venta de libros. Los usuarios registrados, después de iniciar sesión, tienen acceso a un catálogo con función de búsqueda (que incluye búsqueda por título, autor, categoría de libro, rango de precios). Pueden agregar artículos seleccionados al carrito de compras y luego proceder al pago. A continuación se describen algunos ejemplos de pruebas funcionales, no funcionales, de caja negra y de caja blanca (en aras de la simplicidad, solo damos descripciones breves de las pruebas, no describimos los casos de prueba en detalle).

Ejemplos de pruebas funcionales (y de caja negra):

- Validar el registro del usuario en el sistema. • Verificar si es posible registrar un usuario con un nombre existente en el sistema. • Compruebe si es posible registrar un usuario con una dirección de correo electrónico existente en el sistema.
- Consultar compra estándar y pago correcto. • Comprobar si al poner un libro en el carrito de compras, esa copia queda bloqueada en el sistema y otros usuarios no pueden agregarlo a sus carritos. • Comprobar que transcurrido un tiempo determinado y fijo, si no se realiza el pago, la sesión caduca y todos los libros del carrito vuelven a la tienda y quedan disponibles para el resto de usuarios.

Ejemplos de pruebas no funcionales:

- Consultar el tiempo de respuesta tras consultar un catálogo de 1 millón de artículos. • Prueba de vulnerabilidad de inyección SQL³ mediante el cuadro de búsqueda. • Comprobar el tiempo de respuesta del sistema en la situación de ejecución simultánea de una consulta realizada por mil usuarios.

³ La inyección SQL es un ataque a un sitio web o aplicación web en el que se agrega código de lenguaje SQL a un campo en un formulario para obtener acceso a una cuenta o cambiar datos.

Ejemplos de pruebas de caja blanca:

- Verificar la cobertura del extracto del componente de registro de clientes (intención: verificar todas las rutas y posibles excepciones y errores que debe manejar el sistema).

- Comprobar el funcionamiento de todos los elementos del menú principal (abarcando la estructura de la página web). • Probar todas

las funciones de la interfaz relacionadas con el sistema de pago electrónico.

Cada uno de los tipos de pruebas enumerados anteriormente se puede realizar en cualquier nivel de prueba.

Para ilustrar esto, los siguientes son ejemplos de pruebas funcionales, no funcionales, de caja negra y de caja blanca que se realizan en todos los niveles de prueba para una aplicación bancaria. El primer grupo son las pruebas funcionales (y de caja negra):

- Para las pruebas de componentes, las pruebas están diseñadas para reflejar cómo un módulo debe calcular el interés compuesto. • Para las pruebas de integración de componentes, las pruebas están diseñadas para reflejar cómo la información de la cuenta capturada en la interfaz de usuario se transfiere a la capa de lógica empresarial. • Para las pruebas del sistema, se están diseñando pruebas que reflejen cómo los titulares de cuentas pueden solicitar una línea de crédito para sobregiros. • Para las pruebas de integración de sistemas, las pruebas están diseñadas para reflejar cómo un sistema determinado verifica la solvencia de un titular de cuenta utilizando un microservicio externo. • Para las pruebas de aceptación, las pruebas están diseñadas para reflejar cómo un empleado del banco aprueba o rechaza una solicitud de préstamo.

El siguiente grupo contiene ejemplos de pruebas no funcionales:

- Para las pruebas de componentes, se diseñan pruebas de rendimiento que evalúan el número de ciclos de CPU necesarios para realizar cálculos complejos de la cantidad total de interés.
- Para las pruebas de integración de componentes, las pruebas de seguridad están diseñadas para detectar vulnerabilidades de seguridad relacionadas con desbordamientos del búfer de datos pasados desde la interfaz de usuario a la capa de lógica empresarial. • Para las pruebas del sistema, las pruebas de portabilidad están diseñadas para verificar que la presentación La capa funciona en todos los navegadores y dispositivos móviles compatibles.
- Para las pruebas de integración del sistema, las pruebas de confiabilidad están diseñadas para evaluar la resiliencia del sistema en caso de una falta de respuesta del microservicio utilizado para verificar la solvencia. • Para las pruebas de aceptación, se diseñan pruebas de usabilidad para evaluar las características de accesibilidad para personas con discapacidad, aplicadas a la interfaz de procesamiento de préstamos por parte del banco.

El siguiente grupo contiene ejemplos de pruebas de caja blanca:

- Para las pruebas de componentes, las pruebas están diseñadas con el objetivo de garantizar una cobertura completa de las instrucciones y decisiones del código en todos los módulos que realizan cálculos financieros.

- Para las pruebas de integración de componentes, las pruebas están diseñadas para ver cómo cada pantalla de la interfaz de visualización del navegador pasa datos a la siguiente pantalla y a la capa de lógica empresarial.
- Para las pruebas del sistema, las pruebas están diseñadas para garantizar la cobertura de posibles secuencias de páginas web que se muestran al solicitar una línea de crédito.
- Para las pruebas de integración de sistemas, se diseñan pruebas que verifican todos los tipos posibles de consultas enviadas al microservicio utilizado para la verificación de crédito.
- Para las pruebas de aceptación, las pruebas están diseñadas para cubrir todas las estructuras soportadas y rangos de valores para archivos de datos financieros utilizados en transferencias interbancarias.

Esta sección proporciona ejemplos de todos los tipos de pruebas realizadas en todos los niveles, pero no todo el software necesita incluir todos los tipos de pruebas en todos los niveles. Lo importante es que se ejecuten las pruebas adecuadas en cada nivel; esto es especialmente cierto para el nivel más temprano en el que se produce un tipo de prueba.

2.2.3 Pruebas de confirmación y pruebas de regresión

Una vez que se han realizado cambios en el sistema para corregir defectos o agregar o modificar funcionalidades, se deben realizar pruebas para confirmar que los cambios realmente solucionaron el defecto o implementaron correctamente la funcionalidad relevante, sin causar consecuencias adversas imprevistas. El programa de estudios de Foundation Level distingue entre dos tipos de pruebas de este tipo: pruebas de confirmación (también llamadas reevaluaciones) y pruebas de regresión.

Pruebas de confirmación

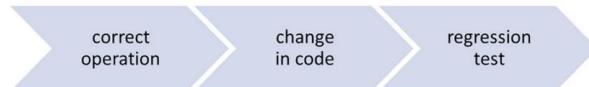
Las pruebas de confirmación (también conocidas como reprobadas) se realizan en relación con la ocurrencia de una falla y la reparación del defecto asociado. La función de las pruebas de confirmación es verificar que el defecto efectivamente ha sido reparado (ver Fig. 2.17). Normalmente, una nueva prueba es la misma prueba que reveló la falla, realizada nuevamente, después de que se haya reparado la falla, pero a veces la prueba de confirmación puede ser una prueba diferente, por ejemplo, si el defecto informado fue una falta de funcionalidad. El mínimo absoluto al realizar pruebas de confirmación es ejecutar los pasos que causaron la falla anteriormente. Si se supera la prueba, podemos asumir que el defecto se ha solucionado.

Como nunca se sabe cuándo ocurrirá una falla y cuándo se reparará, la ejecución de nuevas pruebas no se puede planificar con anticipación. Sin embargo, se debe asignar tiempo suficiente para las pruebas de confirmación durante la planificación (ver Capítulo 5).

Fig. 2.17 Prueba de confirmación



Fig. 2.18 Prueba de regresión



Pruebas de regresión Las pruebas de regresión  verifican si un cambio (arreglo o modificación) realizado en un componente del código afectará inadvertidamente el comportamiento de otras partes del código en el mismo componente, en otros componentes del mismo sistema o incluso en otros sistemas (ver figura 2.18). Además, se deben tener en cuenta los cambios en el entorno, como la introducción de una nueva versión del sistema operativo o del sistema de gestión de bases de datos. Los efectos secundarios no deseados mencionados anteriormente se denominan regresiones (de la palabra "regresión", que significa deterioro de algo), y las pruebas de regresión implican volver a ejecutar pruebas para detectar estas regresiones.

Dado que las pruebas de regresión se realizan con mayor frecuencia cuando se agrega otra funcionalidad o al final de cada iteración, estas pruebas generalmente, a diferencia de las pruebas de confirmación, se pueden programar perfectamente. Dado que las pruebas de regresión deben realizarse con frecuencia y además evolucionan con bastante lentitud, son candidatas naturales para la automatización. La automatización de las pruebas de regresión ahorra tiempo y esfuerzo, ya que los evaluadores pueden dedicar el tiempo ahorrado a otras actividades, como crear y ejecutar nuevas pruebas. La automatización de este tipo de pruebas debería comenzar temprano en el proyecto.

Un problema práctico común con las pruebas de regresión es que su número crece muy rápidamente. Después de cierto punto, tardan bastante en ejecutarse durante la noche y los resultados esperan a los evaluadores a la mañana siguiente. A veces, sin embargo, un conjunto de pruebas de regresión ni siquiera se puede ejecutar de la noche a la mañana. En este caso, se debe tomar una decisión para optimizar la ejecución de la prueba de regresión. Las posibles soluciones incluyen:

- Priorización: realizar primero las pruebas más importantes y relevantes; aquellos que no se ejecutan son menos importantes, por lo que hay poco riesgo de que un fallo importante pase desapercibido. • Reducción: elimina algunas pruebas del conjunto de regresión; El criterio para la eliminación puede basarse, por ejemplo, en la capacidad de la prueba para detectar fallas: si una prueba detecta fallas con bastante frecuencia, definitivamente debe permanecer, mientras que si una prueba nunca ha detectado ninguna falla, puede eliminarse del conjunto de pruebas. pruebas de regresión después de algún tiempo.
- Estrategia mixta: las pruebas más importantes se ejecutan de forma programada, mientras que la parte que no se puede realizar debido a consideraciones de tiempo se ejecuta con menos frecuencia, como cada dos iteraciones.

Las pruebas de confirmación y las pruebas de regresión se pueden realizar en todos los niveles de prueba, especialmente en modelos SDLC iterativos e incrementales; las nuevas funciones, los cambios en las funciones existentes y la refactorización dan como resultado frecuentes cambios de código que implican pruebas adecuadas. Dada la continua evolución del sistema, las pruebas de confirmación y las pruebas de regresión son muy importantes, especialmente para los sistemas relacionados con IoT, donde los objetos individuales (por ejemplo, dispositivos) se actualizan o reemplazan con frecuencia.

Ejemplo Al probar una tienda electrónica de libros (consulte la sección anterior), un evaluador provocó una falla: el sistema permitió que un usuario se registrara con la misma dirección de correo electrónico que ya tiene otro usuario previamente registrado. Llamemos "prueba A" a la prueba que detectó este problema. El evaluador presentó un informe de defecto y se encargó de otras actividades. Específicamente, realizó la prueba B (intentar pagar un pedido estándar), que pasó. En la siguiente iteración, se implementó un nuevo componente para informar las operaciones de pago.

Después de estos cambios, el evaluador volvió a realizar la prueba B para ver si el componente afectaba la funcionalidad de pago en sí. Momentos después, el evaluador recibió un mensaje de que el defecto que había informado anteriormente se había solucionado. El evaluador volvió a ejecutar la prueba A, que esta vez pasó. El probador cerró el defecto.

En el ejemplo anterior, la prueba A y la prueba B se ejecutaron dos veces, en orden: A, B, B, A.

- La primera ejecución de la prueba A y la primera ejecución de la prueba B no se aplican ni a las pruebas de regresión ni a las pruebas de confirmación, ya que estas pruebas se realizaron por primera vez.
- La segunda ejecución de la prueba B es un ejemplo de prueba de regresión, ya que la intención del evaluador es comprobar si la implementación de un nuevo componente ha roto algo en el módulo de pago.
- La segunda ejecución de la prueba A es un ejemplo de prueba de confirmación, ya que la intención del evaluador es verificar que la prueba A, que causó una falla anteriormente, pasará ahora, después de la corrección.

Las pruebas relacionadas con cambios se pueden realizar en todos los niveles de prueba. Refiriéndose al ejemplo de aplicación bancaria de la Sección [2.2.2.4](#), podemos distinguir los siguientes ejemplos de pruebas relacionadas con cambios:

- Para las pruebas de componentes, las pruebas de regresión automatizadas se diseñan por componente (estas pruebas luego se incluirán en el marco de integración continua).
 - Para las pruebas de integración de componentes, las pruebas están diseñadas para confirmar la efectividad de las correcciones relacionadas con defectos en las interfaces a medida que dichas correcciones se colocan en el repositorio de código.
 - Para las pruebas del sistema, todas las pruebas para un flujo de trabajo determinado se realizan nuevamente si alguna de las pantallas cubiertas por ese flujo de trabajo ha cambiado.
- Para las pruebas de integración del sistema, se realizan diariamente pruebas de la aplicación que funciona con el microservicio para verificación de crédito (como parte de la implementación continua de este microservicio).
- Para las pruebas de aceptación, todas las pruebas fallidas anteriormente se realizan nuevamente después de la corrección.
- Se soluciona el defecto detectado en las pruebas de aceptación.

2.3 Pruebas de mantenimiento

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

El momento en que se lanza un sistema a un cliente no es el final, sino sólo un estado intermedio del desarrollo del producto. Aunque muchos administradores de TI no se dan cuenta, muchos estudios empíricos de ingeniería de software muestran que el costo de mantenimiento del sistema suele ser significativamente mayor que el costo de desarrollarlo. Esto se debe a un hecho simple: normalmente lleva mucho más tiempo utilizar un software que desarrollarlo, y durante su uso pueden ocurrir muchas situaciones diferentes que requieren cambios o correcciones en el software. Ejemplos de tales situaciones son:

- La necesidad de un parche para el sistema debido al descubrimiento de un ataque relacionado con la vulnerabilidad de la seguridad
- Agregar, eliminar o modificar características del programa
- Reparar el defecto que causó la falla reportada por el usuario
- La necesidad de archivar datos debido al desmantelamiento del software
- La necesidad de mover el sistema a un nuevo entorno, causado, por ejemplo, por actualizar la versión del sistema operativo

Una vez implementado en un entorno de producción, el software o sistema requiere mayor mantenimiento. Varios tipos de cambios, como los descritos anteriormente, son prácticamente inevitables. Además, el mantenimiento es necesario para mantener o mejorar las características de calidad no funcionales requeridas del software o sistema a lo largo de su ciclo de vida, especialmente en términos de rendimiento, compatibilidad, confiabilidad, seguridad y portabilidad.

Es bueno recordar que las pruebas de mantenimiento se realizan después del lanzamiento del software, durante su uso operativo. Las pruebas realizadas antes de que el software se entregue al cliente no se pueden considerar pruebas de mantenimiento. El mantenimiento, por definición, se aplica a un producto que ya está en uso.

Después de realizar cualquier cambio en la fase de mantenimiento, se deben realizar pruebas de mantenimiento. El propósito de las pruebas de mantenimiento es tanto verificar que el cambio se ha implementado exitosamente como detectar posibles efectos secundarios (p. ej., regresiones) en las partes sin cambios del sistema (es decir, generalmente en la mayoría de las áreas del sistema). Por lo tanto, las pruebas de mantenimiento incluyen tanto las partes del sistema que han sido cambiadas como las partes no modificadas que pueden haber sido afectadas por los cambios. El mantenimiento se puede realizar tanto de forma programada (en relación con las nuevas versiones) como de forma no programada (en relación con las correcciones urgentes).

A la luz de las consideraciones anteriores, está claro que el tipo más común de prueba de mantenimiento será la prueba de regresión. Sin embargo, este no es el único tipo de prueba de mantenimiento. Por ejemplo, si se va a reemplazar el software por otro, se deben realizar pruebas de archivado o de migración de datos. Estas pueden ser pruebas completamente nuevas que realizaremos solo por primera vez en toda la historia, tal vez de años, de uso de la aplicación.

Una versión de mantenimiento puede requerir la ejecución de pruebas de mantenimiento en múltiples niveles de prueba y utilizando diferentes tipos de pruebas, según el alcance de los cambios realizados.

El alcance de las pruebas de mantenimiento incluye:

- El nivel de riesgo asociado con el cambio (por ejemplo, la medida en que el cambio el área de software se comunica con otros componentes o sistemas) • El tamaño del sistema existente • El tamaño del cambio realizado

Hay varias razones para realizar el mantenimiento del software y, por lo tanto, realizar pruebas de mantenimiento. Esto incluye tanto cambios planificados como no planificados. Los eventos que desencadenan el mantenimiento se pueden dividir en las siguientes categorías:

- **Modificación.** Esta categoría incluye, entre otras, mejoras planificadas (p. ej., en forma de nuevas versiones de software), cambios correctivos y de emergencia, cambios en el entorno operativo (p. ej., actualizaciones planificadas del sistema operativo o de la base de datos), actualizaciones de software para Software COTS y parches que solucionan defectos y vulnerabilidades de seguridad.
 - **Actualización o migración.** Esta categoría incluye, entre otras, la transición de una plataforma a otra, lo que puede implicar pruebas del nuevo entorno y el software modificado o pruebas de conversión de datos (al migrar datos de otra aplicación a un sistema mantenido). . • **Jubilación.** Esta categoría se refiere a la situación en la que una aplicación se retira del uso. Esto puede requerir probar la migración
- o el archivado de datos si es necesario almacenarlos durante un período prolongado. También puede ser necesario probar los procedimientos de recuperación después de archivar durante un período prolongado.

Además, es posible que sea necesario incluir pruebas de regresión para garantizar que las funciones que permanecen en uso sigan funcionando correctamente.

En el caso de sistemas relacionados con IoT, pueden ser necesarias pruebas de mantenimiento tras la introducción de elementos completamente nuevos o modificados en el sistema, como dispositivos de hardware o servicios de software. Durante las pruebas de mantenimiento de dichos sistemas, se pone especial énfasis en las pruebas de integración en varios niveles (por ejemplo, en los niveles de red y de aplicación) y en los aspectos de seguridad, especialmente con respecto a los datos personales.

Ejemplo Una empresa produce un sistema CRM (Customer Relationship Management). Este sistema se basa en una base de datos relacional, que no fue muy bien diseñada y contiene muchos datos redundantes. La aplicación hace referencia a la base de datos a través de un conjunto de funciones bien definidas, en lugar de directamente mediante la construcción de consultas.

El arquitecto decidió que se debía mejorar la estructura de la base de datos: el esquema de la base de datos debía transformarse a la llamada tercera forma normal. El cambio requerirá instalación en cada terminal de usuario.

Este es un evento desencadenante de mantenimiento, relacionado con modificación (cambio correctivo) y, en cierto sentido, migración (ya que se modificará la estructura de la base de datos). Por lo tanto, deberás realizar las siguientes pruebas de mantenimiento relacionadas con la instalación del parche (antes de su implementación):

- Pruebas de regresión directamente relacionadas con el uso de funciones que se comunican con el base de datos
- Pruebas de migración y recuperación de datos y retorno a la antigua estructura de la base de datos en caso de que por alguna razón la nueva estructura de la base de datos en el ordenador del cliente no funcione como debería.

Análisis de impacto

El análisis de impacto nos permite evaluar los cambios realizados en la versión mantenida en términos tanto de los efectos previstos como de los efectos secundarios esperados o potenciales y nos permite identificar áreas del sistema que se verán afectadas por los cambios. Además, puede ayudar a identificar el impacto del cambio en las pruebas existentes. Los efectos secundarios del cambio y las áreas del sistema que pueden verse afectadas por él deben probarse para detectar regresión, una actividad que puede estar precedida por una actualización de las pruebas existentes afectadas por el cambio.

Se puede realizar un análisis de impacto antes de realizar un cambio para determinar si el cambio realmente debería realizarse (dadas las posibles consecuencias para otras áreas del sistema).

Realizar un análisis de impacto puede resultar difícil si:

- Las especificaciones (p. ej., requisitos comerciales, historias de usuarios, arquitectura) están desactualizadas o no están disponibles
- Los casos de prueba no se han documentado o están desactualizados
- La trazabilidad bidireccional entre las pruebas y la base de la prueba no se ha establecido •

El soporte de herramientas es inexistente o inadecuado

- Las personas involucradas no tienen conocimiento del campo y/o del sistema en cuestión
- Se ha prestado muy poca atención a las características de calidad del sistema en términos de mantenibilidad durante el desarrollo del software.

Las dificultades para realizar análisis de impacto son bastante comunes. Esto se debe a que el mantenimiento suele realizarse meses o años después del lanzamiento del software y, a menudo, las personas que crearon la versión original del sistema ya no están en el equipo u organización y la documentación está desactualizada y obsoleta.

Ejemplo. Es necesario implementar un parche de software, relacionado con un cambio en la ley sobre cálculo de impuestos. El cambio afecta al componente X. Utilizando la trazabilidad bidireccional, resulta que el módulo X está asociado a los riesgos R1, R3, R6 y R8, todos los cuales tienen un nivel de riesgo muy alto (es decir, son críticos).

(continuado)

Los riesgos restantes (R2, R4, R5 y R7) tienen un nivel bajo (es decir, son insignificantes). Además, el componente X se comunica con otros cuatro componentes entre los ocho módulos del sistema. Este simple análisis de riesgo muestra que el cambio tiene un alto riesgo y su impacto en el sistema es significativo.

Preguntas de muestra

Pregunta 2.1

(FL-2.1.1, K2)

Eres tester en un proyecto para desarrollar software de piloto automático de aviones. El proyecto se lleva a cabo según el modelo V.

¿Cuál de las siguientes oraciones describe MEJOR las consecuencias asociadas con elegir este modelo?

- R. El énfasis en las pruebas estará en técnicas de prueba basadas en la experiencia.
- B. El equipo de prueba no realizará pruebas estáticas.
- C. Al finalizar la primera iteración, se creará un prototipo funcional del sistema disponible.
- D. En las fases iniciales, los evaluadores participan en revisiones de requisitos, análisis de pruebas y diseño de prueba.

Elija una respuesta.

Pregunta 2.2

(FL-2.1.2, K1)

Una buena práctica de prueba dice que para cada actividad de desarrollo, debe haber una actividad de prueba asociada. ¿Qué modelo SDLC muestra este principio explícitamente?

- A. Modelo iterativo.
- B. Modelo espiral de Boehm.
- C. Melé.
- D. Modelo V.

Elija una respuesta.

Pregunta 2.3

(FL-2.1.3, K1)

¿Qué enfoque de prueba primero utiliza criterios de aceptación para historias de usuarios como base para derivar casos de prueba?

- R.TDD.
- B. ATDD.
- C. FDD.
- D. BDD.

Elija una respuesta.

Pregunta 2.4

(FL-2.1.4, K2)

¿Cuál de las siguientes afirmaciones describe cómo el enfoque DevOps respalda las PRUEBAS?

- R. DevOps permite acortar el ciclo de lanzamiento del software mediante el uso de una generación automatizada de datos de prueba para pruebas de componentes.
- B. DevOps permite comentarios rápidos al desarrollador sobre la calidad del código que comprometerse con el repositorio.
- C. DevOps permite la generación automática de casos de prueba para el nuevo código enviado por un desarrollador al repositorio.
- D. DevOps permite reducir el tiempo necesario para la planificación de lanzamientos y la planificación de iteraciones.

Elija una respuesta.

Pregunta 2.5

(FL-2.1.5, K2)

¿Cuál de los siguientes es un ejemplo del uso del enfoque de desplazamiento a la izquierda?

- A. Aplicación del desarrollo impulsado por pruebas de aceptación (ATDD).
- B. Basar las pruebas en pruebas exploratorias.
- C. Creación de prototipos de GUI durante la fase de obtención de requisitos.
- D. Monitoreo continuo de la calidad del producto después de su entrega al cliente.

Elija una respuesta.

Pregunta 2.6

(FL-2.1.6, K2)

¿Cuál de las siguientes oraciones describe mejor las actividades de un evaluador que asiste a una reunión retrospectiva?

- R. El evaluador solo plantea problemas relacionados con la prueba. Todos los demás temas serán planteados por cualquiera de los demás participantes.
- B. El evaluador es un observador, asegurándose de que la reunión siga los principios de retrospectivas y los valores del enfoque ágil.
- C. El evaluador proporciona retroalimentación y otra información sobre todas las actividades realizadas por el equipo durante la iteración completa.
- D. El evaluador recopila la información proporcionada por otros participantes de la reunión para diseñar pruebas para la siguiente iteración basada en esta información.

Elija una respuesta.

Pregunta 2.7

(FL-2.2.1, K2)

Estás probando el sistema de piloto automático del avión. Quiere probar la exactitud de la comunicación entre el componente de geolocalización y el componente del controlador del motor.

¿Cuál de los siguientes es el MEJOR ejemplo de una base de prueba para diseñar estas pruebas?

- A. Diseño detallado del componente de geolocalización.
- B. Diseño arquitectónico.
- C. Informe de análisis de riesgos.
- D. Normatividad del área de aviónica/derecho aeronáutico.

Elija una respuesta.

Pregunta 2.8

(FL-2.2.2, K2)

¿Cuál de los siguientes es un ejemplo de una prueba no funcional?

- A. Cubrir una determinada combinación de condiciones y observar las acciones ejecutadas para comprobar la correcta implementación de una determinada regla de negocio.
- B. Cubriendo el resultado “VERDADERO” en la decisión “SI ($x > 5$) ENTONCES...” en el código.
- C. Verificar que el sistema valida correctamente la sintaxis de la dirección de correo electrónico ingresada por el usuario en el formulario de registro de usuario.
- D. Verificar que el tiempo de inicio de sesión del sistema sea inferior a 5 ms cuando ya hay 1000 usuarios conectados al mismo tiempo.

Elija una respuesta.

Pregunta 2.9

(FL-2.2.3, K2)

¿Cuál de las siguientes pruebas normalmente no se puede programar con anticipación?

- A. Pruebas de regresión.
- B. Pruebas de aceptación operativa (OAT).
- C. Pruebas de aceptación de usuario (UAT).
- D. Pruebas de confirmación.

Elija una respuesta.

Pregunta 2.10

(FL-2.3.1, K2)

Durante las pruebas del software IoT (Internet de las cosas), se descubrió un defecto, pero no se solucionó debido a lo cercano del lanzamiento del software. Después del lanzamiento, hasta ahora no ha causado ningún fallo al cliente. Un mes después del lanzamiento, el equipo decidió solucionar el defecto.

¿Con cuál de los eventos desencadenantes de mantenibilidad estamos lidiando en esta situación?

- A. Actualización de software.
- B. Migración de software.
- C. Modificación del software.
- D. Agregar nuevas funciones al sistema.

Elija una respuesta.

Capítulo 3 Pruebas estáticas



Palabras clave

Anomalía Una condición que se desvía de las expectativas. Despues de ISO 24765 Pruebas dinámicas Pruebas que implican la ejecución del elemento de prueba. Segun ISO 29119-1 Revisión formal Un

tipo de revisión que sigue un proceso definido con un resultado documentado formalmente. Despues de ISO 20246 Revisión informal Un tipo de

revisión que no sigue un proceso definido y no tiene un resultado documentado formalmente. Un tipo de revisión formal que utiliza roles de

Inspección equipo definidos y mediciones para identificar defectos en un producto de trabajo y mejorar el proceso de revisión y el desarrollo de software. proceso. Despues de ISO 20246 Un tipo de prueba estática en la que un producto o proceso de trabajo es evaluado

Revisar por una o más personas para detectar defectos o proporcionar mejoras El proceso de evaluar un componente o sistema sin ejecutarlo, en función de su forma, estructura,

Análisis estático contenido, o documentación. Despues de ISO 24765 Pruebas estáticas Pruebas que no implican la ejecución de un elemento de prueba Revisión técnica Una revisión formal realizada

por expertos técnicos que examinan la calidad de un producto de trabajo e identifican discrepancias con las especificaciones y estándares Tutorial Un tipo de revisión en la que un autor dirige a los miembros de la revisión a través de un producto de trabajo y los miembros hacen preguntas y

comentarios sobre posibles problemas. Despues de ISO 20246.

Sinónimos: recorrido estructurado

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 prueba

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

estáticas FL-3.1.3 (K2) Comparar y contrastar estática y dinámica pruebas

Las pruebas ~~estáticas~~ son un conjunto de métodos y técnicas de prueba en las que el componente o sistema bajo prueba no se ejecuta (no se ejecuta). Las pruebas estáticas también pueden aplicarse a productos de trabajo no ejecutables distintos del software, como diseño, documentación, especificaciones, etc.

Los objetivos de las pruebas estáticas incluyen, en particular, la mejora de la calidad, la detección de defectos y la evaluación de características tales como legibilidad, integridad, corrección, comprobabilidad o consistencia del producto de trabajo bajo revisión. Por tanto, las pruebas estáticas se pueden utilizar tanto para la verificación como para la validación de un producto de trabajo.

En el desarrollo ágil de software, los evaluadores, los representantes comerciales (clientes, usuarios) y los desarrolladores trabajan juntos durante el desarrollo de requisitos (por ejemplo, escribiendo historias de usuarios, creando ejemplos o participando en sesiones de refinamiento del trabajo pendiente). El objetivo de este trabajo colaborativo es garantizar que los requisitos del usuario y los productos de trabajo relacionados cumplan ciertos criterios, como la definición de listo (ver Sección 5.1.3). Se pueden utilizar técnicas de revisión para garantizar que los requisitos sean completos, comprensibles y comprobables y, en el caso de historias de usuarios, contengan criterios de aceptación comprobables. Al hacer las preguntas correctas, los evaluadores examinan, cuestionan y ayudan a mejorar los requisitos propuestos.

Las pruebas estáticas se pueden dividir en dos grupos principales de técnicas:

- Análisis estático •

Revisiones

Análisis estático  Implica evaluar el producto de trabajo bajo prueba (generalmente código, requisitos o documentos de diseño) utilizando herramientas. El programa de estudios de Foundation Level no describe los métodos de análisis estático en detalle. Encontrarás más detalles en el programa de estudios de Nivel Avanzado "Analista de Pruebas Técnicas" [29]. Ejemplos de técnicas de análisis estático incluyen:

- Medición de código (p. ej., medición de su tamaño o complejidad ciclomática)
- Análisis de flujo de control
- Análisis de flujo de datos •

Comprobación de la compatibilidad de tipos de variables, verificación de la correcta aplicación de estándares de escritura de código (p. ej., denominación de variables), etc.

El análisis estático puede identificar problemas antes que las pruebas dinámicas, lo que requiere menos esfuerzo porque no se requieren casos de prueba y el análisis generalmente se realiza con herramientas. El análisis estático a menudo se incluye en los marcos de integración continua como un paso del proceso de implementación automatizada (consulte la Sección 2.1.4). Aunque se utiliza principalmente para detectar defectos de código específicos, el análisis estático también se utiliza ampliamente para evaluar la capacidad de mantenimiento, el rendimiento y la vulnerabilidad del código ante ataques de seguridad.

Las revisiones son técnicas de prueba estáticas mucho más utilizadas. Por lo tanto, los discutiremos con más detalle en la Sección 3.2.

3.1.1 Productos de trabajo examinables mediante pruebas estáticas

Las técnicas de prueba estáticas se pueden aplicar a prácticamente cualquier producto de trabajo. El programa de estudios enumera muchos ejemplos de productos de trabajo sujetos a pruebas estáticas. A continuación se proporciona una lista algo ampliada:

- Todo tipo de especificaciones (negocios, requisitos funcionales, requisitos no funcionales, etc.) • Epics, historias de usuarios, criterios de aceptación y otros tipos de documentación utilizados en proyectos ágiles
- Diseño de arquitectura • Código fuente: quizás excluyendo el código fuente escrito por terceros a los que no tenemos acceso (por ejemplo, en el caso de proyectos comerciales) • Todo tipo de software de prueba, incluidos planes de prueba, procedimientos de prueba, casos de prueba, automatizado guiones de prueba, datos de prueba, documentos de análisis de riesgos y cartas de prueba
- Manuales de usuario, incluida ayuda en línea integrada, manuales del operador del sistema, instrucciones de instalación y notas de la versión.
- Sitios web (en términos de su contenido, estructura, usabilidad, etc.) • Documentos del proyecto (por ejemplo, contratos, planes de proyecto, cronogramas, presupuestos)

Si bien la revisión puede aplicarse básicamente a cualquier producto de trabajo, para que tenga sentido, los participantes en la revisión deben poder leer y analizar el producto con comprensión. Además, en el caso del análisis estático, los productos de trabajo revisados mediante esta técnica deben tener una estructura formal contra la cual se realizan las pruebas. Ejemplos de tales estructuras formales son:

- Código fuente (ya que cada programa está escrito en un lenguaje de programación definido basado en una gramática formal)
- Modelos (por ejemplo, diagramas UML que tienen una sintaxis específica y reglas formales para creándolos)
- Documentos de texto (porque el texto se puede comparar, por ejemplo, con las reglas gramaticales del idioma)

El análisis estático se aplica con mayor frecuencia a productos de trabajo formalizados, como modelos o requisitos formales de arquitectura de sistemas (por ejemplo, escritos en lenguajes de especificación como Z o UML). Para documentos escritos en lenguaje natural, el análisis estático puede

implican, por ejemplo, comprobar la legibilidad, la sintaxis, la gramática, la puntuación o la ortografía.

3.1.2 Valor de las pruebas estáticas

Las pruebas estáticas no suelen ser baratas (requieren una comprobación manual del producto), pero, si se realizan correctamente, son efectivas y eficientes. Esto se debe a que permite encontrar defectos y problemas muy temprano en el SDLC que serían difíciles de detectar en fases posteriores. Esto generalmente se refiere a defectos de diseño. Este tipo de defecto suele ser muy insidioso, porque un defecto de diseño no detectado en las fases iniciales se propaga muy fácilmente a las fases posteriores y se crea una implementación defectuosa basada en un diseño defectuoso. El problema generalmente se hace evidente en la etapa de prueba de aceptación o del sistema, donde puede resultar muy costoso corregir dicho defecto de diseño.

Las pruebas estáticas bien realizadas antes de la ejecución de las pruebas dinámicas dan como resultado una eliminación rápida (y razonablemente económica) de problemas que podrían ser la fuente de otros defectos. Como resultado, se detectan menos problemas en las pruebas dinámicas, por lo que el costo total de las pruebas dinámicas es menor que si no se realizaran pruebas estáticas. Este uso de pruebas estáticas está en línea con el principio de pruebas tempranas y desplazamiento a la izquierda (ver Sección 1.3).

Las pruebas estáticas brindan la oportunidad de evaluar la calidad y generar confianza en el producto del trabajo revisado. Las partes interesadas pueden evaluar si los requisitos documentados describen sus necesidades reales. Debido a que las pruebas estáticas se pueden realizar en una etapa temprana del SDLC, se crea una comprensión común del producto y sus requisitos entre las partes interesadas involucradas en las pruebas estáticas. Este entendimiento compartido también mejora la comunicación. Por esta razón, se recomienda que las revisiones involucren a varias partes interesadas que representen las perspectivas más amplias y diversas sobre el producto del trabajo bajo revisión.

Los defectos en el código se pueden detectar y corregir usando pruebas estáticas de manera más eficiente que cuando se usan pruebas dinámicas, porque en las pruebas dinámicas, la ocurrencia de una falla generalmente requiere un análisis tedioso de la causa de la falla y dedicar una gran cantidad de tiempo a identificar el defecto, causándolo. Esta eficiencia de las pruebas estáticas generalmente da como resultado menos defectos restantes en el código y una menor carga de trabajo general.

La eficacia de las técnicas estáticas está demostrada empíricamente. Jones y Bonsignour [31] proporcionan una gran cantidad de datos que muestran que las técnicas estáticas contribuyen significativamente a la calidad general del producto y a reducir los costos de prueba y mantenimiento. En el caso de la inspección (un tipo de revisión que se describe en la siguiente sección), la eficiencia de eliminación de defectos aumenta en un promedio del 85%!

La siguiente es una lista de beneficios de las pruebas estáticas:

- Detección temprana y eficaz de defectos (y posible eliminación), incluso antes de que comiencen las pruebas dinámicas: la capacidad de realizar pruebas incluso antes de que se cree un prototipo de software funcional.

- Identificar defectos difíciles de detectar en pruebas dinámicas posteriores. Esto es especialmente cierto en el caso de defectos arquitectónicos y de diseño.
- Identificar defectos que son imposibles de detectar en pruebas dinámicas, como localizar código inviable (inalcanzable),¹ código no utilizado, uso incorrecto o falta de uso de patrones de diseño en el código, o todo tipo de defectos en productos de trabajo no ejecutables, como por ejemplo documentación.
- Prevenir la aparición de defectos en el diseño y el código detectando ambigüedades, contradicciones, omisiones, descuidos, elementos redundantes o inconsistencias en documentos como la especificación de requisitos o el diseño arquitectónico.
- Incrementar la eficiencia del trabajo de programación. Se puede mejorar el diseño y la mantenibilidad del código, por ejemplo, imponiendo un estándar uniforme para escribirlos. Esto hace que el código sea más fácil de modificar no sólo por el autor sino también por otros desarrolladores, y disminuye la posibilidad de introducir un defecto al modificar el código.

- Reducir el costo y el tiempo del desarrollo de software, incluidas las pruebas, especialmente las pruebas dinámicas.
- Reducir el coste de la calidad en todo el ciclo de desarrollo de software reduciendo costes en la fase de mantenimiento, así como reduciendo fallos en la fase de explotación del software.

- Mejorar la comunicación entre los miembros del equipo mediante la participación en revisiones, incluidas las reuniones de revisión.

Al evaluar los costos incurridos por las pruebas, se introduce el concepto de "costo de calidad". Es el costo total incurrido por las actividades de calidad, el cual consiste en los costos de:

- Actividades preventivas (por ejemplo, costos de capacitación)
- Detección (por ejemplo, costo de pruebas)
- Fallas internas (por ejemplo, costo de reparar defectos encontrados en producción)
- Fallas externas (por ejemplo, costo de arreglar defectos en campo encontrados por los usuarios)

Aunque las revisiones pueden ser costosas de implementar, los costos generales de calidad suelen ser mucho más bajos que si no se realizan, porque se debe dedicar menos tiempo y esfuerzo a la eliminación de defectos más adelante en el proyecto. Los participantes en el proceso de revisión también se benefician de una mejor comprensión compartida del producto que se revisa.

¹ La identificación de un código inviable sólo es posible en algunos casos, porque en general el problema de la accesibilidad a un determinado lugar en el código es el llamado problema indecidible. Esto significa que no existe ningún algoritmo que, para cualquier programa y cualquier lugar en su código, responda a la pregunta de si hay entradas al programa para las cuales el control llega a ese lugar en el código.

Ejemplo de análisis económico de inspecciones Considerere

una simulación económica simplificada de si vale la pena implementar pruebas estáticas en una organización. En el escenario A, el equipo realizará únicamente pruebas dinámicas. En el Escenario B, las pruebas dinámicas estarán precedidas por pruebas estáticas. El siguiente modelo es muy simple y solo pretende ilustrar cómo las pruebas estáticas pueden reducir el costo total del desarrollo y mantenimiento del software.

Supuestos del modelo:

- El equipo de probadores está formado por seis personas. • El costo mensual del salario de un probador es de \$7000. • Las pruebas estáticas y las pruebas dinámicas tienen una duración de 4 meses cada una y participa todo el equipo de pruebas. Las pruebas estáticas (si las hay) se realizan antes de las pruebas dinámicas.
- El número total de defectos de software es 130. • Las pruebas estáticas encuentran el 50% de todos los defectos existentes (ver, por ejemplo, [31] para datos de la industria).
- Las pruebas dinámicas encuentran el 80% de todos los defectos existentes. El 20% restante son defectos de campo, descubiertos por el cliente después del lanzamiento del software (estos valores se derivan de datos históricos). • Arreglar un defecto encontrado en pruebas estáticas cuesta \$500 en promedio (datos de la industria). • Arreglar un defecto encontrado en las pruebas dinámicas cuesta \$1800 en promedio (datos históricos). • Arreglar un defecto de campo (encontrado por el usuario después de que se lanza el software) cuesta \$12,600 (datos históricos).

Escenario A: sin pruebas estáticas

Número de defectos detectados en pruebas dinámicas: $80\% * 130 = 104$

Número de defectos de campo: $20\% * 130 = 26$

Costo de las pruebas dinámicas: $6 * \$7000 * 4 = \$168,000$

Costo de eliminación de defectos antes del lanzamiento: $104 * \$1800 = \$187,200$

Costo de eliminar defectos después del lanzamiento: $26 * \$12,600 = \$327,600$

Costo total de calidad: $\$168,000 + \$187,200 + \$327,600 = \$682,800$

Escenario B: con pruebas estáticas

Número de defectos detectados en pruebas estáticas: $50\% * 130 = 65$

Número de defectos detectados en pruebas dinámicas: $80\% * (130 - 65) = 52$

Número de defectos de campo: $130 - (65 + 52) = 13$

Costo de las pruebas estáticas: $6 * \$7000 * 4 = \$168,000$

Costo de las pruebas dinámicas: $6 * \$7000 * 4 = \$168,000$

(continuado)

Costo de eliminar defectos encontrados en pruebas estáticas: $65 * \$500 = \$32,500$

Costo de eliminación de defectos antes del lanzamiento: $52 * \$1800 = \$93,600$

Costo de eliminar defectos después del lanzamiento: $13 * \$12,600 = \$163,800$

Costo total de calidad: $\$168,000 + \$168,000 + \$32,500 + \$93,600 + \$163,800$
 $= \$625,900$

Esquemáticamente, la comparación de estos dos escenarios se muestra en la Fig. 3.1.

Los costos de calidad previos al lanzamiento son más altos en el escenario B (\$462,100 versus \$355,200), debido a los altos costos de inspección. Sin embargo, la ganancia se produce después del lanzamiento, debido a la mitad del número de defectos de campo en el Escenario B, gracias al uso de revisiones. Por lo tanto, el costo total resulta ser menor en el escenario B. La diferencia de costo exacta entre los escenarios A y B es $\$682,800 - \$625,900 = \$56,900$.

Por supuesto, en el análisis debemos tener en cuenta los errores de estimación, así como el hecho de que el equipo de pruebas necesitó cuatro meses adicionales para realizar pruebas estáticas. Sin embargo, el análisis muestra que el uso de pruebas estáticas no sólo puede mejorar significativamente la calidad final del producto (26 defectos de campo frente a 13) sino también reducir significativamente los costos de desarrollo. En la simulación anterior, ahorramos alrededor de \$57 000, por lo que los costos en el Escenario B son aproximadamente un 8,3 % menos que los del Escenario A.

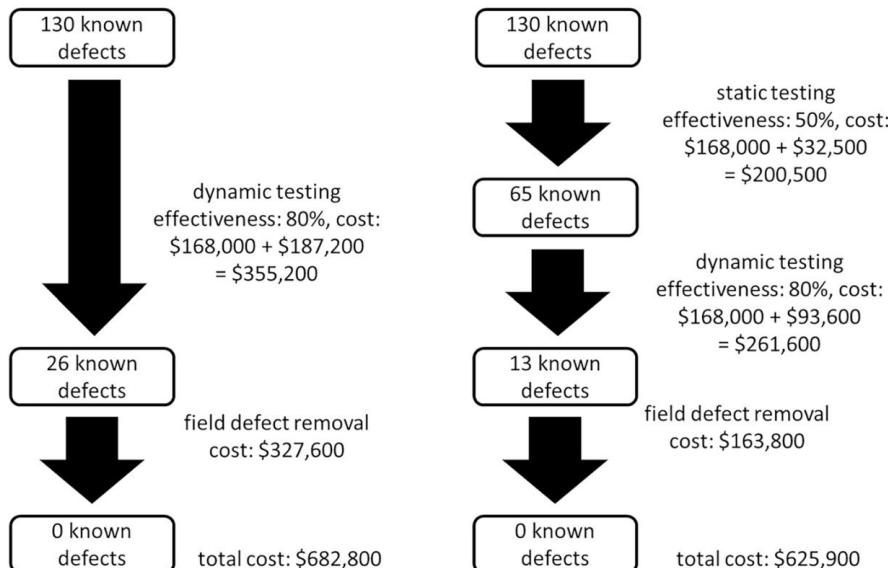


Fig. 3.1 Comparación de procesos de eliminación de defectos sin y con pruebas estáticas

3.1.3 Diferencias entre pruebas estáticas y pruebas dinámicas

Tanto las pruebas estáticas como las dinámicas tienen el mismo objetivo: evaluar la calidad del producto e identificar defectos lo antes posible. Estas actividades son complementarias, ya que permiten detectar diferentes tipos de defectos.

La Figura 3.2 representa simbólicamente la diferencia fundamental entre las pruebas estáticas (parte superior de la figura) y las pruebas dinámicas (parte inferior de la figura). Con las pruebas estáticas, podemos encontrar un defecto directamente en el producto de trabajo. No encontramos fallos, porque con las técnicas estáticas, por definición, no estamos ejecutando el software y un fallo sólo puede ocurrir como consecuencia del sistema que está ejecutando.

En el caso de las pruebas dinámicas, la mayoría de las veces el primer signo de un mal funcionamiento del software es, a su vez, una falla (el resultado de no pasar una prueba). Cuando se observa una falla, se inicia un proceso de depuración, durante el cual se analiza el código fuente para encontrar el defecto responsable de causar esta falla.

A veces, un defecto en un producto de trabajo puede permanecer oculto durante mucho tiempo porque no causa una falla. Además, es posible que la ruta en la que se encuentra rara vez se pruebe o sea de difícil acceso, lo que dificulta el diseño y la ejecución de pruebas dinámicas que lo detecten. Las pruebas estáticas pueden encontrar un defecto con mucho menos esfuerzo. Por otro lado, hay defectos que son mucho más fáciles de detectar con pruebas dinámicas que con pruebas estáticas, como las pérdidas de memoria.

Otra diferencia es que las pruebas estáticas se pueden utilizar para aumentar la coherencia y la calidad interna de los productos de trabajo, mientras que las pruebas dinámicas se centran principalmente en el comportamiento visible externamente.

Además, sólo se pueden aplicar pruebas estáticas a productos de trabajo no ejecutables, como código fuente o documentación. Las pruebas dinámicas, por otro lado, sólo se pueden realizar contra un producto de trabajo en ejecución, es decir, software en ejecución. A

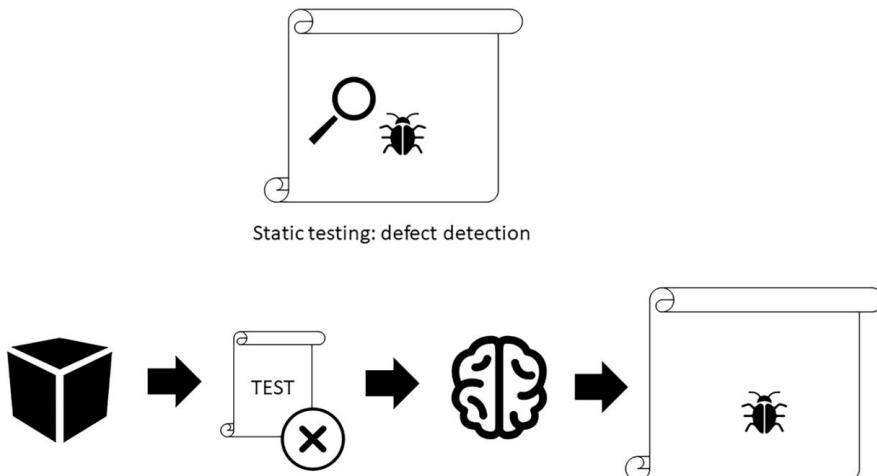


Fig. 3.2 Diferencia entre pruebas estáticas y pruebas dinámicas

La consecuencia de este hecho es que las pruebas dinámicas se pueden utilizar para medir algunas características de calidad que son imposibles de medir en las pruebas estáticas, porque dependen del programa en ejecución. Un ejemplo serían las pruebas de rendimiento que miden el tiempo de respuesta del sistema para una solicitud de usuario específica.

Mencionamos anteriormente que las pruebas estáticas generalmente detectan defectos diferentes a las pruebas dinámicas. Ejemplos de defectos típicos que son más fáciles y económicos de detectar y solucionar con pruebas estáticas que con pruebas dinámicas son:

- Defectos en los requisitos (como inconsistencias, ambigüedades, omisiones, contradicciones, inexactitudes, omisiones, repeticiones, elementos redundantes)
- Defectos de diseño (por ejemplo, algoritmos o estructuras de bases de datos inefficientes, alto acoplamiento, modularización deficiente del código, baja cohesión, ³ código)
- Tipos específicos de código defectos (p. ej., variables con valores indefinidos, variables declaradas pero nunca utilizadas, código inaccesible, código duplicado, algoritmos implementados de manera inefficiente con demasiado tiempo o complejidad de memoria)
- Desviaciones de los estándares (p. ej., falta de cumplimiento con el desarrollo del código estándares)
- Especificaciones de interfaz incorrectas (p. ej., uso de diferentes unidades de medida en los sistemas llamante y llamado, tipo incorrecto u orden incorrecto de parámetros pasados a una llamada de función API)
- Vulnerabilidades de seguridad (p. ej., susceptibilidad a ataques de desbordamiento de búfer, SQL (transferencia cruzada) inyección, XSS ⁴ secuencias de comandos del sitio), ataque ⁵)
- DDoS • Brechas o imprecisiones en la trazabilidad o cobertura (por ejemplo, no hay pruebas que coincidan con los criterios de aceptación para una historia de usuario determinada)

² El acoplamiento se refiere al grado en que un componente de software depende de otro y expresa el nivel de interdependencia entre módulos. Un alto acoplamiento significa que un cambio en un módulo puede requerir un cambio en otros módulos, lo que puede generar mayores costos de mantenimiento y desarrollo y dificultar la modificación y expansión del sistema. Un bajo acoplamiento significa que los componentes son más independientes entre sí, lo que los hace más fáciles de gestionar.

³ La cohesión se refiere al grado en que los componentes de un módulo están interrelacionados y persiguen un objetivo común. Alta cohesión significa que los componentes dentro de un módulo se centran en lograr el mismo objetivo único (tarea o funcionalidad), lo que los hace fáciles de entender, mantener y desarrollar. Una baja cohesión significa que los componentes dentro de un módulo persiguen objetivos diferentes, lo que los hace difíciles de entender, introduce dependencias innecesarias entre ellos y puede conducir a mayores costos de desarrollo y mantenimiento del sistema y a una legibilidad y escalabilidad deficientes.

⁴ XSS es una forma de ciberataque a sitios web para piratear a sus usuarios. El pirata informático coloca un script malicioso en un sitio web aparentemente amigable y seguro, lo que hace que cualquier persona que lo utilice sea vulnerable a ataques (según home.co.uk).

⁵ Un ataque DDoS implica lanzar un ataque simultáneamente desde múltiples ubicaciones al mismo tiempo (desde múltiples computadoras). Un ataque de este tipo se lleva a cabo principalmente desde ordenadores sobre los que se ha tomado el control mediante software especial (p. ej., bots y troyanos) (según home.pl).

Tabla 3.1 Análisis estático
resultados para el código
mantenibilidad

Componente	LOC	COMENTARIO	CC
principal	129	0%	7
dividir	206	1%	12
en_inferior	62	1%	6
a_superior	63	3%	6
fusionar	70	0%	15
configuración	42	15%	8
stdio	243	2%	21

Ejemplo El equipo de pruebas quiere evaluar la facilidad y el costo de mantener el software después de su entrega al cliente. Para ello, pretende utilizar el análisis estático. Se han identificado tres métricas que se utilizarán para medir la código fuente:

- LOC: el número de líneas de código ejecutables en un componente
- COMENTARIO: el porcentaje de líneas de código anotadas en un componente.
- CC: complejidad ciclomática⁶ de un componente

Los resultados se muestran en la Tabla 3.1.

El análisis estático llevó a las siguientes conclusiones:

- Los componentes son bastante pequeños. El mayor de ellos, stdio, tiene 243 líneas de código. Por lo tanto, el tamaño de los componentes no debería ser un problema importante. en cuanto a mantenibilidad.
- El código no está suficientemente comentado. Excepto por el módulo de configuración, en el que El 15% del código está comentado, en todos los demás módulos, el porcentaje de líneas comentadas está entre 0 y 3%. Los módulos principal y de combinación no contener comentarios en absoluto. Este hecho definitivamente debería informarse a los desarrolladores. atención.
- Tres componentes, split, merge y stdio, tienen valores altos (superiores a 10). Complejidad ciclomática. Deberían ser analizados. Esto es especialmente cierto para fusión en la que la densidad de declaraciones de decisión es muy alta: una decisión para aproximadamente cinco líneas de código, lo que puede sugerir una legibilidad muy pobre de la fuente código. No refactorizar estos componentes podría dificultar su mantenimiento. ellos en el futuro.

⁶ La complejidad ciclomática de un componente se define como el número de declaraciones de decisión en el código de ese módulo más uno. Esta es una medida muy simple de la complejidad de la estructura del código.

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 cuando realizar revisiones FL-3.2.4 (K2) Comparar y contrastar los diferentes tipos de revisión FL-3.2.5 (K1) Recordar los factores que contribuyen a una revisión exitosa

En los siguientes párrafos, discutiremos en detalle:

- Ventajas de la retroalimentación temprana y frecuente de los clientes
- Proceso de revisión del producto de trabajo
- Roles y responsabilidades en las revisiones formales
- Tipos de revisiones
- Factores de éxito en las revisiones

En la última subsección de este capítulo (Sección 3.2.6), analizamos técnicas para revisar los productos del trabajo. Esta sección estaba presente en el programa de estudios Foundation Level v3.1 pero se eliminó en el nuevo (v4.0). No obstante, decidimos dejarlo en el libro como una sección adicional, porque analiza la importante cuestión de cómo abordar de manera práctica la actividad misma de revisar un producto de trabajo. Esta etapa es un paso central en el proceso de revisión (ver Sección 3.2.2), por lo que es útil conocer las técnicas básicas de revisión. El material presentado en la Sección. 3.2.6 no es examinable.

3.2.1 Beneficios de la retroalimentación temprana y frecuente de las partes interesadas

Las revisiones  son una forma de proporcionar retroalimentación temprana al equipo porque se pueden realizar en una etapa temprana del ciclo de desarrollo. Sin embargo, la retroalimentación no debe limitarse únicamente a revisiones, sino que debe ser una práctica común durante todo el ciclo de desarrollo del proyecto. Los comentarios pueden provenir de evaluadores o desarrolladores, pero igualmente importante para el equipo será el del propio cliente.

La retroalimentación temprana y frecuente permite una notificación rápida de posibles problemas de calidad y permite al equipo responder rápidamente al problema. Si hay poca participación de las partes interesadas durante el desarrollo de software, es posible que el producto que se está desarrollando no cumpla con su visión original o actual. No cumplir con lo que las partes interesadas esperan puede resultar en costosas reelaboraciones, incumplimiento de plazos y juegos de culpas e incluso llevar al fracaso total del proyecto.

La retroalimentación frecuente de las partes interesadas durante el desarrollo de software puede evitar malentendidos sobre los requisitos y garantizar que los cambios en los requisitos se realicen correctamente.

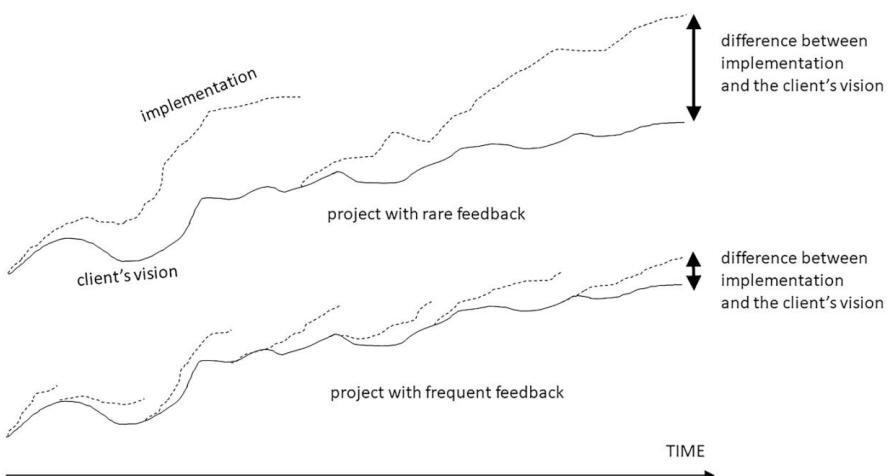


Fig. 3.3 Ilustración del beneficio de la retroalimentación frecuente

entendido e implementado antes. Esto ayuda al equipo a comprender mejor lo que están construyendo. Les permite centrarse en aquellas funciones que ofrecen el mayor valor a las partes interesadas y que tienen el impacto más positivo en los riesgos acordados.

La figura 3.3 muestra simbólicamente el beneficio de la retroalimentación frecuente. La línea continua representa la visión del cliente y la línea discontinua representa cómo se ve realmente el producto en su implementación. Estas dos visiones divergen cada vez más con el tiempo. En los momentos de retroalimentación, la implementación se ajusta a la visión del cliente. El gráfico superior muestra una situación en la que el contacto con el cliente es esporádico y se produce sólo dos veces. Se puede ver que el resultado es una implementación que difiere significativamente de lo que era la visión del cliente. El gráfico inferior representa una situación en la que el cliente frecuentemente proporciona comentarios al equipo. Esto da como resultado menos diferencias entre la visión del cliente y la implementación real, porque transcurre mucho menos tiempo entre las reuniones con el cliente que en la situación del gráfico superior. Como resultado, el producto final está mucho más alineado con la visión del cliente. Además, cabe señalar que el coste de "ajustar" el producto a la visión del cliente será mucho mayor en el caso de feedback poco frecuente, debido a que habrá que modificar y ajustar el producto en mucha mayor medida. que en la situación de retroalimentación frecuente.

3.2.2 Actividades del proceso de revisión

Las revisiones varían en tipo, nivel de formalización u objetivos, pero todos los tipos de revisiones tienden a tener fases o grupos de actividades similares. La norma ISO 20246 Ingeniería de software y sistemas: revisiones de productos de trabajo [6] define un proceso de revisión genérico que proporciona un marco estructurado pero flexible dentro del cual adaptar un

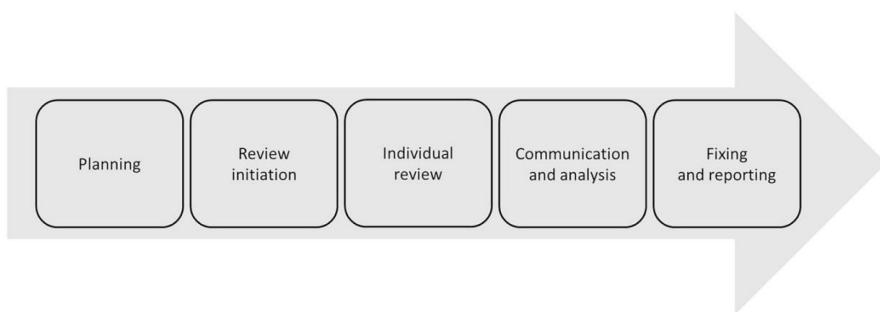


Fig. 3.4 Proceso de revisión genérico

proceso de revisión específico a una situación determinada. Si la revisión requerida es más formal, entonces habrá más tareas o elementos del proceso. En la Fig. 3.4 se muestra un proceso de revisión genérico descrito en ISO/IEC 20246 .

El tamaño de muchos productos de trabajo significa que pueden ser demasiado grandes para ser cubiertos por una sola revisión. En tales casos, el proceso de revisión generalmente se aplica varias veces a las partes individuales que componen el producto del trabajo.

El programa de estudios del nivel básico describe cinco tipos genéricos de actividades que se muestran en la figura 3.4 y que pueden ocurrir en el proceso de revisión del producto del trabajo. Los comentamos a continuación.

Planificación

La planificación define el alcance del trabajo que cubrirá la revisión. La planificación establece los límites de la revisión respondiendo preguntas como quién, qué, dónde, cuándo y por qué. Se define el propósito de la revisión (respondiendo a la pregunta "por qué"), así como qué documentos serán objeto de la revisión. Se definen las características de calidad a revisar (respondiendo a la pregunta "qué"). Se estima la cantidad de trabajo necesario para realizar la revisión y se definen el marco de tiempo y la ubicación de ciertas fases del proceso de revisión, como la reunión de revisión (respondiendo las preguntas de "dónde" y "cuándo"). Con base en el propósito de la revisión, se define el tipo de revisión, junto con los roles, actividades y listas de verificación que se utilizarán (si es necesario). Se seleccionan las personas que participarán en la revisión y se asignan roles (respuesta a la pregunta "quién").

Para revisiones formales como la inspección, se definen criterios formales de entrada y salida, y al final de esta fase también se verifica que se cumplan los criterios de entrada y que la revisión pueda pasar a la siguiente fase.

Inicio de la revisión

Como parte del inicio de la revisión, el producto del trabajo que se va a revisar se envía a los participantes de la revisión (seleccionados en la fase de planificación), junto con todos los demás materiales necesarios, por ejemplo, listas de verificación, declaraciones de procedimiento e informe de defectos. plantillas. Todos los materiales deben distribuirse lo antes posible para que los revisores tengan tiempo suficiente para completar la revisión.

Si es necesario, se dan explicaciones a los participantes sobre cómo se llevará a cabo la revisión y cuál es su papel. Si los participantes tienen alguna pregunta, se brindan respuestas y explicaciones durante la fase de inicio de la revisión (antes de que se lleve a cabo la revisión real del producto del trabajo) para que todos estén conscientes de lo que se supone que deben hacer y conozcan el cronograma de estas actividades.

Es posible organizar una capacitación de revisión durante esta fase. Esto tiene sentido cuando los participantes no tienen experiencia en revisiones y desea que el proceso de revisión se desarrolle sin problemas y de manera eficiente. Para revisiones formales, también se puede celebrar una reunión inicial para explicar a los participantes individuales el alcance de la revisión y sus roles y responsabilidades individuales como revisores.

El propósito de la fase de inicio de la revisión es garantizar que todos los involucrados en la revisión estén preparados y listos para comenzar la revisión.

Revisión individual (es decir, preparación individual)

Esta es la fase central y esencial de cualquier revisión. En esta fase se realizan actividades sustantivas, es decir, se lleva a cabo la revisión real del producto del trabajo por parte de los revisores, utilizando técnicas de revisión específicas (ver Sección 3.2.6). Como parte de estas actividades, se revisa todo o parte del producto del trabajo (la parte que se va a revisar debe determinarse durante la planificación y explicarse cuidadosamente a los participantes en la fase de inicio de la revisión). Los revisores registran cualquier comentario, pregunta, recomendación, inquietud y observación relevante realizada durante la revisión del producto del trabajo.

De todas las fases de revisión, la fase de revisión individual detecta el mayor porcentaje de problemas. Los problemas identificados generalmente se documentan en un registro de problemas, que a menudo está respaldado por una herramienta de soporte de revisión o gestión de defectos.

Según la norma ISO/IEC 20246 [6], este paso del proceso es opcional⁷ y no puede realizarse para ciertos tipos de revisión, como recorridos o revisiones informales. Sin embargo, muchos autores adoptan la posición de que es la actividad más importante de todo el proceso de revisión.

Comunicación y análisis Si no se

programa una reunión de revisión, los problemas encontrados se comunican directamente a las personas (por ejemplo, el autor del producto de trabajo que se está revisando), junto con un análisis de esos problemas. Si la reunión de revisión está incluida en el plan de revisión, los problemas se informan directamente a los participantes de la reunión o, lo que probablemente sea mejor, se envían colectivamente a todos los participantes antes de la reunión para que los revisen con anticipación, de modo que la reunión de revisión funcionará de manera más eficiente.

La reunión incluye un análisis de los problemas (anomalías) informados por los revisores. Dado que no todas las anomalías necesariamente resultan ser un defecto, todos los hallazgos deben revisarse cuidadosamente en la reunión para decidir si existe un problema real (defecto) o simplemente un problema aparente (falso positivo). Si la anomalía resulta ser un defecto, se designan los responsables de solucionarla y se definen parámetros como estado, prioridad o gravedad. Estas acciones se realizan en una revisión.

⁷ En tales situaciones, la detección de defectos puede tener lugar, por ejemplo, durante la llamada reunión de revisión que se analiza en el siguiente párrafo.

reunión o (si no existe dicha reunión) individualmente. Los estados más utilizados para anomalías reportadas son:

- Problema rechazado •
Problema registrado sin tomar ninguna medida • Problema a resolver por el autor del producto de trabajo • Problema actualizado como resultado de un análisis adicional •
Problema asignado a una parte interesada externa

Esta fase también evalúa y documenta el nivel de las características de calidad que se definieron en la fase de planificación como las que se están revisando. Finalmente, las conclusiones de la revisión se evalúan según los criterios de salida para decidir qué hacer con el producto del trabajo revisado. Ejemplos de decisiones incluyen:

- Aceptación del producto del trabajo (normalmente cuando no hay problemas o sólo un pequeño se detectan varios problemas insignificantes).
- Actualizar el producto de trabajo en base a los problemas identificados (decisión típica). • El producto del trabajo debe revisarse nuevamente (generalmente debido a la gran cantidad de problemas y el gran alcance de los cambios).
- Rechazo del producto del trabajo (el tipo de decisión más rara, pero también puede ocurrir).

Reparación e informes

Esta es la etapa final de la revisión. Crea informes de defectos detectados que requieren cambios. Se espera que el autor del producto de trabajo revisado lleve a cabo la eliminación de defectos durante esta fase. Esto se hace informando a las personas relevantes o al equipo relevante de los defectos detectados en el producto de trabajo bajo revisión. Finalmente, cuando se confirman los cambios, se crea un informe de revisión.

Para tipos de revisión más formales, además, se recopilan y utilizan varios tipos de medidas para probar la eficacia de la revisión. También se compara con los criterios de salida y el producto del trabajo se acepta una vez que se determina que se han cumplido los criterios de salida.

Los resultados de una revisión del producto de trabajo pueden variar según el tipo de revisión y el grado de formalización (ver Apartado 3.2.4).

Ejemplo Un equipo ha decidido realizar una revisión del llamado árbol de habilidades en un juego de rol en línea para computadora que están produciendo. El líder de la prueba selecciona a cinco personas como participantes en la revisión: un desarrollador, un evaluador y tres personas externas al equipo, que actúan como partes interesadas externas (jugadores). Se decide que el tipo de revisión realizada será una revisión informal individual.

El líder del equipo distribuye a los participantes un documento que describe el árbol de habilidades, una lista de verificación de las características que se deben verificar y una plantilla de registro de problemas, que se muestra en la Tabla 3.2.

Cada participante, en un momento predeterminado, revisa el documento y completa el formulario de registro de problemas y luego lo envía al líder del equipo. El líder del equipo fusiona los formularios en uno, elimina los defectos redundantes (posiblemente marcando que fueron encontrados por más de una persona) y envía la lista de problemas así creados al

Tabla 3.2 Plantilla de registro de problemas

No.	Página/línea	Categoría de defecto	Tipo de defecto de gravedad	Fuente del defecto	Descripción, comentarios
...
...
Defect categories: Missing, Defect, Redundancy					
<hr/>					

Tabla 3.3 Un registro de problemas completado por uno de los revisores

No.	Página/línea	Categoría de defecto	Gravedad	Tipo de defecto	Descripción, comentarios
1	Figura 1	Defecto	Pequeño	Sintaxis	"Iluminación eléctrica" en lugar de "Rayo eléctrico"
2	Figura 1	Defecto	Grande	Lógica	El elemento que sigue a "Eléctrico Rayo" debería ser la habilidad "Tormenta eléctrica", no "Tormenta de fuego".
3	Figura 1	Desaparecido	Grande	Lógica	La habilidad "cegar al oponente" falta, la introducción de cuál al árbol de habilidades se acordó en la reunión del 4 de febrero

desarrolladores, al mismo tiempo transformando los defectos detectados en informes de defectos y los registra en la herramienta de gestión de defectos.

En la Tabla 3.3 se muestra un ejemplo de los defectos informados por uno de los revisores .

Dado que el revisor no utilizó la lista de verificación, la columna "Fuente del defecto" no fue necesario, por lo que esta columna se eliminó del registro de problemas.

Los autores corren los defectos encontrados y los registran en la herramienta de gestión de defectos. El líder de revisión verifica que se hayan solucionado todos los defectos. El líder de revisión entonces crea un informe de revisión resumido que contiene información sobre:

- Número de personas que participan en la revisión
- Duración de la revisión
- Tamaño del documento que se está viendo (por ejemplo, número de líneas de código, número de páginas del documento)
- Número de defectos encontrados, desglosados por gravedad (grandes/pequeños)

Aquí es donde termina el proceso de revisión.

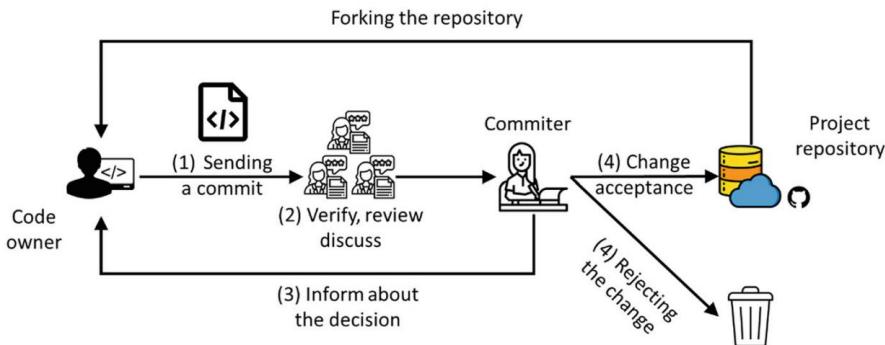


Fig. 3.5 Proceso de revisión de código moderno

Efectividad de las reuniones de revisión

El plan de estudios de Foundation Level dice que la reunión de revisión es uno de los pasos del proceso de revisión. Si analizamos históricamente cómo se han organizado las revisiones, podemos ver que hasta hace algún tiempo, las revisiones se centraban en reuniones de revisión, que eran la actividad principal y sustancial del proceso de revisión. En los últimos doce años se ha estudiado el problema de la necesidad y eficacia de estas reuniones. Estos estudios han concluido que, en general, estas reuniones no aumentan significativamente la eficacia de la detección de defectos (que suele ser el objetivo más importante de las revisiones). Las reuniones de revisión son costosas, por lo que no parecen agregar ningún valor particular. Además, resulta que si hay demasiadas personas en una reunión (más de cinco), la sobrecarga de comunicación comienza a tener un impacto negativo en el aprendizaje y mejora de habilidades de los asistentes.

Por otro lado, las reuniones de revisión son una excelente oportunidad para hablar e intercambiar experiencias, compartir conocimientos o lograr consensos. Los estudios también demuestran que estas reuniones ayudan a reducir el número de falsos positivos (informes de anomalías que resultan no ser defectos). Así, el valor añadido puede revelarse no necesariamente en el número de defectos detectados, sino en la mejora de todo el proceso de desarrollo y en la mejora de las habilidades de los miembros del equipo [32].

Revisión de código moderno (MCR)

Hoy en día, muchas empresas utilizan un proceso de revisión de código conocido como Revisión de código moderno (MCR). En la figura 3.5 se muestra un diagrama de este proceso. MCR es una técnica eficaz de control de calidad que puede verificar la calidad del software y la satisfacción del cliente identificando defectos, mejorando el código y acelerando el proceso de desarrollo. Es un proceso de revisión asincrónico y liviano respaldado por herramientas de revisión como Gerrit. Es una versión ligera del proceso de inspección de Fagan (ver Sección 3.2.4) y ha evolucionado como una práctica para el desarrollo de software industrial y de código abierto [33].

3.2.3 Funciones y responsabilidades en las revisiones

En una revisión típica, se distinguen los siguientes roles (la descripción de los roles y la lista de responsabilidades se han tomado del programa de estudios; al final, damos algunas notas adicionales sobre algunos de los roles):

Gerente •

Es responsable de programar la revisión • Decide realizar una revisión

- Designa personal y establece un presupuesto y un calendario.
- Supervisa la rentabilidad de la revisión de forma continua. • Ejecuta decisiones de control en caso de resultados insatisfactorios.

Autor •

Crea el producto de trabajo bajo revisión • Elimina defectos en el producto de trabajo bajo revisión (si es necesario)

El autor es la persona responsable de preparar los materiales para revisión, aunque formalmente los materiales pueden ser distribuidos por el líder de revisión. Si es necesario, el autor puede proporcionar a los revisores explicaciones técnicas del producto del trabajo que se está revisando. Es muy importante que los autores comprendan y acepten las críticas profesionales de los revisores sobre el producto de su trabajo y que no defiendan ni nieguen los comentarios de los revisores.

Después de todo, el objetivo de la reseña suele ser descubrir tantos problemas como sea posible, y la crítica no está dirigida al autor (porque todo el mundo es falible), sino al producto.

Sin embargo, puede suceder que el autor no comprenda el comentario del revisor y el autor no comprenda cuál es el problema y, por lo tanto, no sepa cómo solucionarlo. Entonces puede ser necesario comunicarse entre el autor y el revisor para explicar las inquietudes del revisor con más detalle.

El autor también puede evaluar el trabajo de los revisores en términos del valor de los comentarios que hacen. Este tipo de retroalimentación se puede utilizar para planificar revisiones futuras en la organización.

Moderador (también conocido como facilitador) •

Garantiza el buen funcionamiento de las reuniones de revisión (si se llevan a cabo) •

Actúa como mediador si es necesario conciliar diferentes puntos de vista • Garantiza que se cree una atmósfera segura de confianza y respeto mutuos en la revisión reunión

Escriba (también conocido como registrador)

• Reúne posibles anomalías detectadas e informadas como parte de la revisión individual • Registra nuevos defectos potenciales encontrados durante la reunión de revisión, así como decisiones tomadas en la reunión (si se lleva a cabo)

El escribano debe desempeñar un papel “transparente” durante la reunión de revisión, es decir, debe ser “invisible” para los demás participantes. Su función es aliviar al otro.

participantes de la tarea de registrar cualquier comentario realizado durante la reunión. Una reunión de revisión requiere la máxima concentración por parte de los revisores y del autor, y trabajarán de manera más efectiva cuando no tengan que distraerse cada vez que se encuentre un nuevo problema y sea necesario escribirlo.

Revisor •

Realiza una revisión, identificando defectos potenciales en el producto de trabajo bajo revisión. • Puede ser un experto en la materia, una persona que trabaja en el proyecto, una parte interesada en el producto de trabajo y/o una persona con experiencia técnica o comercial específica.

- Puede representar diferentes puntos de vista (por ejemplo, el punto de vista de un evaluador, desarrollador, usuario, operador, analista de negocios, especialista en usabilidad)

Los revisores desempeñan un papel clave en el proceso de revisión, ya que encuentran problemas en el producto del trabajo que se está revisando, y este suele ser el objetivo principal de las revisiones. A los revisores se les debe dar tiempo suficiente para prepararse e informar los problemas que encuentren. Los revisores deben dirigir sus comentarios al producto, no al autor.

Líder de revisión •

Tiene la responsabilidad general del proceso de revisión • Decide quién participará en la revisión, determina el lugar y la fecha de la revisión y es responsable de organizar las reuniones de revisión

Roles versus personas

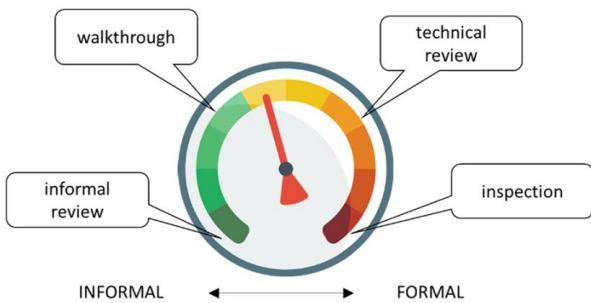
Para algunos tipos de revisiones, una persona puede desempeñar varios roles y, según el tipo de revisión, las actividades asociadas con cada rol también pueden variar. Además, con la llegada de herramientas para ayudar en el proceso de revisión (y en particular el registro de defectos, puntos abiertos y decisiones), a menudo no es necesario nombrar un escribano. Tampoco será necesario un escribano si no se planea una reunión de revisión.

3.2.4 Tipos de revisión

Hay muchos tipos de revisiones con diferentes niveles de formalidad, desde revisiones informales hasta revisiones altamente formalizadas (ver Fig. 3.6). El nivel de formalidad requerido depende de factores como el modelo SDLC utilizado, la madurez del proceso de desarrollo, la criticidad y complejidad del producto de trabajo que se revisa, cualquier requisito legal o reglamentario y la necesidad de una pista de auditoría.

Elegir el tipo correcto de revisión es fundamental para lograr los objetivos de revisión requeridos. La selección se basa no sólo en los objetivos sino también en factores como las necesidades del proyecto, los recursos disponibles, el tipo de producto del trabajo, los riesgos, el dominio empresarial y la cultura organizacional.

Fig. 3.6 Nivel de formalidad de los diferentes tipos de revisión



El programa de estudios de Foundation Level describe cuatro tipos de revisiones:

- Inspección •
- Revisión técnica •
- Recorrido • Revisión informal

La inspección es una revisión formal. La revisión técnica y el recorrido pueden variar desde muy formalizados hasta bastante informales. El sello distintivo de las revisiones informales es que no siguen un proceso definido y la información obtenida a través de ellas no necesita estar documentada formalmente. Las revisiones formales, en cambio, se realizan de acuerdo con procedimientos documentados y con la participación de un equipo preestablecido, y los resultados deben documentarse obligatoriamente.

Las revisiones se pueden realizar con diversos fines, pero uno de los objetivos principales de cualquier revisión es encontrar defectos. Las reseñas se pueden clasificar según varios atributos.

La Tabla 3.4 muestra los tipos de revisión más comunes discutidos en el programa de estudios de Foundation Level, junto con sus atributos correspondientes. A continuación, proporcionaremos información más detallada sobre estos tipos de revisiones.

Orden de revisiones

Un producto de trabajo puede estar sujeto a más de un tipo de revisión, y si se realizan varias revisiones de diferentes tipos, su orden puede variar. Por ejemplo, se puede realizar una revisión informal antes de una revisión técnica para garantizar que el producto del trabajo esté listo para la revisión técnica.

Revisiones por pares

Todos los tipos de revisiones pueden implementarse como revisiones por pares, es decir, realizadas por colegas de un nivel organizacional similar o con responsabilidades o experiencia similares.

Tipos de defectos encontrados durante las revisiones

Los tipos de defectos encontrados en las revisiones varían, dependiendo en particular del producto de trabajo que se revisa. Para ver ejemplos de defectos encontrados en revisiones de diversos productos de trabajo, consulte la Sección. 3.1.3, y para obtener información sobre revisiones formales, consulte [34].

Pasamos ahora a una discusión detallada de los cuatro tipos de revisiones descritas en el programa de estudios. En la Tabla 3.4 se resume una comparación resumida de los tipos de revisión que se analizan a continuación .

Tabla 3.4 Comparación de tipos de revisión

tipo de reseña	Informal revisar	Tutorial	Revisión técnica	Inspección
Objetivos principales	Detectar posibles defectos	Detectar potencial defectos, mejorar calidad, considere alternativas, evaluar el cumplimiento con estándares	Obtener consenso, detectar potencial defectos	Detectar potencial defectos, evaluar el calidad de la producto de trabajo, aumentar la confianza en él, prevenir similares defectos de ocurriendo en el futuro
Adiciones potenciales objetivos nacionales	Generar nuevo ideas, rápidamente resolver sencillo problemas	Intercambiar información, capacitar a los participantes, llegar consenso	evaluar la calidad de un producto de trabajo, aumentar la confianza en él, generar nuevas ideas, motivar a los autores a mejorar el futuro productos del trabajo, evaluar alternativas	Motivar a los autores para mejorar el futuro productos de trabajo y El software proceso de desarrollo, crear condiciones para ello, llegar a un consenso
Proceso formal Ninguno		Preparación individual opcional antes de reunión	Preparación individual obligatoria antes de la reunión	Proceso formal basado en reglas y listas de verificación; individuo obligatorio preparación antes reunión
Roles	Se puede ejecutar por un autor, por un autor compañero, o por un grupo de gente	La Junta es generalmente presidido por el autor; el presencia de un se requiere escribanos	La reunión debiera ser realizado por un moderador, no por el autor; el presencia de un se requiere escribanos	Estrictamente definido; La reunión está dirigida por un moderador, no el autor; escriba es obligatorio; lector opcional role
Documentación de resultados	Opcional	Defecto opcional registros y revisión los informes son creado	En general, defecto registros y revisión se crean informes	En general, defecto registros y revisión se crean informes
Nivel de formalismo	Informal	De informal a formal; puede tomar la forma de escenarios, simulacros, o simulaciones	De informal a formal; revisar reunión opcional	Formal; entrada y Los criterios de salida están en lugar; medidas se recogen que se utilizan para mejorar todo proceso, incluyendo la inspección proceso
Listas de verificación	Opcional	Opcional	Opcional	Usualmente usado

Revisión informal La

revisión informal no sigue ningún proceso establecido o predefinido. El resultado de una revisión informal tampoco está documentado de manera formal. El objetivo principal de una revisión informal es detectar posibles anomalías.

Un ejemplo de revisión informal podría ser una conversación entre dos desarrolladores, cuando uno de ellos le pide ayuda al otro para encontrar un defecto que causa un error de compilación. Otro ejemplo podría ser una conversación entre dos ingenieros en la cocina de la oficina durante el almuerzo sobre algún problema técnico. Después de una revisión informal, a menudo no queda rastro y no se documenta nada. En metodologías ágiles, este es el tipo de revisión más común.

Ejemplos de tipos de revisión informal incluyen:

- Revisión de compañeros
- Revisión de pareja

Tutorial El tutorial

implica la revisión secuencial de un producto de trabajo. Lo lleva a cabo el autor del producto del trabajo y puede incluir varios objetivos diferentes, como evaluar la calidad del producto del trabajo, aumentar la confianza en el producto del trabajo, mejorar la comprensión de los revisores sobre el producto del trabajo, llegar a un consenso y generar nuevas ideas. y motivar a los autores a crear mejores productos y encontrar defectos de manera más efectiva. Los revisores pueden realizar una preparación individual antes de la reunión de recorrido, pero no es obligatorio.

Este tipo de revisión también se suele utilizar cuando el equipo no puede detectar la causa de una falla del software. Luego puede adoptar la forma de los llamados ensayos. Una ejecución de prueba puede implicar una simulación manual de la ejecución del código. Luego, el equipo analiza el código cuidadosamente, línea por línea, para comprender bien cómo funciona y localizar el defecto que causa la falla.

El tutorial lo realiza el autor del código, ya que el autor generalmente puede explicar sobre la marcha qué hace el código (o al menos cuáles eran sus intenciones a este respecto cuando implementaron el código). Cuando las revisiones de código se realizan en reuniones de revisión, generalmente toman la forma de un recorrido.

Revisión técnica Los

objetivos de una revisión técnica, realizada por revisores técnicamente calificados, son lograr consenso y tomar decisiones sobre un problema técnico, pero también detectar defectos potenciales, evaluar la calidad y generar confianza en el producto del trabajo, generar nuevas ideas y motivar. y permitir a los autores mejorar el producto de su trabajo.

Una revisión técnica generalmente toma la forma de un "panel de expertos", es decir, una reunión en la que personas competentes tienen la tarea de tomar alguna decisión técnica o de diseño, generalmente importante y significativa, que tiene un impacto importante en el desarrollo futuro del proyecto. . Por este motivo, es obligatoria la preparación individual antes de la reunión, para que en la propia reunión todos puedan participar de forma informada y competente.

En general, una reunión de revisión en una revisión técnica es opcional. Sin embargo, si se lleva a cabo, debe ser dirigido por un facilitador. Por lo general, la revisión técnica finaliza con un informe, ya que debe dejarse un rastro, especialmente si se tomaron decisiones de diseño importantes durante la revisión.

Inspección

Las inspecciones son uno de los tipos de revisión más formales. Su origen se remonta a la década de 1970, cuando Michael Fagan los introdujo en IBM [35]. La realización de inspecciones de software se considera una de las llamadas mejores prácticas en ingeniería de calidad del software debido a los indudables beneficios que aporta. La alta dirección ha enfatizado sistemáticamente el importante papel de las revisiones por pares (incluidas las inspecciones) como elemento clave para garantizar una alta calidad del producto final [36].

Dado que las inspecciones son el tipo de revisión más formal, siguen el proceso de revisión completo descrito en la Sección. 3.2.2. El objetivo principal es lograr la máxima eficiencia en la búsqueda de defectos. Otros objetivos son evaluar la calidad, generar confianza en el producto del trabajo y motivar y permitir a los autores mejorar sus productos. Una característica especial de las inspecciones es recopilar métricas y utilizarlas para mejorar todo el proceso de desarrollo de software, incluido el proceso de inspección en sí. En las inspecciones, el autor no debe asumir el papel de líder de revisión, moderador o escribano.

Las inspecciones siguen un proceso bien definido basado en reglas y listas de verificación. Los resultados de las inspecciones deben documentarse. Un ejemplo de una lista de verificación utilizada durante una inspección podría ser el siguiente.

Lista de verificación de requisitos

Lo completo

1. ¿Los requisitos especificados se relacionan con la misión del proyecto de manera consistente? ¿manera?
2. ¿Los requisitos incluyen las necesidades clave y críticas del usuario, operador y equipo de apoyo?
3. ¿Es cada requisito una unidad de especificación independiente de los demás o tiene dependencias claramente definidas?
4. ¿No existen deficiencias en los requisitos?
5. ¿Se distinguen los requisitos necesarios de los requisitosopcionales?

Exactitud

1. ¿Los requisitos no son contradictorios entre sí?
2. ¿Se realiza un seguimiento de los requisitos hasta el diseño y el código?
3. ¿Cada requisito tiene un número único?

Estilo

1. ¿Cada requisito es fácil de entender y está escrito en un lenguaje sencillo?
2. ¿Los requisitos hacen una distinción clara entre editorial y funcional? ¿cambios?
3. ¿Se utilizan de manera coherente y uniforme la nomenclatura y las definiciones de los términos?

Medidas de inspección Aunque

muchas organizaciones utilizan inspecciones, hay pocos datos disponibles públicamente sobre su eficacia. Sin embargo, basándose en la información existente (particularmente del Experimento Nacional de Calidad del Software, realizado en 1992), se pueden extraer las siguientes conclusiones [37]:

- No se rastrea el código fuente hasta los requisitos, lo que resulta en pérdida de control "intelectual", imprecisión de los procedimientos de verificación y dificultad en la gestión de cambios.
- Las buenas prácticas de escritura de código se aplican de manera no rigurosa y no sistemática, lo que resulta en un alto porcentaje de defectos en lógica, datos, interfaces y funcionalidad.
- Los diseños arquitectónicos de las aplicaciones a menudo se crean ad hoc, lo que reduce drásticamente la comprensibilidad, adaptabilidad y mantenibilidad del producto.
- No se hace un uso significativo de los patrones de diseño y programación existentes.
- La distribución de los tipos de defectos detectados durante la inspección es la siguiente:
 - Defectos de documentación—40,51% –
 - Incumplimiento de estándares—23,20% – Defectos de lógica—7,22% – Defectos funcionales—6,57% – Defectos de sintaxis—4,79% – Defectos de datos—4,62% – Defectos de mantenibilidad—4,09%
- En promedio, un equipo necesita entre 8 y 16 minutos para detectar un defecto y alrededor de 80 minutos para detectar un defecto de alta gravedad (conocido como defecto mayor).
- Aproximadamente 3,2 defectos mayores y 17 defectos menos significativos por cada Se encuentran 1000 líneas de código en el código bajo inspección.
- En promedio, el equipo es capaz de analizar 625 líneas de código en una hora.
- En promedio, el equipo encuentra 4,6 defectos por sesión.
- La relación entre el esfuerzo de preparación para la inspección y el esfuerzo de inspección es 0,58.
- La tasa de detección de defectos oscila entre el 80% y el 90%; esto significa que alrededor del 80% al 90% de todos los defectos detectados se encuentran durante la inspección.
- El retorno de la inversión (ROI calculado como la relación entre ganancias y costos) es 4.1.

Las métricas que se pueden recopilar como parte del proceso de inspección incluyen:

- Tiempo de preparación por defecto
- Tiempo de preparación por defecto mayor
- Número de defectos críticos por 1000 líneas de código •
- Número de defectos no críticos por 1000 líneas de código • Número de líneas de código verificadas en 1 h

(continuado)

- Número de defectos por sesión •
- Esfuerzo para preparar versus esfuerzo para inspeccionar • Número de líneas revisadas en una sesión

El análisis de estas métricas, especialmente a partir de una serie de inspecciones realizadas, permite estimar el coste, el esfuerzo y la eficacia de esta forma de prueba estática y calcular el retorno de la inversión. Estos análisis pueden resultar muy útiles para convencer a los directivos de que utilicen revisiones en el trabajo diario de los equipos, demostrando su eficacia y que dan lugar a una reducción del coste general de producción y mantenimiento del software.

Ejemplo Como parte de las técnicas de prueba estáticas, la organización ABC utiliza dos tipos de revisiones: recorridos e inspecciones. Los tutoriales se utilizan con fines de verificación. Se llevan a cabo para adquirir conocimientos sobre diversos aspectos del producto del trabajo. Un objetivo secundario de los tutoriales es:

- Lograr una visión de producto consistente y unificada dentro de la organización • Obtener consenso entre las partes interesadas sobre características específicas del producto •
- Obtener acuerdo sobre las técnicas de ingeniería que se utilizarán • Determinar la integridad y corrección de las capacidades y características de el software que se está desarrollando

Las revisiones son realizadas por los autores de cada producto de trabajo. Dentro de cada actividad del ciclo de desarrollo, se pueden realizar varias revisiones del mismo producto de trabajo. La única cantidad medida es el número de recorridos realizados.

Se realizan inspecciones para cumplir con los requisitos del equipo responsable de la gestión de calidad. Este equipo verifica que el producto del trabajo cumpla con los estándares aplicables que la organización debe seguir. Al final de cada actividad del ciclo de desarrollo se llevan a cabo inspecciones formales, las llamadas puertas de calidad, durante las cuales se verifican criterios de salida específicos para una fase de desarrollo determinada.

Dado que la inspección es un proceso formal, sigue un proceso bien definido que se muestra en la figura 3.7.

La inspección verifica estrictamente el producto del trabajo para:

- Integridad •
- Corrección •
- Coherencia •
- Estilo •
- Normas de construcción

La inspección la lleva a cabo el moderador y, además, en el proceso participan un escriba, de dos a cinco revisores y el autor. La realización de una inspección se trata como una verificación formal de los criterios de salida de una fase. Durante la inspección, se toman mediciones del producto y del proceso, y se informa toda la sesión en

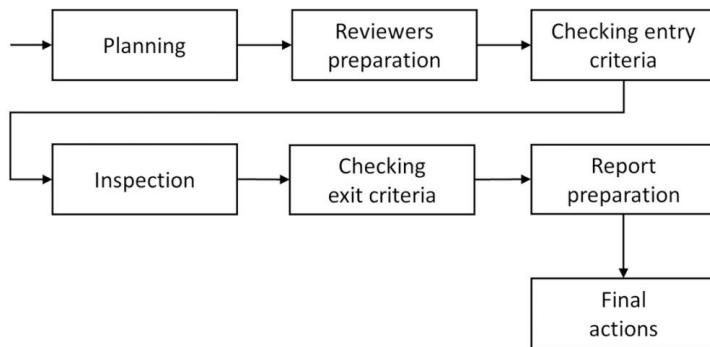


Fig. 3.7 Proceso de inspección en una organización ABC

Tabla 3.5 Comparación de recorridos e inspecciones utilizados en la organización ABC

Categoría	Tutorial	Inspección
Objetivo general	Hacer el trabajo correcto	hacer bien el trabajo
Específico objetivo	Educación, comprensión, consenso	Detección de defectos, cumplimiento comprobación
Desencadenar	solicitud del autor	Criterios de salida de fase
Medición	Instancias de tutorial	Mediciones de productos y procesos.

plantillas de informes definidas específicamente para este fin. Defectos encontrados durante el Las inspecciones se rastrean en la herramienta de gestión de defectos hasta que se cierran.

La Tabla 3.5 resume las diferencias entre recorridos e inspecciones.

Para obtener más información sobre revisiones, consulte [34, 38, 39].

Concluiremos esta sección con dos ejemplos de lo que es una revisión de muestra del cómo podría verse la arquitectura del sistema.

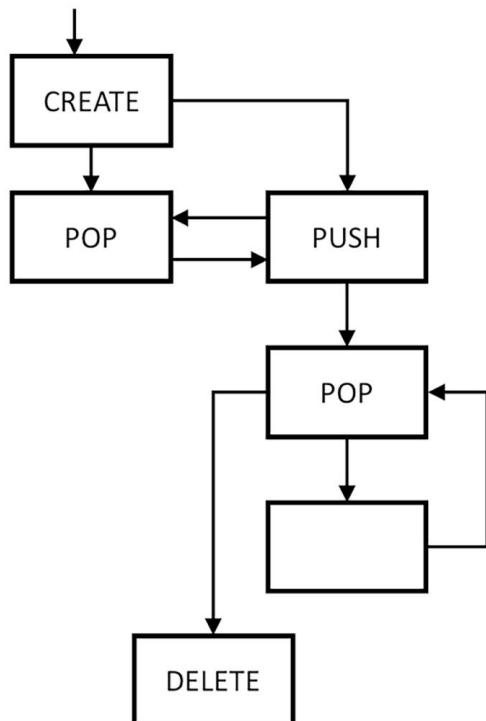
Ejemplo La Figura 3.8 muestra el diseño de un sistema de flujo de información en un Cierta estructura de datos utilizada en el sistema, llamada pila. Posibles operaciones en el pila son:

- CREATE: creación de una pila
- EMPUJAR: colocar un elemento encima de la pila
- POP: sacar un artículo de la parte superior de la pila
- ELIMINAR: elimina la estructura de pila del sistema.

El evaluador se encuentra actualmente en la fase de preparación individual. La lista de verificación utilizada por El probador consta de los siguientes elementos:

1. ¿Se realiza cada operación PUSH y POP solo después de que se haya creado la pila? (CREAR)?
2. ¿El sistema no permite la ejecución de POP en una pila vacía?

Fig. 3.8 Flujo del sistema de operaciones de pila



3. ¿El sistema prohíbe eliminar la pila (DELETE) cuando no está vacía?
4. ¿La cantidad de elementos de la pila nunca excederá los 10?

El evaluador utiliza una técnica de revisión basada en listas de verificación y verifica qué propiedades en la lista están satisfechos y cuáles no.

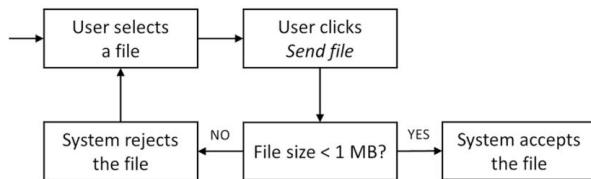
Se cumple el punto 1: cada instrucción PUSH y POP se crea después de la operación CREATE y no se puede ejecutar ninguna instrucción PUSH o POP después de la operación DELETE. Un evaluador curioso notará, sin embargo, que la especificación no dice si la pila está vacía cuando se crea o si contiene, por ejemplo, algunos elementos iniciales predeterminados. Vale la pena señalar al autor del documento que debe dejar esto claro en la especificación.

El punto 2 no se cumple. Después de la operación CREATE, la operación POP (en una pila vacía) se puede realizar inmediatamente.

Se cumple el punto 3. Después de cada operación PUSH, la siguiente operación debe ser una operación POP. Por lo tanto, solo se puede almacenar un elemento en la pila, que se eliminará de la pila en el siguiente paso. En este punto, el evaluador puede señalar que, dado que puede haber como máximo un elemento en la pila, ¿por qué usar una estructura de pila complicada cuando se podría usar una estructura más simple (como una variable o un registro) que nos permita almacenar solo uno? ¿Elemento?

Se cumple el punto 4; así se desprende del análisis del punto 3.

Fig. 3.9 Modelo de proceso de negocio



Como puede ver en el ejemplo anterior, el revisor no se limitó a "marcar" elementos individuales de la lista, sino que también notó algunas deficiencias en la especificación. Esta es una valiosa retroalimentación de revisión que puede servir para mejorar el producto del trabajo (especificación) de su autor.

Ejemplo Un arquitecto le ha pedido a un desarrollador que lleve a cabo una revisión informal por pares.

La revisión se refiere a un modelo de proceso creado por el arquitecto que implementa el siguiente requisito comercial: "Un usuario puede cargar cualquier archivo de menos de 1 GB a través de un formulario web. Si el tamaño del archivo es menor que este valor, el sistema lo acepta; de lo contrario, rechaza el archivo".

El modelo revisado se muestra en la Fig. 3.9.

El promotor recibió del arquitecto tanto un requisito como un diagrama que muestra el proceso. El desarrollador verifica que el diagrama corresponda al requisito y observa que hay un error tipográfico menor, pero muy significativo, al decidir si el sistema debe aceptar o rechazar el archivo: la condición dice "Tamaño de archivo < 1 MB" pero debería decir "Archivo tamaño < 1 GB". El desarrollador informa al arquitecto del error tipográfico observado y el arquitecto corrige el diseño.

3.2.5 Factores de éxito de las revisiones

La clave para una revisión exitosa es la selección cuidadosa del tipo de revisión y las técnicas de revisión utilizadas. Además, se deben tener en cuenta una serie de otros factores que pueden afectar el resultado. Los factores de éxito incluyen, entre otros, los siguientes factores:

- Cada revisión tiene objetivos claros definidos durante la planificación que pueden servir como criterios de salida
- Los tipos de revisiones utilizados son conducentes a lograr los objetivos establecidos y son apropiados para el tipo y nivel de los productos de trabajo de software y para los participantes.
- Se utilizan varias técnicas de revisión (como la revisión basada en listas de verificación o basada en roles). Se utiliza para identificar eficazmente los defectos presentes en un producto de trabajo.
- Las listas de verificación utilizadas están actualizadas y abordan los principales riesgos. • Los documentos grandes se escriben y revisan en lotes, de modo que los autores obtengan comentarios rápidos y frecuentes sobre los defectos (lo cual es parte del control de calidad).

- Los participantes tienen tiempo suficiente para prepararse para la revisión; las revisiones están programadas por adelantado.
- Los autores reciben comentarios de los revisores, para que puedan mejorar el producto de su trabajo y la calidad de su trabajo. • La dirección apoya el proceso de revisión (por ejemplo, designando tiempo suficiente para revisar las actividades en el cronograma del proyecto).
- Las revisiones se consideran una parte natural de la cultura organizacional para promover aprendizaje y mejora de productos y procesos.
- La revisión involucra a personas cuya participación es propicia para lograr sus objetivos; por ejemplo, personas con diferentes habilidades o puntos de vista que potencialmente utilizarán el documento en el curso de su trabajo. • Los evaluadores son reconocidos como participantes importantes en la revisión y el conocimiento que obtienen sobre el producto del trabajo les permite preparar pruebas más efectivas con anticipación.
- Los participantes dedican tiempo suficiente para participar en la revisión y muestran la debida atención a los detalles.
- Las revisiones se realizan en piezas pequeñas para que los revisores no pierdan la concentración durante la revisión individual y/o la reunión de revisión (si se lleva a cabo). • Los defectos detectados se reconocen, confirman y tratan objetivamente. • Las reuniones de revisión se moderan de la manera correcta para que los participantes no desperdicien tiempo en actividades innecesarias.
- La revisión se lleva a cabo en una atmósfera de confianza mutua y sus resultados no son utilizado para evaluar a los participantes.
- Los participantes evitan gestos y comportamientos que puedan indicar aburrimiento, irritación u hostilidad hacia otros participantes. • Se brindó la debida capacitación a los participantes, especialmente para los tipos más formales de revisiones (como inspecciones).
- Se ha creado un ambiente propicio para ampliar el conocimiento y mejorar los procesos. sido creado.

Ejemplo Se le pidió a un líder de revisión que organizara una revisión del código del componente X (en forma de recorrido) para un grupo de desarrolladores. El líder de revisión realizó la inspección e invitó a participar al autor del componente X y a un equipo de evaluadores.

Este escenario carecía de factores de éxito tanto organizativos como relacionados con las personas:

- Se pidió al líder de revisión que organizara un recorrido, pero organizó una inspección, que no es el mejor tipo de revisión para una revisión de código. Por lo tanto, no se cumplió el factor de éxito organizacional (elegir el tipo de revisión apropiado para lograr los objetivos dados y para adaptarse al tipo de producto de trabajo, los participantes de la revisión, las necesidades y el contexto del proyecto).
- Se pidió al líder de revisión que organizara una revisión de código para desarrolladores, pero, a excepción del autor, sólo invitó a evaluadores. Una revisión de código de este tipo será ineficaz. Por tanto, no se ha cumplido el factor de éxito de carácter personal (la revisión involucra a personas cuya participación favorece el logro de sus objetivos).

3.2.6 (*) Técnicas de revisión

El programa de estudios de Foundation Level en la versión anterior (3.1) describía cinco técnicas diferentes que un revisor puede utilizar como parte de una actividad de revisión individual, es decir, preparación individual para la revisión. Por supuesto, esta actividad puede realizarse como parte de todos los tipos de revisión, en particular los cuatro tipos de revisión descritos en la Sección. [3.2.3.](#)

El propósito de estas técnicas es detectar defectos en el producto de trabajo revisado.

Las cinco técnicas mencionadas son:

- Revisión ad hoc •

Revisión basada en listas de

verificación • Escenarios y
simulacros • Revisión

basada en roles • Lectura basada en perspectivas

Revisión ad hoc

Este es el enfoque tradicional para la detección de defectos por parte de los revisores. El proceso está completamente desestructurado. Cada revisor tiene la tarea de detectar tantos defectos de todos los tipos posibles como sea posible. La eficacia de dicha revisión depende en gran medida de las habilidades de los revisores individuales. Generalmente conduce a la detección de los mismos problemas (generalmente obvios o triviales) por parte de muchos revisores diferentes.

Durante una revisión ad hoc, los revisores reciben poca (o ninguna) orientación sobre cómo realizar la tarea. Los participantes suelen leer el producto del trabajo de forma secuencial, identificando y documentando los problemas encontrados sobre la marcha.

Revisión basada en listas de

verificación Una revisión basada en listas de verificación es una técnica más estructurada que una revisión ad hoc. Una buena práctica para este tipo de revisión es que diferentes revisores reciban listas de verificación diferentes. Esto aumentará la cobertura potencial del producto y detectará más defectos. También reduce el riesgo de que más personas detecten el mismo problema varias veces, lo que supone una pérdida de tiempo y recursos. La ventaja más importante de la técnica basada en listas de verificación es la cobertura sistemática de los tipos de defectos más comunes.

Con este enfoque existe el riesgo de que los revisores se limiten estrictamente a las cuestiones de la lista de verificación e ignoren otros problemas potenciales en el producto de trabajo que se está revisando. Por lo tanto, los revisores deben ser conscientes de que tienen más responsabilidad que simplemente seguir ciegamente los elementos de la lista de verificación.

Las listas de verificación deben seleccionarse según el tipo de producto del trabajo y el propósito del equipo. Por lo tanto, una lista de verificación para el uso de una revisión de requisitos será bastante diferente de, por ejemplo, una lista de verificación para el uso de pruebas de seguridad o pruebas de usabilidad de la interfaz. La lista de verificación puede incluso ser específica del método particular utilizado para producir el producto del trabajo. Por ejemplo, una lista de verificación utilizada para probar productos relacionados con la banca puede incluir regulaciones legales impuestas a los bancos, mientras que una utilizada en la industria automotriz puede, a su vez, basarse en la norma ISO 26262 [40].

Un problema común que surge al utilizar este enfoque es que las listas de verificación se vuelven cada vez más largas con el tiempo y, por lo tanto, menos prácticas de usar. un típico

La lista de verificación no debe contener más de aprox. 10 elementos y deben revisarse y actualizarse periódicamente. Las actualizaciones deben abordar especialmente aquellos problemas, fallos y defectos que nosotros mismos hemos detectado en nuestro producto. Según nuestra experiencia, una lista de verificación de este tipo será mucho mejor (en términos de eficiencia de detección de defectos) que, por ejemplo, una lista de verificación estándar que se encuentra en Internet.

Un enfoque común es utilizar el análisis de riesgos y ampliar la lista de verificación para incluir los riesgos identificados con el nivel de gravedad más alto. Esto permite comprobar directamente los mayores riesgos cuando se utiliza un enfoque de lista de verificación.

En la sección 1 se presenta un ejemplo de revisión basada en una lista de verificación. [3.2.4](#) (un ejemplo de pila).

Escenarios y simulacros El

enfoque basado en escenarios funciona bien cuando los requisitos, el diseño o las pruebas mismas están documentados en un formato de "escenario" apropiado, como los casos de uso. En este enfoque, los revisores realizan los llamados ensayos en seco del producto de trabajo, verificando que la funcionalidad del producto se describa correctamente y que las excepciones comunes causadas por el mal comportamiento del producto se manejen adecuadamente.

Existe el riesgo de que los espectadores sigan demasiado de cerca los escenarios definidos. En tal situación, pueden pasar fácilmente por alto algunos defectos, como la falta de funcionalidad en el producto.

Al igual que con el enfoque de lista de verificación, los escenarios se pueden ampliar para incluir aspectos que surjan del análisis de riesgos para garantizar que los escenarios más importantes y utilizados con más frecuencia se exploren más a fondo.

Revisión basada en

roles La revisión basada en roles es una técnica en la que los revisores evalúan un producto de trabajo desde la perspectiva de los roles particulares de las partes interesadas. Los roles típicos incluyen tipos específicos de usuarios finales (experimentados, inexpertos, ancianos, niños, etc.) o roles específicos en la organización (usuario, administrador, administrador del sistema, evaluador de rendimiento, etc.).

A menudo, los roles se modelan mediante la llamada persona. Una persona es un personaje ficticio pero concreto que simboliza un determinado tipo de usuario. Al hacer que esta persona sea concreta (especificando, por ejemplo, su género, edad e intereses), es más fácil para el equipo "entrar en" el tipo de usuario. En la figura 3.10 se muestra un ejemplo de persona .

Lectura basada en perspectiva La

técnica de lectura basada en perspectiva se describe en la literatura como uno de los métodos de revisión individual más eficaces [41]. Se basa en el hecho de que diferentes revisores adoptan diferentes perspectivas de las partes interesadas al revisar un producto de trabajo y revisan el producto desde ese ángulo. Al considerar múltiples vistas del producto, los revisores pueden descubrir una mayor cantidad de problemas potenciales. Al mismo tiempo, al asignar perspectivas específicas a los revisores, cada revisor puede revisar el producto a fondo y en detalle, y dado que cada uno corresponde a una perspectiva diferente, el riesgo de detectar defectos duplicados es bajo.

La lectura basada en perspectiva no se limita sólo a adoptar diferentes puntos de vista. También requiere que los revisores intenten utilizar el producto de trabajo bajo revisión para generar un primer prototipo del producto para ver si es posible hacerlo con base en la información.

**Benjamin Thompson**

Position:	Mobile applications tester
Age:	28 yo
City:	New York
Hobbies:	movies, FPS games, travelling, new technologies

Character traits: vigorous, impatient, curious

Fig. 3.10 Ejemplo de persona

disponible en el producto de trabajo revisado. Por ejemplo, los evaluadores intentarán generar versiones preliminares de las pruebas de aceptación si realizan una revisión basada en la perspectiva de la especificación de requisitos para verificar que la especificación contenga toda la información necesaria. Además, la lectura basada en la perspectiva suele utilizar listas de verificación.

Las perspectivas típicas que podría adoptar un espectador son:

- Usuario
- Analista de negocios •
- Diseñador •
- Tester •
- Administrador de sistemas •
- Gerente de marketing y promociones • Ingeniero de soporte técnico

Si la técnica se limita al punto de vista del usuario final, también se puede utilizar una técnica de lectura basada en perspectiva, distinguiendo entre tipos de usuarios del sistema, como por ejemplo:

- Usuario final •
- Administrador del sistema •
- Ingeniero de soporte técnico

Un factor clave para el éxito de la lectura basada en perspectivas es tener en cuenta y equilibrar los puntos de vista de las distintas partes interesadas de una manera que sea adecuada al riesgo. Los puntos de vista que adoptemos deberían depender del contexto. Por ejemplo:

- Si el documento que se revisa son requisitos, las perspectivas típicas serán usuario, diseñador y probador.
- Si el sistema está relacionado con un área altamente regulada (por ejemplo, aviación, banca, sistemas médicos), el punto de vista del regulador definitivamente debe incluirse en la revisión.
- Si el sistema se va a utilizar durante un largo tiempo, se debe adoptar el punto de vista del equipo de mantenimiento del sistema.

Revisión basada en roles versus lectura basada en perspectivas La lectura basada en perspectivas es similar a la técnica de revisión basada en roles discutida anteriormente. Los dos enfoques a veces se equiparan en la literatura, pero difieren. La diferencia tal vez pueda explicarse de la forma más sencilla como sigue:

- En una revisión basada en roles, los "tipos" de usuarios cambian, pero las tareas a realizar siguen siendo los mismos (por ejemplo, diferentes tipos de clientes bancarios).
- En la lectura basada en perspectivas, las "perspectivas" de las partes interesadas o, en el caso de los propios usuarios, los tipos de tareas a realizar (por ejemplo, usuario final, administrador, analista de negocios, evaluador) cambian.

Ejemplo El equipo utiliza una técnica de lectura basada en perspectiva. La persona que revisa el documento (la especificación de requisitos) adopta la perspectiva del evaluador y utiliza el siguiente procedimiento para leer la especificación.

PROCEDIMIENTO DE LECTURA. Para cada requisito, cree una prueba o un conjunto de pruebas para garantizar que la implementación cumpla con el requisito. Utilice un enfoque de prueba estándar y criterios y técnicas de prueba para crear el conjunto de pruebas. Al crear pruebas para cada requisito, responda las siguientes preguntas:

1. ¿Tiene suficiente información para identificar el elemento de prueba y los criterios de prueba?
¿Puede crear casos de prueba razonables para cada elemento según estos criterios?
2. ¿Existe otro requisito para el cual podría generar un caso de prueba similar, pero
¿Con un resultado esperado diferente?
3. ¿Está seguro de que la prueba que generó proporcionará los valores correctos en el momento correcto?
¿unidades?
4. ¿Existen otras interpretaciones de este requisito que un desarrollador podría adoptar, según cómo se define el
requisito? ¿Afectaría sus pruebas?
5. ¿Es el requisito razonable y racional en términos de su conocimiento de la aplicación y de lo que se incluye en
la descripción general del sistema?

Preguntas de muestra

Pregunta 3.1

(FL-3.1.1, K1)

Su organización acaba de preparar un documento oficial que describe cómo las revisiones debe realizarse en la organización.

¿Se puede revisar este documento?

- R. Sí, porque cualquier documento comprensible para humanos puede someterse a pruebas estáticas.
- B. No, porque tendríamos que aplicarnos las reglas descritas en este documento durante la revisión.
- C. No, porque el documento no es un producto del trabajo ni del proceso de prueba ni del el proceso de desarrollo.
- D. No, porque solo se pueden realizar revisiones de especificaciones y código fuente.

Elija una respuesta.

Pregunta 3.2

(FL-3.1.2, K2)

Mientras analizaba el código, el evaluador notó que la complejidad ciclomática de uno de los componentes del código era muy alta. El evaluador pasó esta información a los desarrolladores, quienes refactorizaron el código, haciéndolo más legible y comprobable. Este escenario muestra el beneficio de utilizar:

- A. Pruebas dinámicas.
- B. Pruebas estáticas.
- C. Gestión de pruebas.
- D. Técnica de prueba formal.

Elija una respuesta.

Pregunta 3.3

(FL-3.1.3, K2)

¿Cuál es la diferencia entre técnicas estáticas y dinámicas, según el propósito de estas técnicas?

- R. Las técnicas estáticas detectan fallas directamente, mientras que las técnicas dinámicas detectan directamente detectar defectos.
- B. Las técnicas estáticas se suelen utilizar al principio del SDLC, mientras que las técnicas dinámicas Las técnicas se utilizan normalmente en fases posteriores del SDLC.
- C. No hay diferencia, ya que ambos tipos de técnicas tienen como objetivo detectar defectos lo antes posible. como sea posible.
- D. Las técnicas estáticas generalmente requieren habilidades de programación, mientras que las técnicas dinámicas normalmente no lo hago.

Elija una respuesta.

Pregunta 3.4

(FL-3.2.1, K1)

Considera las siguientes afirmaciones sobre la retroalimentación temprana.

- i. La retroalimentación temprana brinda a los desarrolladores más tiempo para producir nuevas funciones del sistema, ya que dedican menos tiempo a modificar las funciones planificadas en una iteración determinada.
- ii. La retroalimentación temprana permite a los equipos ágiles ofrecer primero las funciones con mayor valor comercial, ya que la atención del cliente permanece centrada en las funciones más importantes desde el punto de vista del cliente. III. La retroalimentación temprana reduce el costo general de las pruebas porque reduce la cantidad de tiempo que los evaluadores necesitan para probar el sistema. IV. La retroalimentación temprana aumenta la probabilidad de que el sistema producido se acerque a las expectativas de los clientes, ya que tienen la oportunidad de realizar cambios durante cada iteración.

¿Cuáles de estas afirmaciones son ciertas?

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

Elija una respuesta.

Pregunta 3.5

(FL-3.2.2, K2)

¿Cuál de las siguientes es parte de la fase de inicio de revisión?

- A. Seleccionar quién participará en la revisión.
- B. Recopilación de métricas.
- C. Responder a las preguntas de los participantes sobre el alcance, los objetivos, el proceso, las funciones y el trabajo. productos.
- D. Identificar el alcance del trabajo, incluido el tipo de revisión y los documentos (o partes de los mismos) que son objeto de la revisión y las características de calidad a evaluar.

Elija una respuesta.

Pregunta 3.6

(FL-3.2.3, K1)

¿Qué función es responsable del buen funcionamiento de la reunión de revisión?

- A. Líder de revisión.
- B. Moderador.
- C. Autor.
- D. Revisor.

Elija una respuesta.

Pregunta 3.7

(FL-3.2.4, K2)

Durante los últimos días, el equipo ha estado intentando encontrar la causa de un extraño fallo del sistema. Como no pudieron descubrir la causa, el equipo decidió "simular la computadora" ejecutando manualmente el código línea por línea para comprender qué estaba haciendo exactamente el programa y así descubrir la causa del extraño comportamiento del software.

¿Qué tipo de revisión será la más apropiada para esta actividad?

- A. Inspección.
- B. Revisión informal.
- C. Revisión técnica.
- D. Tutorial.

Elija una respuesta.

Pregunta 3.8

(FL-3.2.5, K1)

La inspección se realizará mejor si:

- R. Su objetivo principal se definirá como “evaluación de alternativas”.
- B. El gerente asistirá a las reuniones de revisión.
- C. Los revisores estarán capacitados para realizar revisiones.
- D. No se recopilarán métricas durante el proceso de revisión.

Elija una respuesta.

Capítulo 4 Análisis y diseño de pruebas



Palabras clave

Criterios de aceptación	Los criterios que un componente o sistema debe satisfacer para ser aceptado por un usuario, cliente u otra entidad autorizada. Referencias: ISO 24765.
Desarrollo basado en pruebas de aceptación: un enfoque de prueba primero basado en la colaboración que define las pruebas de aceptación en el lenguaje de dominio de las partes interesadas.	Abreviatura: ATDD. una
Técnica de prueba de caja negra	técnica de prueba basada en un análisis de la especificación de un componente o sistema. Sinónimos: técnica de diseño de pruebas de caja negra, técnica basada en especificaciones. una técnica de prueba de caja negra en la que los casos de prueba se diseñan en función de valores límite. la
Análisis de valor límite	una técnica de prueba basada en la experiencia en la que se diseñan casos de prueba para ejercitarse los elementos de una lista de verificación.
Cobertura de sucursales	cobertura de sucursales en un gráfico de flujo de control.
Pruebas basadas en listas de verificación	una técnica de prueba basada en la experiencia en la que se diseñan casos de prueba para ejercitarse los elementos de una lista de verificación.
Enfoque de prueba basado en la colaboración: un enfoque de prueba que se centra en evitar defectos mediante la colaboración entre las partes interesadas. el	grado en que los elementos de cobertura especificados son ejercidos por un conjunto de pruebas, expresado como porcentaje. Sinónimos: cobertura de prueba.
Cobertura	

Artículo de cobertura	un atributo o combinación de atributos derivado de una o más condiciones de prueba mediante el uso de una técnica de prueba.
Prueba de tabla de decisiones	una técnica de prueba de caja negra en la que se prueba Los casos están diseñados para ejercer la combinaciones de condiciones y acciones resultantes mostradas en una decisión mesa.
Partición de equivalencia	una técnica de prueba de caja negra en la que se prueba las condiciones son particiones de equivalencia ejercido por un miembro representativo de cada partición. Según ISO 29119-1. Sinónimos: prueba de partición.
Error al adivinar	una técnica de prueba en la que se derivan pruebas sobre la base del conocimiento del evaluador sobre fracasos pasados o conocimiento general de modos de fallo. Referencias: ISO 29119-1.
Técnica de prueba basada en la experiencia	una técnica de prueba basada en la experiencia del probador. experiencia, conocimiento e intuición. Sinónimos: diseño de pruebas basado en la experiencia técnica, técnica basada en la experiencia.
Prueba exploratoria	un enfoque de prueba en el que los evaluadores diseñar y ejecutar pruebas dinámicamente basándose en su conocimiento, exploración de el elemento de prueba y los resultados de pruebas anteriores. pruebas. Según ISO 29119-1.
Pruebas de transición estatal	una técnica de prueba de caja negra en la que se prueba Los casos están diseñados para ejercitarse elementos de Un modelo de transición estatal. Sinónimos: finito pruebas estatales.
Cobertura de declaración	la cobertura de declaraciones ejecutables.
Técnica de prueba	un procedimiento utilizado para definir las condiciones de prueba, diseñar casos de prueba y especificar datos de prueba. Sinónimos: técnica de diseño de pruebas.
Técnica de prueba de caja blanca	una técnica de prueba basada únicamente en el interior estructura de un componente o sistema. Sinónimos: diseño de prueba de caja blanca técnica, técnica basada en estructuras.

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.

En esta sección, describimos:

- Cómo elegir la técnica de prueba adecuada •

Qué tipos de técnicas de prueba se describen en el programa de estudios del nivel básico •

¿Cuáles son las características comunes de cada grupo de técnicas de prueba?

Según la definición del diccionario Merriam-Webster, una técnica (del gr. *technē* (τέχνη)—arte, artesanía, maestría, habilidad) es “un conjunto de métodos técnicos (como en un oficio o en una investigación científica)” o “un método para lograr un **objetivo** deseado”.

En el contexto de las pruebas de software, esta actividad consistirá en utilizar ciertos métodos para derivar (producir) condiciones de prueba, casos de prueba y datos de prueba basados en la base científica del conocimiento del software.

Digamos de inmediato que cada técnica de prueba.  está estrictamente definido, es decir, es formal en el sentido de los principios que lo constituyen (aunque en sí mismo, como tal, formal no necesita serlo, por ejemplo, conjeturas erróneas o pruebas exploratorias). Esto se debe a que detrás de cada técnica de prueba hay lo que Glenford Myers llama una “hipótesis de error”. [10]. En el contexto del vocabulario ISTQB® [42], tal vez un nombre más apropiado sería “hipótesis del defecto”, pero por razones históricas, nos quedaremos con el nombre original (de todos modos, las técnicas de prueba también detectan defectos debidos a errores de programación específicos).).

Esta hipótesis del error, o, como a veces se la llama, teoría del error, es la base científica para la construcción de cada técnica. Al analizar cada uno de ellos en las siguientes subsecciones, prestaremos especial atención a qué tipos de problemas es capaz de detectar la técnica. Por lo tanto, las técnicas de prueba son independientes de las tecnologías específicas, las herramientas utilizadas o el tipo de software bajo prueba. Cada técnica se basa en un problema abstracto relacionado con una hipótesis de error específica y está diseñada para detectar ese tipo específico de error o defecto.

El programa de estudios de Foundation Level analiza nueve técnicas de prueba y las agrupa en tres categorías:

- Técnicas de prueba de caja negra (cuatro técnicas)
- Técnicas de prueba de caja blanca (dos técnicas) •

Técnicas de prueba basadas en la experiencia (tres técnicas)

Esta división se realiza en función de la fuente de conocimiento sobre el objeto de prueba. La figura 4.1 justifica esquemáticamente la razón de tal división de técnicas.

Antes de desarrollar un software, hay que diseñarlo. La función de la documentación de diseño puede cumplirse mediante especificaciones de requisitos, casos de uso o procesos comerciales.

¹<https://www.merriam-webster.com/dictionary/technique>

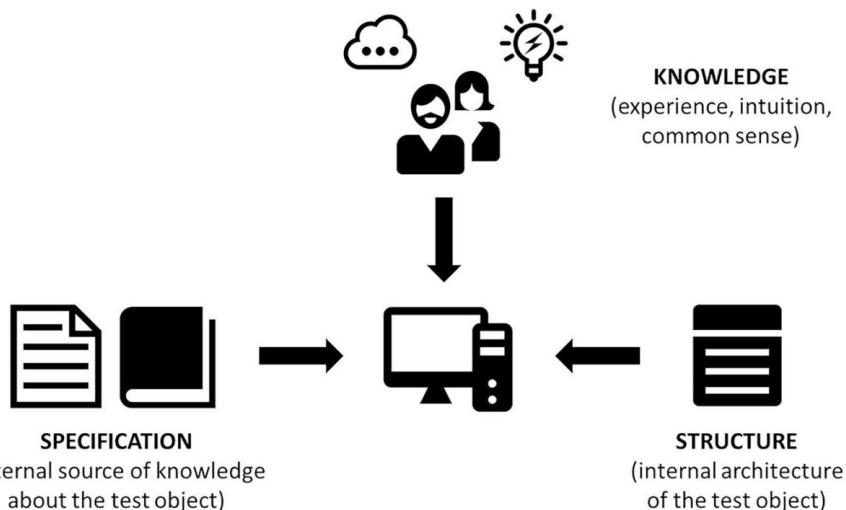


Fig. 4.1 Fuentes de conocimiento sobre el objeto de prueba como base para la categorización de técnicas de prueba

descripción. A partir de ellos se crea un software que tiene una estructura específica. Es, por ejemplo, código, estructura de menú, estructura de flujo de procesos de negocio u otra forma de descripción de la arquitectura. Además de estas fuentes formales de conocimiento, también existen otras menos formales, relacionadas directamente con las personas que trabajan en el desarrollo de software. Estas personas tienen conocimientos, experiencia e intuición, de las que también se pueden derivar pruebas valiosas.

Ahora discutiremos las diferentes categorías de técnicas de prueba con más detalle.

Técnicas de prueba de caja negra

Las técnicas de prueba de caja negra también se conocen como técnicas de comportamiento o técnicas basadas en especificaciones. Utilizan conocimientos externos al objeto de prueba sobre cómo debería comportarse. Este conocimiento puede ser, por ejemplo, especificación de requisitos, descripción de casos de uso, historias de usuarios (en proyectos ágiles) y procesos de negocio. Todos estos modelos describen el comportamiento deseado del sistema, tanto funcional como no funcional, sin hacer referencia a su diseño interno.

La ventaja de utilizar técnicas de caja negra es que los documentos antes mencionados generalmente existen mucho antes de que comience la implementación de un componente o sistema. Esto significa que las actividades de prueba (por ejemplo, análisis y diseño de pruebas) pueden comenzar mucho antes del desarrollo del código. Esto nos permite utilizar técnicas de prueba de caja negra no solo durante las pruebas dinámicas sino también en la etapa de diseño de las pruebas, como una especie de método de prueba estático. Las técnicas de caja negra se pueden utilizar tanto en pruebas funcionales como no funcionales.

Técnicas de prueba de caja blanca

Las técnicas de prueba de caja blanca también se denominan técnicas estructuradas o basadas en estructura. La base del diseño de pruebas de caja blanca es la estructura interna del objeto de prueba.

Muy a menudo, esta estructura es el código fuente, pero también puede ser otro modelo de más alto nivel del sistema o arquitectura del módulo. Por ejemplo, en el nivel de prueba de integración, dicho modelo puede ser el llamado gráfico de llamadas, y en el nivel de prueba del sistema, el flujo de información en el proceso de negocio.

Un lector atento puede notar que dado que, cuando probamos un programa, nos referimos al programa mismo (por ejemplo, al código fuente), llegamos a una situación paradójica en la que el programa se convierte en su propio oráculo, es decir, decide por sí mismo cómo debería comportarse. De hecho, este es un problema aparente. El conocimiento de la estructura interna del programa se utiliza sólo para diseñar pruebas que cubran elementos específicos de este modelo (por ejemplo, declaraciones de código, decisiones en el código, rutas en el programa). Por el contrario, para cada prueba, su resultado esperado debe determinarse sobre la base de conocimientos externos al sistema bajo prueba, por ejemplo, sobre la base de especificaciones o sentido común.

El código nunca puede ser un oráculo en sí mismo.

Técnicas de prueba basadas en la experiencia El

grupo de técnicas de prueba basadas en la experiencia se diferencia de los otros dos grupos en que no se basa en ningún documento formal: diseño, requisitos, código, etc. Esto se debe a que las técnicas de prueba basadas en la experiencia utilizan algo más "fuentes blandas de conocimiento sobre el sistema bajo prueba. Estas fuentes son conocimiento, intuición, experiencia, conocimiento de defectos encontrados en versiones anteriores del sistema o aplicaciones similares, etc. Por lo tanto, se refieren directamente a las cualidades y habilidades de los propios evaluadores, en lugar de conocimiento "objetivo" sobre el sistema bajo prueba.

Las técnicas pueden aprovechar no sólo la experiencia de los evaluadores sino también de todas las demás partes interesadas del proyecto. Los ejemplos incluyen el uso de la experiencia de desarrolladores, usuarios finales, clientes, arquitectos, analistas de negocios y gerentes de proyectos. Las técnicas de prueba basadas en la experiencia se utilizan a menudo en combinación con técnicas de prueba de caja negra y caja blanca. Esto brinda a los evaluadores la oportunidad de detectar problemas que fácilmente se pasan por alto mediante el uso de técnicas de prueba más formales, que no pueden tener en cuenta todos los matices del proyecto o el contexto en el que se desarrolla el software.

La Tabla 4.1 muestra las diferencias básicas entre los tres grupos de técnicas de prueba. mencionado anteriormente.

Como se mencionó anteriormente, el programa de estudios de Foundation Level introduce nueve técnicas de prueba, cuyo conocimiento es obligatorio para el examen. Éstas incluyen cuatro técnicas de caja negra, dos técnicas de caja blanca y tres técnicas basadas en la experiencia. En la figura 4.2 se muestra un resumen de estas técnicas. El programa de estudios no describe estas técnicas en detalle ni da ejemplos de su uso práctico, aunque la capacidad de aplicarlas correctamente es una de las cualidades más importantes de un buen evaluador. Por lo tanto, en las siguientes subsecciones las discutiremos con gran detalle, destacando los aspectos importantes de cada técnica. Para hacer el material más accesible, ilustraremos el uso de estas técnicas con muchos ejemplos.

Por supuesto, existen muchas otras técnicas de prueba. El programa de estudios analiza sólo los los más populares y utilizados. Las técnicas de prueba y las medidas de cobertura correspondientes se describen en la norma internacional ISO/IEC/IEEE 29119-4 [4]. También se proporciona más información sobre técnicas de prueba en [16, 43–48].

Tabla 4.1 Comparación de categorías de técnicas de prueba

Criterio de comparación	Caja negra	Caja blanca	Basado en la experiencia
Fuente para derivar condiciones de prueba, casos de prueba y datos de prueba.	Bases de prueba externas al objeto de prueba: requisitos, especificaciones, casos de uso, historias de usuarios	Base de prueba que describe la estructura interna del objeto de prueba: código, arquitectura, diseño detallado; Las especificaciones se utilizan a menudo para describir los resultados esperados.	Conocimiento, experiencia, intuición de los evaluadores, desarrolladores, usuarios y otras partes interesadas.
Tipo de problemas detectados	Discrepancias entre los requisitos (comportamiento declarado) y su implementación.	Problemas asociados con el flujo de control o el flujo de datos.	Depende de la persona que realiza las pruebas o, por ejemplo, de la lista de verificación utilizada en las pruebas
Cobertura	Medido contra los elementos probados de la base de prueba y la técnica aplicada a la base de prueba.	Medido contra elementos probados de la estructura, como código o interfaces.	No definida

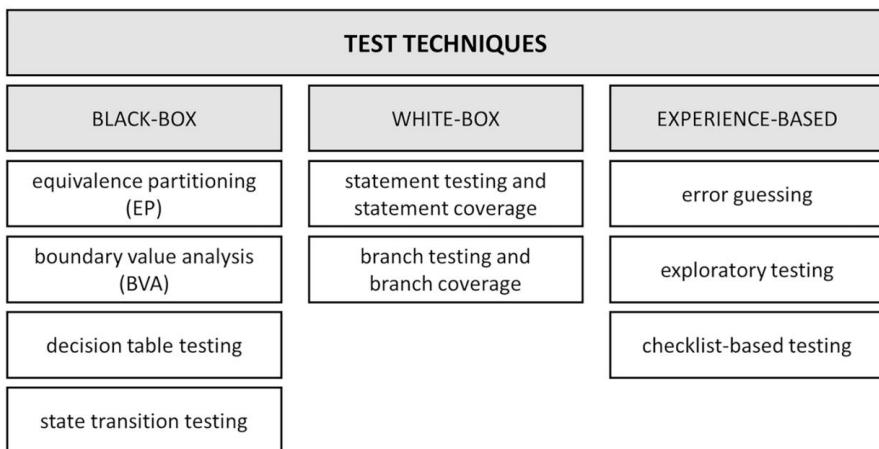


Fig. 4.2 Técnicas de evaluación descritas en el programa de estudios del nivel básico

Selección de la técnica de prueba

Muchos factores influyen en la elección de las técnicas de prueba. Estos se pueden dividir en tres grupos principales:

- Factores formales (por ejemplo, documentación, leyes y regulaciones vigentes, disposiciones contractuales con el cliente, procesos implementados en la organización, objetivos de prueba, modelo SDLC utilizado).

- Factores del producto (por ejemplo, software, su complejidad, importancia de diversas características de calidad, riesgos, tipos esperados de defectos, uso esperado del software). • Factores del proyecto (por ejemplo, tiempo disponible, presupuesto, recursos, herramientas, habilidades, conocimientos). y experiencia de los probadores).

Ejemplo Está trabajando en un proyecto que implica el desarrollo de una aplicación de biblioteca universitaria. En su organización, existe la obligación de probar las aplicaciones que se están desarrollando, mientras que la estrategia de prueba (ver Sección 5.1.1) implementada en su proyecto impone, debido a los contratos firmados con el cliente, la necesidad de realizar pruebas utilizando pruebas formales. técnicas para que el diseño, implementación, ejecución y resultados de las pruebas puedan documentarse. El proyecto se lleva a cabo bajo el modelo V.

Los requisitos se han recopilado y presentado en forma de especificación de requisitos. Actualmente, los arquitectos del sistema están trabajando en el diseño de la lógica de negocios del módulo responsable de verificar cuántos libros puede pedir prestado un usuario según este tipo de usuario (estudiante, empleado) y multas vencidas. Este es un componente clave para garantizar la adecuada implementación de las reglas y regulaciones de la biblioteca, por lo que el impacto del riesgo de su mal funcionamiento es muy alto.

Tienes 3 días para analizar y diseñar pruebas manuales. No se espera la automatización de pruebas debido a que las pruebas de regresión no desempeñarán un papel importante en este proyecto. Además, la organización no cuenta con herramientas especializadas ni las habilidades técnicas para crear scripts de prueba automatizados.

De la descripción anterior se desprende claramente que en el contexto del componente responsable de aplicar las normas de préstamo de libros:

- Necesitamos realizar pruebas (esto se desprende directamente de la documentación y contrato).
- Necesitamos diseñar y documentar estas pruebas (ídem). • Las pruebas deben centrarse en la lógica empresarial, por lo que una opción sensata sería utilizar técnicas de prueba apropiadas para este aspecto del software.
- El impacto del riesgo asociado con este componente es alto, por lo que tiene sentido utilizar métodos que verifiquen la lógica empresarial con mucho cuidado. • Las pruebas se realizarán manualmente (falta de herramientas y experiencia, bajo papel de las pruebas de regresión, que suelen ser un candidato natural para la automatización).

Como puede verse en el ejemplo anterior, la decisión sobre la elección de la técnica, el nivel de detalle del análisis, la elección del diseño, implementación y ejecución de la prueba está influenciada por muchos factores formales, de producto y de proyecto.

No existe una técnica de prueba perfecta y universal. Cada técnica tiene sus propias ventajas y desventajas. Cada uno se desempeñará mejor en una situación y peor en otra. Por ejemplo, en el caso de la situación antes mencionada con un componente

Tabla 4.2 Ejemplos de técnicas de prueba con diferentes niveles de formalización

Grado de formalización	Ejemplo
Muy bajo	Ejecución no planificada e indocumentada de adivinación de errores, sin guardar los resultados de las pruebas.
Bajo	Realización de pruebas exploratorias con una carta de prueba.
Medio	Utilizar pruebas de tablas de decisiones para probar la lógica empresarial; Los casos de prueba se documentan en las especificaciones de casos de prueba y los resultados de las pruebas se registran.
Grande	Utilizar el modelo de máquina de estados y la técnica de prueba de transición de estados para probar el comportamiento del programa; El diseño de la prueba (máquina de estado), los casos de prueba (en forma de scripts de prueba) y los resultados de la prueba (registros) están documentados.

que verifica las reglas para el préstamo de libros en una biblioteca, una de las técnicas de prueba de caja negra (prueba de tabla de decisiones (ver Sección 4.2.3)) parece ser una buena opción, porque prueba la lógica de negocios y esta es la funcionalidad. Lo que más nos importa son las pruebas. Por otro lado, no tiene sentido utilizar, por ejemplo, técnicas de prueba de caja blanca que se basan en la estructura interna del programa (código fuente), ya que el problema de prueba aquí es la prueba de lógica de negocios (las técnicas antes mencionadas se analizan en detalle más adelante en este capítulo).

Una buena práctica, que suelen utilizar los evaluadores experimentados, es combinar técnicas. Por ejemplo, al utilizar tablas de decisión en el problema anterior, el evaluador puede en ocasiones aplicar la técnica de análisis de valores límite (consulte la Sección 4.2.2) y verificar las reglas comerciales para dichos valores (por ejemplo, el número máximo posible de libros que un estudiante puede pedir prestados).).

El uso de técnicas de prueba también puede considerarse en el contexto de los niveles de prueba. Algunas técnicas son más universales; por lo tanto, están naturalmente presentes en todos los niveles de prueba, desde las pruebas de componentes hasta las pruebas de aceptación. Algunas otras técnicas, por otra parte, tienen un ámbito de aplicación algo más limitado; por ejemplo, las técnicas de caja blanca se aplican con mayor frecuencia en el nivel de prueba de componentes (por ejemplo, pruebas unitarias de desarrollador). Por supuesto, se pueden imaginar ejemplos del uso de esta técnica a nivel de prueba de aceptación, pero en la práctica estas situaciones son bastante raras.

El grado de formalización del desarrollo de pruebas utilizando técnicas de prueba puede variar desde muy informal hasta muy formal. En el Cuadro 4.2 damos algunos ejemplos de los distintos grados de formalización.

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.

En esta sección, además de analizar las cuatro técnicas de prueba de caja negra descritas en el programa de estudios, analizamos una quinta técnica adicional: las pruebas basadas en casos de uso (Sección 4.2.5). Se trata de una materia optativa y no examinable. Esta técnica estaba presente en una versión anterior del programa de estudios. Creemos que es tan importante y tan utilizado que vale la pena discutirlo, incluso como contenido superobligatorio en relación al plan de estudios.

4.2.1 Partición de equivalencia (EP)

Uno de los siete principios de prueba presentados en la Sección. 1.3 dice que “las pruebas exhaustivas son imposibles”. Esto es bastante obvio: el número de combinaciones posibles de datos de entrada es prácticamente infinito, mientras que el evaluador tiene la capacidad de realizar sólo un número finito y muy pequeño de pruebas.

La partición de equivalencia  La técnica intenta superar el principio de imposibilidad de realizar pruebas exhaustivas. Generalmente hay un número infinito de entradas posibles, pero el número de comportamientos esperados de un programa en estas entradas suele ser finito, especialmente si consideramos comportamientos específicos relacionados con un aspecto estrictamente definido del funcionamiento de la aplicación. Veamos algunos ejemplos.

Ejemplo El sistema establece el monto del impuesto según los ingresos. El impuesto puede ser del 0%, 19%, 33% o 45%. Hay potencialmente infinitos valores posibles para el ingreso, pero sólo cuatro tipos posibles de decisiones tomadas por el sistema.

Ejemplo Un usuario completa un formulario web y luego quiere imprimirlo. La impresión puede ser por una o ambas caras. Hay formas potencialmente infinitas de completar un formulario, especialmente si es complejo. Sin embargo, lo que queremos probar en este caso son sólo dos tipos de comportamiento: impresión correcta a una cara e impresión correcta a dos caras.

Ejemplo El usuario completa el formulario, incluido el campo "edad". A continuación, el usuario podrá comprar un billete, siendo el procedimiento diferente en función de la edad del usuario. El diagrama de flujo exacto del proceso se muestra en la Fig. 4.3. Hay muchas posibilidades para llenar el formulario, pero sólo cuatro acciones posibles: mensaje de error, comprar entrada por un adulto, comprar entrada por un menor y cerrar sesión.

Como puede ver, normalmente es posible reducir el comportamiento de prueba de un programa a un número finito de variantes. El método de partición de equivalencia divide un dominio determinado en subconjuntos llamados particiones (o particiones de equivalencia) de manera que por cada dos elementos de una partición, tenemos un comportamiento de programa idéntico. Por ejemplo, si el sistema asigna un descuento a personas menores de 18 años, todos los números menores de 18 años formarán una partición de equivalencia, correspondiente a la asignación del descuento. De esta forma, desde el punto de vista del evaluador, los valores que pertenecen a la misma partición se tratan de la misma manera. Por lo tanto, cada elemento de una partición determinada es una opción igualmente buena para probar.

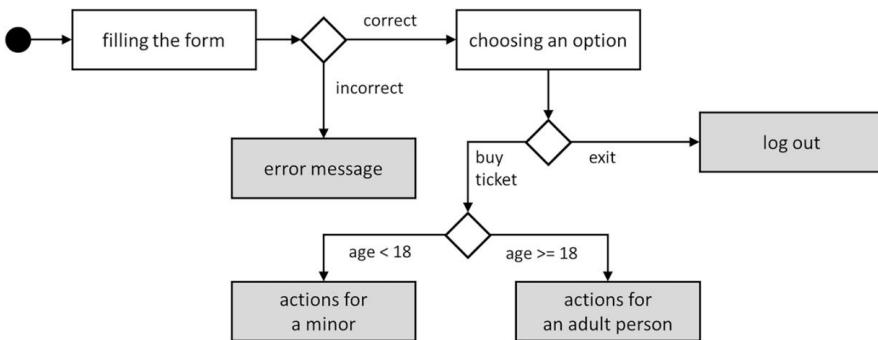


Fig. 4.3 Ejemplo de un proceso que describe posibles escenarios de uso del sistema

Aplicación La

técnica EP es versátil. Se puede utilizar prácticamente en cualquier situación, en cualquier nivel de prueba y en cualquier tipo de prueba. Esto se debe a que todo se reduce a la división de posibles datos en grupos. Vale la pena mencionar que la técnica se puede aplicar no sólo a dominios de entrada sino también a dominios de salida y dominios internos (es decir, aquellos relacionados con variables que no se dan directamente en la entrada ni se devuelven en la salida).

El dominio que dividimos en particiones de equivalencia no tiene por qué ser un dominio numérico. De hecho, puede ser cualquier conjunto no vacío. A continuación se muestran algunos ejemplos de dominios a los que se puede aplicar la técnica de partición de equivalencia:

- Un conjunto de números naturales (p. ej., partición en números pares e impares) •
- Una colección de palabras (p. ej., partición por longitud de palabra, una letra, dos letras, etc.) •
- Colecciones relacionadas con el tiempo (p. ej., partición por año de nacimiento, por mes en un determinado año, etc)
- Una colección de tipos de sistemas operativos: {Windows, Linux, macOS} (por ejemplo, partición en particiones de un solo elemento, {Windows}, {Linux}, {macOS})

Corrección de partición

Es muy importante que el particionamiento que hagamos sea correcto, lo que significa que:

- Cada elemento del dominio pertenece exactamente a una partición de equivalencia. •
- Ninguna partición de equivalencia está vacía.

La Figura 4.4 ilustra la cuestión de la corrección de la partición. Los puntos negros indican los elementos del dominio y los rectángulos grises indican las particiones de equivalencia. La Figura 4.4a muestra la partición de equivalencia correcta, que cumple las dos condiciones anteriores. La Figura 4.4b muestra una partición incorrecta: un elemento pertenece a dos particiones de equivalencia, lo que viola las condiciones de corrección.

La Figura 4.4c muestra otra partición incorrecta: uno de los elementos del dominio no ha sido asignado a ninguna partición de equivalencia.

La cuestión de la corrección de la partición parece bastante obvia, pero en la práctica no es un problema trivial. En aplicaciones reales, los dominios y las particiones pueden ser muy complejos.

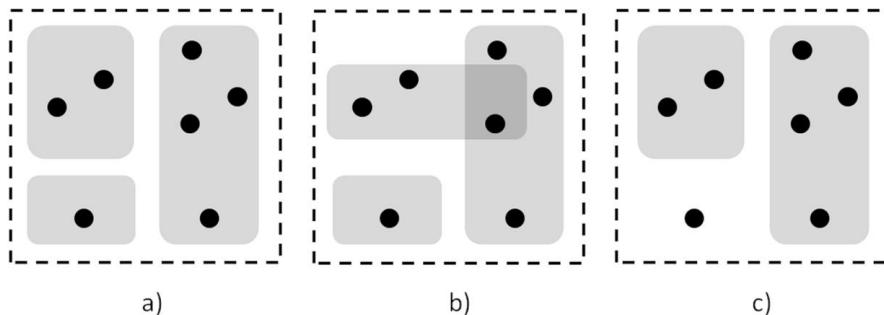


Fig. 4.4 Ejemplos de partición de dominio correcta (a) e incorrecta (b, c)

y tienen una estructura muy complicada. A menudo sucede que para una partición que hemos realizado, se violan algunas condiciones de corrección.

Ejemplo Consideremos el problema de prueba clásico descrito por Myers [10]. El programa toma tres números enteros no negativos, a , b , c , como entrada y genera el tipo de triángulo que se puede construir a partir de segmentos de las longitudes dadas. Las posibles respuestas del programa son “triángulo equilátero”, “triángulo isósceles”, “triángulo escaleno” y “no es un triángulo”. Supongamos que queremos aplicar la técnica EP al dominio de entrada (es decir, al conjunto de todos los triples posibles (a, b, c) de enteros no negativos), respecto al dominio de salida (es decir, el tipo de triángulo). Parece natural dividir el dominio de entrada en las cuatro particiones correspondientes a las cuatro posibles salidas mencionadas anteriormente. Sin embargo, tras una inspección más cercana, resulta que todo triángulo equilátero es también un triángulo isósceles, por lo que estamos tratando con una partición de equivalencia, que es un subconjunto propio de otra partición. Necesitamos corregir nuestra partición, definiendo las siguientes particiones:

- Entradas que representan triángulos equiláteros
- Entradas que representan triángulos isósceles, que no son equiláteros •
- Entradas que representan triángulos escalenos • Entradas que caen en la categoría “no es un triángulo”

Ahora bien, esta partición es correcta: cada triple de números dados a la entrada corresponde exactamente a una de las cuatro posibles particiones anteriores.

Ejemplo Queremos clasificar el conjunto de todas las posibles secuencias finitas de números en función de su orden. Una partición de equivalencia natural de dichos datos en particiones de equivalencia podría verse así: secuencias ascendentes, secuencias descendentes y secuencias desordenadas. Pero tenga en cuenta que una secuencia de un elemento es tanto ascendente como descendente. Además, surge la pregunta de dónde clasificar la cadena vacía (que contiene 0 elementos). Por lo tanto, la partición correcta debe tener en cuenta estos “casos extremos” y las particiones pueden verse así:

- Partición 1: la secuencia vacía • Partición 2: todas las secuencias de un elemento • Partición 3: todas las secuencias ascendentes con al menos dos elementos • Partición 4: todas las secuencias descendentes con al menos dos elementos • Partición 5: todas las secuencias desordenadas con más de dos elementos

Las particiones que contienen valores “normales” y “correctos”, es decir, valores esperados (aceptados) por el sistema, se denominan particiones válidas. Las particiones que contienen valores que el componente o sistema debería rechazar (por ejemplo, datos con sintaxis incorrecta, que exceden los rangos aceptables, etc.) se denominan particiones no válidas.

En nuestro último ejemplo, las cinco particiones que hemos identificado son válidas porque cada una de ellas contiene los datos correctos que espera el sistema (quizás el caso de la cadena vacía sea controvertido; si la especificación no menciona explícitamente cadenas no vacías, entonces la partición 1 puede considerarse una partición no válida). Además de estas particiones identificadas, podemos distinguir, por ejemplo, una partición no válida formada por elementos que no son cadenas numéricas (por ejemplo, contienen caracteres alfanuméricos).

Las definiciones de particiones válidas e inválidas, y en consecuencia de valores válidos e inválidos, pueden entenderse de diferentes maneras, por lo que si vamos a utilizar estos términos, vale la pena definirlos con precisión. Por ejemplo, los valores válidos se pueden entender al menos de dos maneras:

- Como aquellos que deben ser procesados por el sistema. •
Como aquellos para los cuales la especificación define su procesamiento.

Del mismo modo, se pueden entender valores incorrectos:

- Como aquellos que deben ser ignorados o rechazados por el sistema. •
Como aquellos para los cuales la especificación no define su procesamiento.

Ejemplo En el formulario de registro de usuario, el sistema espera que se ingrese una dirección de correo electrónico válida en el campo “correo electrónico”. Cuando el usuario ingresa allí la cadena de caracteres “abc@def@ghi”, el sistema rechaza esta entrada como inválida, informándole al usuario con el mensaje “Correo electrónico incorrecto”. En esta situación, la entrada “abc@def@ghi” puede ser tratada por algunos como un valor no válido (debido a que el sistema lo rechazó, el registro se realizará solo si se proporciona la dirección de correo electrónico correcta) y por otros como un valor proveniente de la partición válida (debido a que el sistema está preparado para este tipo de situaciones, la prueba de ello es el mensaje de error; por lo que es en cierto sentido un valor “esperado” y, por lo tanto, proveniente de la partición válida).

Mayor división de particiones en subparticiones Una ventaja importante de la técnica EP es que si divide alguna (incluso todas) particiones en subparticiones, aún tendrá una partición equivalente. Las pruebas para un dominio tan fragmentado serán más precisas. El evaluador puede decidir que ciertas particiones deben subdividirse aún más por determinadas razones. Este es un enfoque natural que aprovecha la naturaleza jerárquica de las particiones. Considere el siguiente ejemplo.

Ejemplo PESEL es el número de identificación personal de ciudadano utilizado por el gobierno polaco. Cada ciudadano polaco tiene asignado un número PESEL único. Es un número de 11 dígitos, en los que los primeros seis dígitos codifican la fecha de nacimiento, el último es el dígito de control y la imparidad del penúltimo dígito codifica el género: masculino o femenino.² El sistema toma el número PESEL del usuario, y, dependiendo de si la persona es mayor de edad o no, toma las medidas oportunas. Queremos realizar una partición de equivalencia para todas las secuencias posibles de dígitos. Como primer paso, podemos dividirlos en números válidos y no válidos, es decir, números que representan números PESEL válidos y todo lo demás. A continuación, podemos considerar cada una de estas particiones en términos de una posible división adicional. Sin duda, los números PESEL correctos deben dividirse, según la especificación, en números correspondientes a adultos y menores. Una división adicional de cada una de estas particiones podría, por ejemplo, tener en cuenta la codificación adecuada del número del mes (por ejemplo, para los nacidos entre 1900 y 1999, el mes se codifica normalmente; para los nacidos entre 2000 y 2099, 20 es agregado al número del mes (por ejemplo, febrero se codifica como 22, para 2100 a 2199, se agrega 40; para 2200 a 2299, se agrega 60 y para 1800 a 1899, se agrega 80);

A su vez, el número PESEL puede ser incorrecto por diversas razones, como estructurales (por ejemplo, longitud incorrecta) o sustantivas (por ejemplo, inconsistencia en el número de cheque). En la Fig. 4.5 se muestra un ejemplo del proceso de división de particiones en subparticiones .

Tengamos en cuenta que las personas nacidas entre 2000 y 2099 pueden ser tanto adultos como menores (suponiendo que hagamos la división en 2024), por lo que dividimos este período de tiempo en 2000–2006 y 2007–2099. Asumimos, a efectos de prueba, que las personas nacidas después de 2024 (lo que obviamente es imposible en 2024) se clasifican como menores. En las pruebas, sólo nos preocuparemos de verificar si se aceptan los números PESEL con la codificación de mes correcta, y queremos verificar esto para todas las codificaciones definidas en la especificación PESEL (1800–1899, 1900–1999, 2000–2099, 2100 –2199, 2200–2299). Finalmente, como resultado de la división jerárquica de particiones, dividimos el dominio relacionado con los números PESEL en 12 particiones de equivalencia:

- Tres particiones válidas para adultos (1800–1899, 1900–1999, 2000–2006) • Tres particiones válidas para menores (2007–2099, 2100–2199, 2200–2299) • Tres particiones no válidas debido a la sintaxis (estructura) de el número PESEL • Tres particiones no válidas debido a la semántica (significado) del número PESEL

Note que podríamos ir más lejos con la división. Por ejemplo, cada partición de equivalencia válida podría dividirse en dos particiones, codificando los géneros “masculino” y “femenino”.

Derivación de casos de prueba

En el caso de la técnica EP, los elementos de cobertura solo² particiones de equivalencia. El conjunto mínimo de casos de prueba para garantizar una cobertura del 100% es uno que cubra cada partición de equivalencia, es decir, para cada partición identificada, hay un caso de prueba que contiene

² El sistema PESEL es bastante antiguo; se introdujo en la década de 1970 y, por lo tanto, no tenía en cuenta otros géneros que el masculino y el femenino.

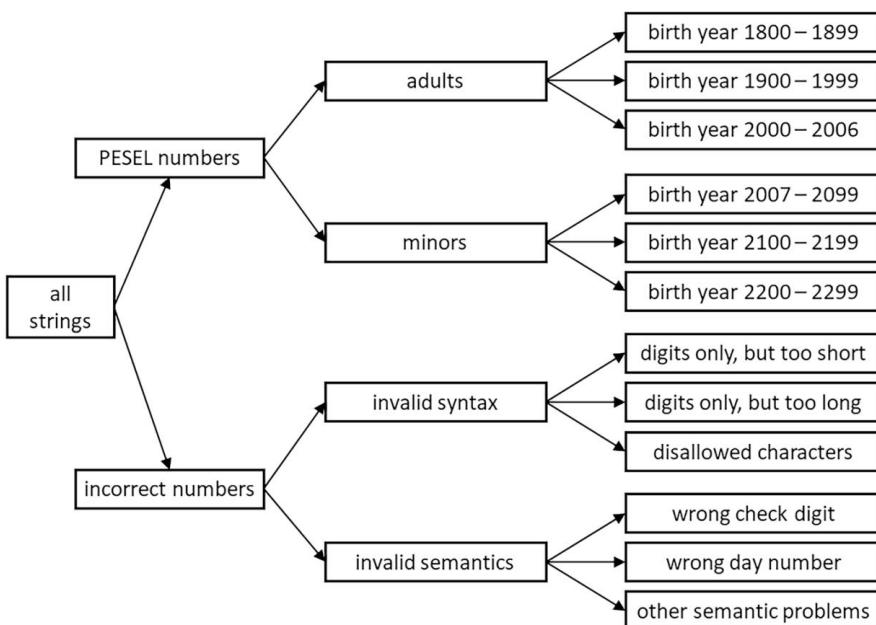


Fig. 4.5 Ejemplo de partición de equivalencia jerárquica

un valor de esa partición. En el caso unidimensional (un dominio y una división), el número mínimo de casos de prueba debería ser tanto como las particiones de equivalencia que hemos identificado. En el caso multidimensional (más de un dominio), el asunto se complica un poco más, ya que el número de casos de prueba dependerá, por ejemplo, de la forma en que tratemos las combinaciones de particiones no válidas, así como de posibles dependencias o restricciones entre valores y particiones de diferentes dominios. La cobertura se mide como el número de particiones de equivalencia probadas utilizando al menos un valor dividido por el número total de particiones de equivalencia definidas y generalmente se expresa como un porcentaje.

Ejemplo Consideremos nuevamente el sistema de verificación PESEL. Supongamos que la partición se parece a la de la Fig. 4.5 y que hemos definido los siguientes casos de prueba:

- PESEL de un adulto nacido en 1898 •
- PESEL de un adulto nacido en 1999 •
- PESEL = "1234" (demasiado corto)

Este conjunto de tres casos de prueba cubre 3 de las 12 particiones de equivalencia identificadas (ver Fig. 4.5), por lo que logra una cobertura de $3/12 = 1/4 = 25\%$.

Cubrir múltiples particiones de equivalencia simultáneamente: enmascaramiento de defectos Se necesita un enfoque ligeramente diferente en una situación en la que nuestras pruebas deben cubrir simultáneamente particiones de equivalencia derivadas de más de un dominio. De tal

En una situación, es una buena práctica no crear casos de prueba que cubran dos o más particiones no válidas. Esto tiene que ver con el llamado enmascaramiento de defectos. La estrategia recomendada es la siguiente:

1. Primero, cree la menor cantidad posible de casos de prueba compuestos únicamente de datos de prueba de particiones válidas, que cubrirán todas las particiones válidas de todos los dominios.
2. Luego, para cada partición no válida descubierta, cree un caso de prueba separado en el que se producirán los datos de esa partición y todos los demás datos provendrán de las particiones válidas.

Considere el siguiente ejemplo para ilustrar este enfoque.

Ejemplo El sistema otorga una calificación a un estudiante basándose en dos datos: el número de puntos de los ejercicios (0–50) y del examen (0–50). El alumno aprueba la asignatura si la puntuación total supera 50. Así, los datos de entrada del caso de prueba constarán de dos partes: puntos por los ejercicios y puntos por el examen. Supongamos que hemos distinguido los siguientes dominios y sus particiones:

Dominio de la variable A = "puntos de ejercicio":

- (A1) partición no válida: números menores a 0 • (A2) partición válida: números del 0 al 50 • (A3) partición no válida: números mayores a 50

Dominio de la variable B = "puntos del examen":

- (B1) partición no válida: números menores a 0 • (B2) partición válida: números del 0 al 50 • (B3) partición no válida: números mayores a 50

Queremos cubrir todas las particiones de equivalencia de los dos dominios, A y B. Las particiones válidas son A2 y B2. Todas las demás (A1, A3, B1, B3) son particiones no válidas.

Cada caso de prueba contiene como datos de entrada una cierta cantidad de puntos de ejercicio y una cierta cantidad de puntos de examen, por lo que un solo caso de prueba siempre cubre una partición del dominio A y otra del dominio B. Siguiendo la estrategia descrita anteriormente, cubrimos primero las particiones válidas. Como sólo tenemos una partición válida en cada dominio, un caso de prueba es suficiente, por ejemplo:

TC1: A = 25, B = 30 (cubre la partición válida A2 y la partición válida B2).

Quedan dos particiones no válidas para cubrir desde A y dos desde B. Por lo tanto, necesitamos cuatro casos de prueba más en los que estas particiones se probarán individualmente (la otra partición en un caso de prueba determinado debe ser la partición válida):

TC2: A = -8, B = 35 (cubre la partición inválida A1; cubre adicionalmente B2), TC3: A = 48, B = -11 (cubre la partición inválida B1; cubre adicionalmente A2), TC4: A = 64, B = 4 (cubre la partición inválida A3; cubre adicionalmente B2), TC5: A = 12, B = 154 (cubre la partición inválida B3; cubre adicionalmente A2).

Si estuviéramos probando una situación en la que ambos valores provienen de particiones no válidas, podría ocurrir el fenómeno de enmascaramiento de defectos. Supongamos que el sistema verifica que un

El estudiante ha aprobado una materia mediante el siguiente procedimiento (descrito en pseudocódigo):

```

ENTRADA: EjerciciosPuntos, ExamenPuntos
SI (Puntos de ejercicios + Puntos de examen > 50) ENTONCES
    VOLVER "Curso aprobado."
DEMÁS
    VOLVER "Curso fallido".

```

Ahora considere el siguiente caso de prueba:

TC6: A = -28, B = 105 (cubre las particiones A1 y B3 no válidas).

En esta situación, la puntuación total será $-28 + 105 = 77$ y, por lo tanto, el sistema devolverá un resultado de "Curso aprobado", ja pesar de que ambas entradas son incorrectas!

Cobertura de "cada elección"

La cobertura de "cada elección" se aplica al caso multidimensional, es decir, el caso en el que hay más de un dominio y cada caso de prueba cubre una partición de la distribución de cada dominio. Esta cobertura es uno de los tipos de cobertura más simples (y débiles) aplicados al caso multidimensional. Requiere que cada partición de cada dominio se pruebe al menos una vez. En la práctica, al utilizar este método, el evaluador intenta que el siguiente caso de prueba cubra tantos elementos de cobertura previamente descubiertos como sea posible.

Ejemplo Estamos probando un producto COTS para venta general. Esto significa que necesitamos probarlo en diferentes entornos. El programa funciona con diferentes sistemas operativos y diferentes navegadores. Por tanto, es necesario probar el funcionamiento de diferentes navegadores en diferentes sistemas operativos. Supongamos que tenemos los siguientes cuatro navegadores para probar:

- Google Chrome (GC) •
- Firefox (F) •
- Safari (S) •
- Opera (O)

y los siguientes tres sistemas operativos:

- Windows (W) •
- Linux (L) • iOS

Para simplificar, no incluimos versiones específicas de sistemas operativos o navegadores. Cada tipo de navegador forma una partición de equivalencia de un elemento, lo que da como resultado cuatro particiones: {GC}, {F}, {S} y {O}. Cada sistema operativo, a su vez, crea una partición de equivalencia de un elemento, haciendo un total de tres particiones: {W}, {L} y {iOS}.

Según el criterio de cobertura de "cada elección", para cada partición identificada, debe haber un caso de prueba que cubra un valor de esa partición. En nuestro ejemplo, los casos de prueba están representados por un par de datos de prueba (tipo de navegador, sistema operativo).

En nuestro ejemplo, cuatro casos de prueba son suficientes para cumplir el criterio de cobertura, por ejemplo:

TC1: (GC, G)

TC2: (F, L)

TC3: (S, iOS)

TC4: (O, M)

Tipos de problemas detectados La

técnica EP identifica problemas resultantes de un procesamiento de datos defectuoso, es decir, resultantes de errores en el modelo de dominio.

Definir el dominio es clave La técnica

EP siempre se aplica a un dominio específico que modela el problema. A veces, la elección del dominio es obvia, pero otras veces, el asunto puede resultar un poco más complicado. Consideré el siguiente ejemplo bastante típico.

Ejemplo. El termostato apaga la calefacción cuando la temperatura (calculada en grados completos) supera los 21 grados y se enciende cuando la temperatura baja de los 18 grados. Diseñe casos de prueba utilizando particiones de equivalencia.

¿Cómo aplicar la técnica EP a este problema? ¿Cuántas particiones de equivalencia habrá que comprobar? Eso depende de qué propiedad específica del sistema queremos verificar. Podemos abordar nuestro problema al menos de tres maneras: Método 1. Analizando solo el dominio, notamos que para los valores 22 y superiores, el sistema apaga la calefacción; para valores 17 y menores, se enciende; y para valores entre 18 y 21, no realiza ninguna acción. Por lo tanto, tenemos tres particiones de equivalencia para el dominio de "temperatura", hasta 17 grados, de 18 a 21 grados y por encima de 21 grados (ver Fig. 4.6a), y para cubrirlas, necesitamos tres casos de prueba, por ejemplo:

- Temperatura = 15 (potencia esperada: calefacción encendida). •
- Temperatura = 20 (salida esperada: el estado de calentamiento no cambia en comparación con el estado anterior). •
- Temperatura = 22 (salida esperada: calefacción apagada).

Método 2. Tenga en cuenta que en el rango de temperatura 18-21, la calefacción puede estar encendida o apagada. Por tanto, consideramos un dominio compuesto por pares (temperatura, estado de calentamiento). Ante esta situación tenemos cuatro posibilidades:

- Temperatura <18, calefacción encendida
- Temperatura 18–21, calefacción encendida

(continuado)

- Temperatura 18–21, calefacción apagada •

Temperatura >21, calefacción apagada

Así pues, tenemos cuatro situaciones que cubrir (véase la figura 4.6b). La situación ahora es un poco más complicada que antes, porque antes de ejecutar la prueba, debemos forzar la condición de calentamiento adecuada para temperaturas de 18 a 21°C. Por lo tanto, los casos de prueba que cubren las cuatro particiones mencionadas anteriormente podrían verse así:

- TC1: temperatura = 15 (potencia esperada, calefacción encendida). • TC2: temperatura = 18 después de aumentar desde 17 (potencia esperada, la calefacción aún está en funcionamiento).
- en).
- TC3: temperatura = 21 después de bajar de 22 (potencia esperada, la calefacción es todavía apagado).
- TC4: temperatura = 22 (potencia esperada, calefacción apagada).

Método 3: No podemos considerar pares estáticos (temperatura, estado de calentamiento), sino transiciones entre dichos pares. Entonces nuestro dominio describirá posibles transiciones entre pares (temperatura, estado de calentamiento) y constará de seis elementos posibles (en la siguiente lista de elementos del dominio, los números denotan temperaturas, y APAGADO y ENCENDIDO denotan calentamiento apagado y calentamiento encendido, respectivamente):

- (17, ENCENDIDO) → (18, ENCENDIDO)
- (18, ENCENDIDO) → (17, ENCENDIDO)
- (18, APAGADO) → (17, APAGADO)
- (17, APAGADO) → (21, APAGADO)
- (22, APAGADO) • (21, APAGADO) → (22, APAGADO)
- (22, APAGADO) → (21, APAGADO)

Esta situación se muestra en la figura 4.6c. Por lo tanto, necesitamos seis casos de prueba para simular estas transiciones; por ejemplo, para el primer elemento, la configuración inicial del sistema es de 17 grados y el estado de calentamiento está activado. La prueba consiste en aumentar la temperatura a 18 grados y ver si se apaga la calefacción.

Tenga en cuenta que podríamos agregar cuatro transiciones más a nuestra lista, que implican permanecer en el mismo dominio, si no queremos limitarnos a cambios que involucran transiciones entre diferentes rangos de temperatura que forman particiones de equivalencia del dominio de "temperatura":

- (17, ENCENDIDO) → (16, ENCENDIDO)
- (19, ENCENDIDO) → (18, APAGADO)
- (20, APAGADO) → (19, APAGADO)
- (18, APAGADO) → (22, APAGADO)
- (22, APAGADO) → (23, APAGADO)

Entonces, como puede ver en el ejemplo anterior, el objetivo de la prueba afecta significativamente la forma del dominio. En nuestro caso, el dominio se modeló en tres

(continuado)

diferentes maneras: como un conjunto de números que representan temperaturas, como un conjunto de pares (temperatura, estado de calentamiento) y como un conjunto de transiciones entre dichos pares. La aplicación de la técnica EP a cada uno de estos resultó en la verificación de un aspecto significativamente diferente del sistema. Por lo tanto, siempre necesitamos saber exactamente cuál es la forma del dominio en el que trabajaremos antes de aplicar esta técnica.

Tenga en cuenta también que, en ocasiones, la necesidad de especificar con precisión el resultado esperado de la prueba nos permite detectar errores o ambigüedades en la especificación. En el método 1 anterior, para los datos de la partición 18 a 21, no está muy claro cuál se supone que es el resultado esperado: si se supone que el termostato debe estar encendido o apagado.

Esto puede indicar que el problema que intentamos solucionar está mal planteado.

Al final del análisis de este ejemplo, observemos una cosa más muy importante. Es decir, para este problema particular, la técnica de partición de equivalencia no es una buena opción (los problemas que encontramos al aplicarla lo ilustran muy bien). Esto se debe a que el PE se centra en detectar problemas en la implementación del dominio (estático), y la cuestión fundamental en el problema de prueba que estamos considerando aquí es la verificación del comportamiento del termostato. Parece que una técnica más apropiada sería la prueba de transición de estado (ver Sección 4.2.4).

4.2.2 Análisis de valor límite (BVA)

BVA como extensión del análisis de valores

Límite EP es una técnica basada en la técnica de partición de equivalencia. Por lo que su versatilidad y el tipo de problemas detectados serán similares a este último. La diferencia con la partición de equivalencia es que en BVA seleccionamos elementos muy específicos de las particiones de equivalencia para realizar pruebas, es decir, aquellos que se encuentran en los límites de estas particiones.

Valores límite: ¿a qué dominios se aplica la BVA?

Un valor límite de una partición es el elemento más pequeño o más grande de esa partición.

Hablar de “más pequeño” y “más grande” sólo tiene sentido si definimos una relación de orden entre los elementos del dominio. Por lo tanto, la técnica BVA sólo se puede aplicar a dominios cuyos elementos están ordenados en términos de alguna relación de orden (por ejemplo, a conjuntos de números, con una relación “ $<$ ”). Ejemplos de tales dominios son conjuntos de números naturales, enteros o reales, valores relacionados con la hora o la fecha, etc.

Los valores límite siempre se definen para una partición de equivalencia específica. Por ejemplo, si el dominio “edad”, que contiene los números naturales entre 1 y 120, se divide en dos particiones de equivalencia, $\{1, 2, \dots, 18\}$ (niño) y $\{19, 20, \dots, 120\}$ (adulto), entonces los valores límite de la partición “secundaria” son 1 (la más pequeña) y 18 (la más grande). Los valores límite para la partición “adulta” son 19 y 120.

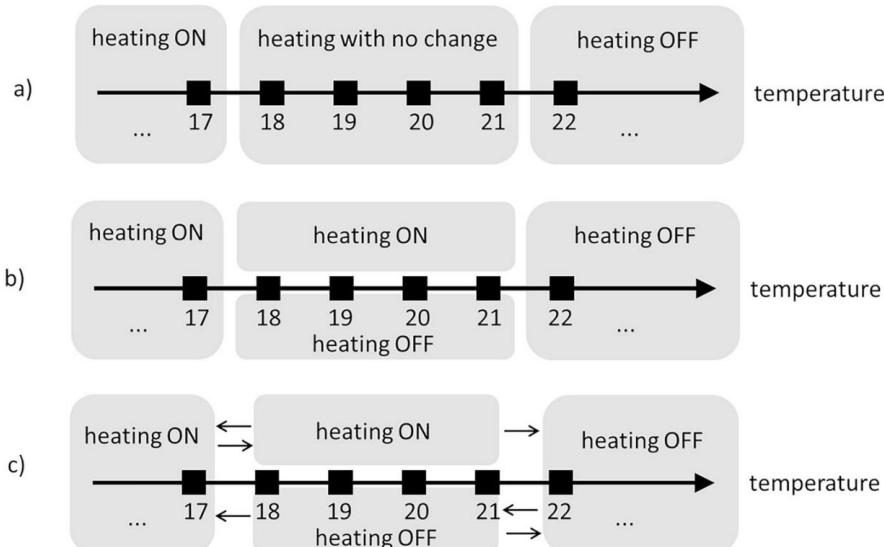


Fig. 4.6 Tres enfoques diferentes para seleccionar un dominio para particiones de equivalencia

Además, debemos suponer una cosa más: las particiones de equivalencia consideradas no pueden tener “huecos”, es decir, formalmente hablando, si dos valores a y b tales que $a < b$ pertenecen a la misma partición de equivalencia, entonces todos los elementos intermedios c , es decir, tal que $a < c < b$, también debe pertenecer a la misma partición. ¿Por qué? Considere el ejemplo mencionado anteriormente pero elimine el elemento 4 de la partición “secundaria”. Esta partición tiene la forma $\{1, 2, 3, 5, 6, \dots, 18\}$. Sus valores extremos siguen siendo, por supuesto, 1 y 18, pero surge la pregunta: ¿no son los valores 3 y 5 también “límites” de esta partición? Después de todo, ¡ambos están adyacentes a un elemento que no pertenece a la partición! En este caso, entonces, tendríamos que dividir nuestra partición en dos, $\{1, 2, 3\}$ y $\{5, 6, \dots, 18\}$, y el valor de 4 sería una partición separada. Entonces, el dominio se dividiría en cuatro particiones: $\{1, 2, 3\}$, $\{4\}$, $\{5, 6, \dots, 18\}$ y $\{19, \dots, 120\}$.

Derivar casos de prueba con BVA

El procedimiento general para derivar casos de prueba utilizando la técnica BVA es el siguiente:

1. Identifique el dominio que desea analizar.
2. Realice la partición de equivalencia de este dominio en particiones de equivalencia.
3. Para cada partición de equivalencia identificada, determine sus valores límite (nota: a veces el análisis puede limitarse solo a ciertas particiones y es posible que no tenga que tener en cuenta todas las particiones de equivalencia determinadas).
4. Para cada valor límite, determine los elementos de cobertura (los elementos que se probarán) para este valor límite.

Los dos primeros pasos simplemente aplican la técnica EP. En el paso 3, identificamos los límites de las particiones. En el paso 4, con base en los valores límite identificados,

determinar los elementos que se utilizarán en los casos de prueba como datos de entrada de prueba. En la técnica BVA, los valores límite (condiciones de prueba) no son necesariamente los mismos que los elementos que se seleccionarán para la prueba (elementos de cobertura). Esto dependerá de qué particiones estemos considerando y de la variante elegida del método BVA. Esto se debe a que existen dos variantes principales del BVA: los llamados BVA de 2 valores y BVA de 3 valores. Discutamoslos en detalle.

BVA de 2 valores

En el BVA de 2 valores [10, 43], para cada valor límite identificado, ese valor y su vecino más cercano que no pertenece a la partición a la que pertenece el valor límite se seleccionan para la prueba. Por ejemplo, si consideramos los límites de la partición $P = \{1, 2, 3, 4, 5, 6\}$ en el dominio de los enteros (es decir, los valores 1 y 6), seleccionamos para probar:

- Valor de 1 (como valor límite de P) • Valor de 0
(como el vecino más cercano de 1 que no pertenece a P)
- Valor de 6 (como valor límite de P) • Valor de 7 (como el vecino más cercano de 6 no perteneciente a P)

BVA de 3 valores

En el BVA de 3 valores [45, 49] para cada valor de límite identificado, tomamos ese valor y sus dos vecinos para realizar pruebas, independientemente de a qué particiones pertenezcan. En el ejemplo anterior, seleccionaríamos para probar:

- Valor de 1 (como valor límite de P) • Valor de 0
(como vecino izquierdo de 1) • Valor de 2 (como vecino derecho de 1)
- Valor de 6 (como valor límite de P) • Valor de 5 (como vecino izquierdo de 6) • Valor de 7 (como vecino derecho de 6)

Comparación de BVA de 2 y 3 valores

Esquemáticamente, el principio de determinación de los valores límite, junto con los elementos de cobertura asociados, se muestra en la Fig. 4.7. Un caso especial puede ser una partición de un solo elemento, pero el principio funciona de manera análoga. El único elemento de esta partición es tanto su elemento más pequeño como su elemento más grande. En el ejemplo de la figura, la partición 3 contiene solo el valor 7. En el caso del BVA de 2 valores, los valores tomados para la prueba serían los siguientes:

- Un valor de 7 como valor de límite mínimo y un valor de 6 como vecino fuera de la partición.
- Un valor de 7 como valor de límite máximo y un valor de 8 como vecino fuera de la partición.

Entonces tomaríamos el conjunto de valores 6, 7 y 8 para realizar la prueba. De manera similar, para el BVA de 3 valores, los valores para la prueba son 6, 7 y 8 determinados por 7 como valor mínimo y los mismos valores (6, 7, 8) determinados por 7 como valor máximo. Por lo tanto, para la prueba, tomaríamos un conjunto de los mismos valores que en el BVA de 2 valores: 6, 7 y

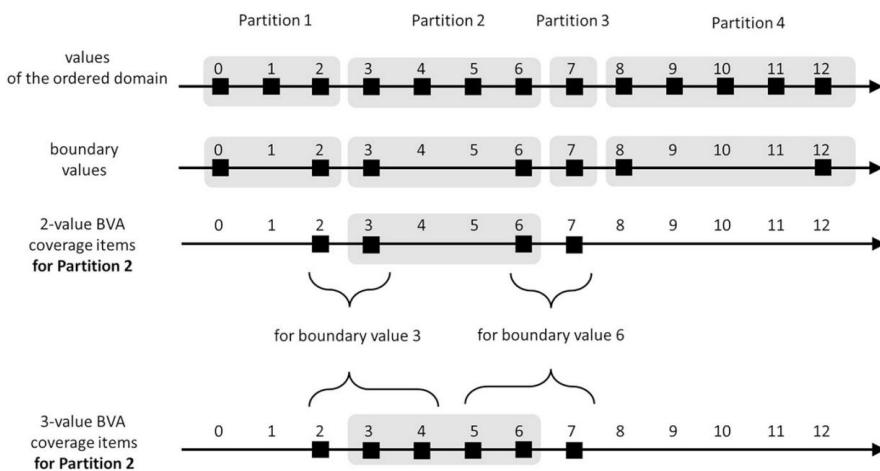


Fig. 4.7 Determinación de límites y elementos de cobertura en las versiones de dos y tres puntos del método AWB

8. Este es el caso sólo para particiones de un solo elemento. Para particiones con al menos dos elementos, el BVA de 3 valores siempre nos dará más elementos para probar que el BVA de 2 valores.

Consideremos ahora algunos ejemplos de la aplicación de la técnica BVA.

Ejemplo El sistema asigna un descuento en el billete a pasajeros menores de 18 años (descuento niños) y mayores de 65 años (descuento senior). Un pasajero entre 18 y 65 años no tiene derecho a descuento. Queremos comprobar la exactitud de la asignación de

el descuento. Llevemos a cabo el procedimiento de cuatro pasos descrito anteriormente para identificar los elementos de cobertura.

Paso 1. Identificar el dominio. La variable a analizar es la edad del pasajero, que es un número entero no negativo. Por tanto, el dominio tiene la forma $\{0, 1, 2, 3, \dots\}$.

Paso 2. Identificar particiones de equivalencia. A partir de la especificación identificamos las siguientes particiones de equivalencia:

- P1: edad elegible para descuento para niños $\{0, 1, 2, \dots, 17\}$ • P2: edad elegible para un boleto regular $\{18, 19, \dots, 64, 65\}$ • P3: edad elegible para adulto mayor descuento $\{66, 67, 68, \dots\}$

Paso 3. Identificar los valores límite:

- Los valores límite de P1 son 0 y 17. • Los valores límite de P2 son 18 y 65. • El valor límite de P3 es 66 (en este ejemplo, P3 no tiene el valor límite más grande valor).

Paso 4. Identificar los valores a probar. En caso de que queramos aplicar el BVA de 2 valores para todas las particiones de equivalencia, de hecho, se deben tomar todos los valores límite identificados para realizar la prueba, y eso es suficiente. Esto se debe a que tenemos una especie de simetría entre dos valores de límite adyacentes cualesquiera de dos particiones. Por ejemplo, 65 como valor límite de P2 tiene un vecino 66 desde fuera de la partición, que también es un valor límite de P3, y su vecino de otra partición es 65. Por lo tanto, utilizando el BVA de 2 valores, para realizar pruebas, seleccione todos los valores límite identificados: 0, 17, 18, 65 y 66. Asumimos aquí que el valor -1 no es factible.

En el caso del BVA de 3 valores, elegimos para probar 0, 1, 16, 17, 18, 19, 64, 65, 66 y 67 porque:

- Para el valor límite 0, tomamos este valor y su vecino: 0 y 1 (asumimos que el valor -1 es inviable).
- Para el valor límite 17, tomamos este valor y sus vecinos: 16, 17 y 18. • Para el valor límite 18, tomamos este valor y sus vecinos: 17, 18 y 19. • Para el valor límite 65 , tomamos este valor y sus vecinos: 64, 65 y 66. • Para el valor límite 66, tomamos este valor y sus vecinos: 65, 66 y 67.

Por supuesto, algunos valores se repiten, pero tomamos cada valor para probarlo solo una vez. Si solo la partición del medio ("boleto normal") estuviera sujeta a nuestro análisis, nos limitaríamos a identificar los límites de esta partición únicamente, por lo que los valores 18 y 65. Entonces, en el BVA de 2 valores, elegiríamos los valores 17, 18, 65 y 66 para realizar la prueba, y en el BVA de 3 valores, elegiríamos los valores 17, 18, 19, 64, 65 y 66.

Ejemplo En algún lugar del código, se toma una decisión dependiendo del valor de la variable entera x. La decisión debe ser de la forma:

```

SI x < 16 ENTONCES
    Realizar la ACCIÓN 1
DEMÁS
    Realizar la ACCIÓN 2
  
```

En este caso existen dos particiones de equivalencia: la primera contiene valores que provocan la ejecución de la "ACCIÓN 1", que son números enteros menores a 16; el segundo es su complemento, es decir, los valores mayores o iguales a 16. Provocan la ejecución de la "ACCIÓN 2". El límite de la primera partición es 15 y el límite de la segunda partición es 16. La variable x no tiene ni el valor más pequeño ni el más grande, por lo que la primera partición no tiene un valor de límite mínimo y la segunda partición no tiene un valor de límite máximo. Así, en el BVA de 2 valores elegiremos 15 y 16 para realizar la prueba, y en el BVA de 3 valores elegiremos 14, 15, 16 y 17.

En la práctica, normalmente existen los valores más grandes y más pequeños a nivel global: estos son, por ejemplo, los rangos aceptados por campos del tipo correspondiente (por ejemplo, un campo acepta un número que consta de hasta cinco dígitos) o los valores de las variables correspondientes. Por ejemplo, una variable de tipo int (entero) en C++ tiene un rango de -231 a 231 - 1, es decir, de -2.147.483.648 a 2.147.483.647. Por lo tanto, un buen evaluador comprobará no sólo los límites de partición establecidos por la especificación sino también los límites establecidos por la arquitectura de la solución bajo prueba, por ejemplo, el rango de valores variables.

Determinación cuidadosa de los valores límite En el caso del BVA, se debe tener mucho cuidado acerca de cómo se definen los límites de la partición. Supongamos que estamos trabajando en el dominio de los números naturales. Entonces, por ejemplo, la formulación "la partición contiene elementos no menores a 7" significa que el número más pequeño que pertenece a esta partición es 7. A su vez, la formulación "la partición contiene elementos mayores a 7" significa que el valor más pequeño de esta la partición es 8. De manera similar, "como máximo 65" significa números hasta 65 inclusive, y la condición " $x < 65$ " significa que el valor más grande que satisface esta desigualdad es 64. La frase "la partición contiene los valores que van desde x hasta y " significa que la partición contiene los valores de x y, incluidos y . Quienes tengan dudas al respecto se animan a plantearse el problema: ¿qué días abre la tienda, en el que cuelga un papel que dice: "la tienda abre de lunes a viernes?". ¿La tienda está abierta 5 días a la semana de lunes a viernes o solo de martes a jueves?

Aplicación EI

método BVA es muy simple pero extremadamente efectivo para detectar tipos específicos de defectos. La razón de esto es que los desarrolladores a menudo cometen lo que se llama "errores por uno", implementando incorrectamente los límites de partición de equivalencia. A continuación se muestran dos ejemplos de errores típicos de los desarrolladores que provocan defectos detectables por la BVA:

- El desarrollador debería haber implementado la condición "si $x < 10$ " (desigualdad estricta) pero la implementó erróneamente como "si $x \leq 10$ " (desigualdad débil).
- El desarrollador debería inicializar la variable de control del bucle (iterador) como "para $i=0$ a 10", pero asumió erróneamente que los elementos de la matriz están indexados desde 1 y lo escribió como "para $i=1$ a 10".

En el primer caso, las particiones de equivalencia, según la especificación, deberían quedar así: $\{..., 8, 9\}, \{10, 11, ...\}$. Sin embargo, la implementación incorrecta dividió el dominio (debido al flujo de control realizado) de la siguiente manera: $\{..., 9, 10\}, \{11, 12, ...\}$. Analizando este ejemplo con el BVA, vemos que los valores límite (válidos, según la especificación) son 9 y 10. Si utilizamos el BVA de 2 valores, tomamos estos mismos valores para probar. En el caso del valor 10, según la especificación, el programa debería tomar el camino correspondiente a la rama "falso" en la decisión "si $x < 10$ ", pero en la implementación incorrecta, la decisión " $x \leq 10$ " es cierto, por lo que el control del programa tomará el camino equivocado. Existe una buena posibilidad de detectar este defecto con un caso de prueba que utiliza 10 como valor de datos de prueba para x .

El BVA de 3 puntos es más fuerte que el BVA de 2 puntos, es decir, hay situaciones en las que el BVA de 2 valores no tiene posibilidades de detectar una falla, mientras que el BVA de 3 valores puede provocar una falla. Considere el siguiente ejemplo.

Ejemplo Un desarrollador implementa un componente para controlar la temperatura en una sala de laboratorio. La temperatura, medida en grados completos, no debe exceder los 10 °C. Cuando se supera este umbral, se activa el mecanismo de enfriamiento. La lógica de negocio es la siguiente:

SI la temperatura no supera los 10°C ENTONCES

Hacer nada

DEMÁS

Encienda el enfriamiento

El desarrollador ha implementado la decisión correcta IF $x \leq 10$ THEN correctamente, ... incorpora como IF $x == 10$ THEN.... Según la especificación, la partición del dominio debe ser $\{..., 9, 10\}, \{11, 12, ...\}$. Sin embargo, según la implementación incorrecta, la partición es la siguiente, $\{..., 8, 9\}, \{10\}, \{11, 12, ...\}$, con la partición del medio dando el valor de verdad y las partes izquierda y particiones correctas que dan el valor falso de la decisión en la implementación incorrecta.

Si utilizamos el BVA de 2 valores, identificamos los valores 10 y 11 como valores límite y solo los probamos. Para el valor 10 tanto en la implementación esperada como en la incorrecta, la decisión es verdadera: porque es cierto que $10 \leq 10$ y que $10 == 10$.

En cambio, para el valor 11, ambas decisiones son falsas: pues no es cierto que $11 \leq 10$, y no es cierto que $11 == 10$. Por lo tanto, ninguna de estas dos pruebas podrá detectar el defecto. — el flujo de control del programa (¡que contiene la implementación incorrecta de la decisión!) en ambos casos irá por el camino correcto: ¡el que resulta de la implementación esperada! Esto se debe a que a 10 °C el programa no realizará ninguna acción y a 11 °C encenderá el enfriamiento. Por tanto, el BVA de 2 valores no podrá detectar un defecto en el código.

Sin embargo, si utilizamos el BVA de 3 valores, tendremos que tomar cuatro valores para probar: 9, 10, 11 y 12. En particular, el valor de 9 diferencia las rutas de ejecución según la implementación correcta e incorrecta: en implementación correcta , la decisión $9 \leq 10$ es verdadera y, en caso de implementación incorrecta, la decisión $9 == 10$ es falsa. Por lo tanto, existe la posibilidad de que para $x = 9$, detectemos una operación incorrecta del programa debido a la selección de la ruta de flujo de control incorrecta. Al fin y al cabo, según las especificaciones, a 9 °C el programa no debe realizar ninguna acción, pero activará la refrigeración. Las consideraciones anteriores se resumen en la Tabla 4.3.

Valores fuera del dominio

Finalmente, consideraremos un problema más. En el ejemplo con la asignación de un descuento en el boleto, indicamos que para un valor límite de 0, no tomamos su vecino -1 para probar, porque asumimos que es inviable, es decir, el usuario no puede ingresar información incorrecta. , valores negativos. Sin embargo, en la práctica esto no tiene por qué ser siempre así. Si, por ejemplo, el usuario ingresa un valor de edad en un campo de formulario desde el teclado, puede, por supuesto, ingresar un valor negativo. La prueba o no de valores límite extremos fuera del dominio depende de si la interfaz lo permite. Si es así, el evaluador, por supuesto, debería realizar la prueba.

Cobertura

Los elementos de cobertura en el BVA son los valores límite de las particiones de equivalencia (en el BVA de 2 valores) o los valores límite junto con todos sus valores vecinos (en el BVA de 3 valores). Por lo tanto, para BVA de 2 valores, la cobertura se define como el cociente del número de valores límite probados y el número total de valores límite identificados. Para el BVA de 3 valores, es el cociente del número de pruebas

Tabla 4.3 Diferencias entre BVA de 2 y 3 valores

	BVA de 2 valores		BVA de 3 valores			
	Valores límite identificados (y valores para prueba): $x = 10$ $x = 11$					
			$x = 9$	$x = 10$	$x = 11$	$x = 12$
Resultados de la prueba						
Especificación: $x \leq 10$	(10 ≤ 10) VERDADERO	(11 ≤ 10) FALSO	(9 ≤ 10) VERDADERO	(10 ≤ 10) VERDADERO	(11 ≤ 10) FALSO	(12 ≤ 10) FALSO
Implementación incorrecta: $x = 10$	(10 = 10) VERDADERO	(11 = 10) FALSO	(9 = 10) FALSO	(10 = 10) VERDADERO	(11 = 10) FALSO	(12 = 10) FALSO
Resultado:	Prueba superada	prueba superada	Prueba fallida	Prueba aprobada	Prueba aprobada	Prueba aprobada
Interpretación:	Ambas pruebas pasaron, defecto no detectado		Hay una prueba que detectó el defecto.			

valores de límite con sus vecinos y el número total de valores de límite identificados y sus vecinos. Esto se debe a que en el BVA de 3 valores, no solo tomamos valores límite para realizar pruebas, sino también valores del interior de las particiones. Por ejemplo, para la partición {2, 3, 4, 5, 6}, los valores tomados para la prueba en el BVA de 3 puntos serán, en particular, los números 3 y 5, que no son los valores límite para esta partición. .

4.2.3 Prueba de la tabla de decisiones

La prueba de

la tabla de decisiones de aplicaciones  es una técnica que se utiliza para verificar la exactitud de las implementaciones de reglas comerciales. Una regla de negocio suele adoptar la forma de una implicación lógica:

SI (condición) ENTONCES (acción),

donde tanto la condición como la acción pueden estar compuestas por más de un factor. Entonces estamos ante una combinación de condiciones o acciones. A continuación se muestran algunos ejemplos de reglas comerciales:

- SI (edad del cliente < 18) ENTONCES (asignar un descuento en el billete); • SI ($a+b>c>0$ AND $a+c>b>0$ AND $b+c>a>0$) ENTONCES (puedes construir un triángulo con lados de longitud a , b , c); • SI (Salario mensual > 10000) ENTONCES (otorgar préstamo bancario Y ofrecer un tarjeta dorada).

Las tablas de decisión nos permiten probar sistemáticamente la corrección de la implementación de combinaciones de condiciones. Este es uno de los llamados combinatorios.

Tabla 4.4 Ejemplo de tabla de decisiones

	Reglas comerciales 1 a 8						
	12345678						
Condiciones							
¿Tiene una tarjeta de fidelización?	SI	SI	SI	SI	NO	NO	NO
¿Cantidad total > \$1000?	SÍ	SÍ	NO	NO	SÍ	SÍ	NO
¿Compras en los últimos 30 días?	SÍ	NO	SÍ	NO	SÍ	NO	Acciones
Descuento concedido	10%	5%	5%	0%	0%	0%	0%

técnicas. Para obtener más información sobre estas técnicas, consulte el programa de estudios Nivel avanzado: Analista de pruebas [28].

Construcción de la tabla de decisión

Describiremos la construcción de la tabla de decisión usando un ejemplo (ver Tabla 4.4).

La tabla de decisiones consta de dos partes, que describen las condiciones (parte superior) y las acciones (parte inferior), respectivamente. Las columnas individuales describen reglas comerciales. La Tabla 4.4 describe las reglas para asignar un descuento en compras en función de tres factores que describen al cliente en cuestión:

- ¿Tiene el cliente una tarjeta de fidelización? (Sí o NO) • ¿El monto total de las compras excede los \$1000? (Sí o NO) • ¿El cliente ha realizado compras en los últimos 30 días? (Sí o no)

Según las respuestas a estas preguntas, se asigna un descuento: 0%, 5% o 10%.

Ejemplo Un cliente tiene una tarjeta de fidelización y hasta el momento ha realizado compras por valor de 1250 \$, y las últimas compras se realizaron hace 5 días. Esta situación corresponde a la regla 1 (tiene tarjeta de fidelización, monto total >\$1000, compras en los últimos 30 días). Entonces, el sistema debería asignar un descuento del 10% a este cliente.

Derivar casos de prueba a partir de la tabla de decisiones

El proceso de creación de casos de prueba específicos utilizando tablas de decisión se puede presentar en los siguientes cinco pasos.

Paso 1. Identifique todas las condiciones individuales posibles (a partir de la base de prueba, por ejemplo, basadas en especificaciones, conversaciones con clientes, el llamado sentido común) y enumérlas en las filas consecutivas en la parte superior de la tabla. Si es necesario, divida las condiciones compuestas en condiciones individuales. Las condiciones suelen aparecer en las especificaciones como fragmentos de oraciones precedidos por palabras como "si", "en el caso de que", etc.

Paso 2. Identifique todas las acciones correspondientes que pueden ocurrir en el sistema y que dependen de estas condiciones (también derivadas de la base de prueba), y enumérlas en las líneas consecutivas en la parte inferior de la tabla. Las acciones suelen aparecer en las especificaciones como fragmentos de oraciones precedidos por palabras como "entonces", "en este caso", "el sistema debería", etc.

Paso 3. Generar todas las combinaciones de condiciones y eliminar las combinaciones de condiciones inviables. Para cada combinación factible, se crea una columna separada en la tabla con los valores de cada condición enumerada en esta columna.

Paso 4. Identificar, para cada combinación de condiciones identificada, qué acciones y cómo deberían ocurrir. Esto da como resultado completar la parte inferior de la columna correspondiente de la tabla de decisiones.

Paso 5. Para cada columna de la tabla de decisión, diseñe un caso de prueba en el que la entrada de la prueba represente una combinación de condiciones especificadas en esta columna. La prueba se supera si, tras su ejecución, el sistema realiza las acciones descritas en la parte inferior de la tabla en la columna correspondiente. Estas entradas de acción sirven como resultado esperado para el caso de prueba.

Notación y posibles entradas en la tabla de decisiones Normalmente, los valores de condición y acción toman la forma de valores lógicos VERDADERO o FALSO. Se pueden representar de varias maneras, como los símbolos T y F o Y y N (sí/no) o 1 y 0 o como las palabras "verdadero" y "falso". Sin embargo, los valores de las condiciones y acciones pueden ser, en general, cualquier objeto, como números, rangos de números, valores de categorías, particiones de equivalencia, etc. Por ejemplo, en nuestra tabla (Tabla 4.4), los valores de la acción "descuento otorgado" son categorías que expresan diferentes tipos de descuentos: 0%, 5% y 10%. En una misma tabla, puede haber condiciones y acciones de diferentes tipos, por ejemplo, condiciones lógicas, numéricas y categóricas pueden ocurrir simultáneamente.

La tabla de decisiones que solo tiene valores booleanos (verdadero/falso) se denomina tabla de decisiones de entrada limitada. Si para cualquier condición o acción existen entradas distintas a las booleanas, dicha tabla se denomina tabla de decisiones de entradas extendidas.

Cómo determinar todas las combinaciones de condiciones Si

necesitamos determinar manualmente combinaciones de condiciones y tememos perder algunas combinaciones, podemos usar un método de árbol muy simple para determinar sistemáticamente todas las combinaciones de condiciones. Considere el siguiente ejemplo:

Ejemplo Supongamos que una tabla de decisiones tiene tres condiciones:

- Ingresos (dos valores posibles: S, pequeño; L, grande) • Edad (tres valores posibles: Y, joven; MA, mediana edad; O, viejo) • Lugar de vida (dos valores posibles: C, ciudad; V, aldea)

Para crear todas las combinaciones de valores de triples (edad, ingresos, residencia), construimos un árbol, de cuya raíz derivamos todas las posibilidades de la primera condición (ingresos). Este es el primer nivel del árbol. A continuación, de cada vértice de este nivel derivamos todas las posibilidades de la segunda condición (edad). Obtenemos el segundo nivel del árbol. Finalmente, de cada vértice de este nivel se derivan todos los valores posibles de la tercera condición (lugar de vida). Por supuesto, si hubiera más condiciones, procederíamos de manera análoga. Nuestro árbol final se parece al de la Fig. 4.8.

Cada combinación posible de condiciones es una combinación de etiquetas de vértice en los caminos que van desde la raíz (el vértice en la parte superior del árbol) hasta cualquier vértice en la parte inferior del árbol. Por lo tanto, habrá tantas combinaciones como vértices en

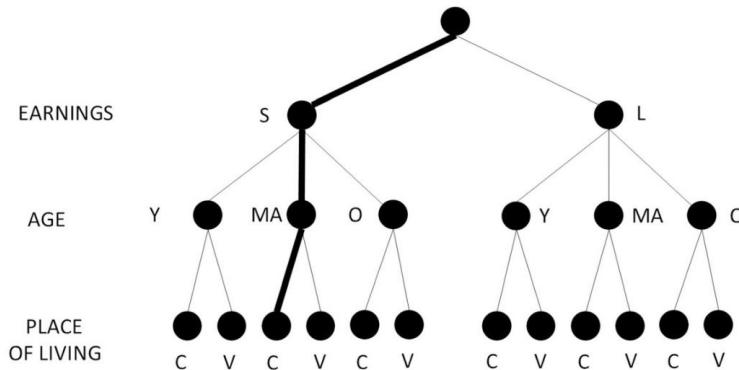


Fig. 4.8 Árbol de soporte para identificar combinaciones de condiciones

el nivel más bajo (en nuestro caso, 12; esto, por supuesto, se deriva del número de combinaciones: $2 \times 3 \times 2 = 12$). En la figura, las líneas en negrita indican el camino correspondiente a la combinación de ejemplo (S, MA, C), que significa ingresos bajos, mediana edad y ciudad como lugar de vida. Ahora podemos ingresar cada una de estas combinaciones en las columnas individuales de nuestra tabla de decisiones. Al mismo tiempo, estamos seguros de que ninguna combinación se ha quedado fuera. Las condiciones de la tabla de decisión así creada se muestran en la Tabla 4.5.

Combinaciones inviables A

veces, después de enumerar todas las combinaciones posibles de condiciones, es posible que descubra que algunas de ellas no son factibles por diversas razones. Por ejemplo, supongamos que tenemos las dos condiciones siguientes en la tabla de decisiones:

- ¿La edad del cliente es >18 ? (valores posibles: SÍ, NO) • ¿Edad del cliente ≤ 18 ? (valores posibles: SÍ, NO)

Es obvio que aunque tenemos cuatro posibles combinaciones de estas condiciones, (SÍ, SÍ), (SÍ, NO), (NO, SÍ) y (NO, NO), solo dos de ellas son factibles, (SÍ, NO) y (NO, SÍ), ya que no es posible tener más de 18 y menos de 19 años al mismo tiempo. Eso sí, en este caso podríamos sustituir estas dos condiciones por una, "edad", y por dos valores posibles: mayor que 18 y menor o igual a 18.

A veces, la tabla de decisiones no contendrá algunas combinaciones, no por razones puramente lógicas sino por razones semánticas. Por ejemplo, si tenemos dos condiciones:

- ¿Se definió el objetivo? (SÍ, NO) • ¿Se logró la meta? (SÍ, NO)

entonces la combinación (NO, SÍ) es inviable (sin sentido), porque es imposible posible lograr una meta que no has definido previamente.

Tabla 4.5 Combinaciones de condiciones de la tabla de decisión formadas a partir del árbol de soporte

	1 2 3		4	5	6	7	8 9		10	11	12
Ganancias	SSS		S	SSLLL				I	LL		
Edad	YY MA MA OOYY		MA MA OO								
Lugar de residencia	CVCV	CVCVC						VCV			

Minimizar la tabla de decisiones

En ocasiones, algunas condiciones pueden no tener ningún efecto sobre las acciones tomadas por el sistema. Por ejemplo, si el sistema permite que sólo los clientes adultos compren un seguro, dependiendo de si fuman o no obtienen un descuento en ese seguro, entonces

Mientras el cliente sea menor de edad, el sistema no le permitirá contratar un seguro. independientemente de si el cliente fuma o no. Estos valores irrelevantes son los más a menudo marcado en la tabla de decisiones con un símbolo de guión o N/A (no aplicable).

Normalmente, se utiliza un guión cuando ocurre la condición correspondiente, pero su valor es irrelevante para determinar la acción. El símbolo N/A, por otro lado, se utiliza cuando la condición no puede ocurrir. Por ejemplo, considere dos condiciones: "tipo de pago" (tarjeta o efectivo) y "¿el PIN es correcto?" (sí o no). Si el tipo de pago es efectivo, nosotros Ni siquiera puedo verificar el valor de la condición "¿Es correcto el PIN?" porque la condición no ocurre en absoluto. Entonces tenemos sólo tres combinaciones posibles de condiciones: (tarjeta, sí), (tarjeta, no) y (efectivo, N/A).

Esta minimización, o colapso, hace que la tabla de decisiones sea más compacta con menos columnas y, por lo tanto, menos casos de prueba para ejecutar. Por otra parte, para una prueba con un valor irrelevante en el caso de prueba real, tenemos que elegir algún valor específico, valor concreto para esta condición. Por lo tanto, existe el riesgo de que si se produce un defecto en algún combinación específica de valores marcados como irrelevantes en la tabla de decisión, podemos Es fácil pasarlo por alto y no probarlo.

Sin embargo, minimizar las tablas de decisión es un ejercicio de mitigación de riesgos: tal vez el La maqueta actual de la GUI no permite ciertas entradas, pero la implementación real o una API futura también podrían hacerlo.

Considere la tabla de decisiones contraída que se muestra en la Tabla 4.6. Si quisieramos diseñar un caso de prueba concreto para la primera columna, tendríamos que decidir si el El cliente fuma o no (aunque desde el punto de vista de la especificación esto es irrelevante), porque esta información debe darse. Podemos decidir la combinación. (adulto = NO, fuma = NO), y dicha prueba pasará, pero podemos imaginar que debido debido a algún defecto en el código, el programa no funciona correctamente para la combinación (adulto = NO, fuma = SI). Esta combinación no ha sido probada por nosotros y El fallo no será detectado.

En el examen real, puede haber preguntas que involucren tablas de decisión minimizadas, pero no se requiere que el candidato sea capaz de realizar la minimización sino sólo que sea capaz de comprender, interpretar y utilizar tablas de decisión que ya están minimizadas. El Se requiere minimización en el examen de certificación de Nivel avanzado: Analista de pruebas. Por lo tanto, en este libro no presentamos un algoritmo para minimizar tablas de decisión.

Cobertura

En las pruebas de tablas de decisiones, los elementos de cobertura son las columnas individuales de la tabla, que contienen posibles combinaciones de condiciones (es decir, las llamadas columnas factibles).

Tabla 4.6 Tabla de decisiones con valores irrelevantes

CONDICIONES			
¿Adulto?	NO	SÍ	SÍ
¿Fuma?	-	SÍ	NO
COMPORTAMIENTO			
¿Subvención de seguro?	NO	SÍ	SÍ
¿Descuento de subvención?	NO	NO	SÍ

Para una tabla de decisiones determinada, una cobertura total del 100% requiere que al menos un caso de prueba correspondiente a cada columna factible sea preparada y ejecutada. Se pasa la prueba si el sistema realmente ejecuta las acciones definidas para esa columna.

Lo importante es que la cobertura cuenta contra el número de (factibles) columnas de la tabla de decisiones, no contra el número de todas las combinaciones posibles de condiciones. Por lo general, estos dos números son iguales, pero en el caso de que ocurra de combinaciones inviables, como comentamos anteriormente, este podría no ser el caso.

Por ejemplo, para lograr una cobertura del 100% para la tabla de decisiones en Tabla 4.6, necesitamos tres (no cuatro, como sugeriría el número de combinaciones)

Casos de prueba. Si tuviéramos las siguientes pruebas:

- Adulto = Sí, fuma = Sí.
- Adulto = NO, fuma = Sí.
- Adulto = NO, fuma = NO.

entonces lograríamos una cobertura de 2/3 (o alrededor del 66%), ya que los dos últimos casos de prueba cubrir la misma primera columna de la tabla.

Tablas de decisión como técnica de prueba estática

La prueba de la tabla de decisiones es excelente para detectar problemas con los requisitos, como su ausencia o contradicción. Una vez creada la tabla de decisiones a partir de la especificación o incluso mientras aún se está creando, es muy fácil descubrir tales problemas de especificación como:

- Incompletitud: no hay acciones definidas para una combinación específica de condiciones.
- Contradicción: definir en dos lugares diferentes de especificación dos comportamientos del sistema frente a la misma combinación de condiciones
- Redundancia: definir el mismo comportamiento del sistema en dos lugares diferentes del sistema. especificación (quizás descrita de manera diferente)

4.2.4 Pruebas de transición estatal

Solicitud

La prueba de transición de estados  es una técnica utilizada para comprobar el comportamiento de un componente o sistema. Por lo tanto, verifica su aspecto conductual: cómo se comporta a lo largo del tiempo y cómo cambia su estado bajo la influencia de varios tipos de eventos.

El modelo que describe este aspecto del comportamiento es la llamada transición de estado. diagrama. En la literatura, diferentes variantes de este modelo se denominan autómata finito, autómata de estados finitos, máquina de estados o sistema de transición etiquetado. El

El programa de estudios utiliza el nombre “diagrama de transición de estados” para indicar la forma gráfica del modelo de transición de estados y “tabla de transición de estados” para indicar la forma tabular equivalente del modelo.

Construcción del diagrama de transición de estados Un diagrama de transición de estados es un modelo gráfico, como se describe en el estándar UML. Desde un punto de vista teórico, es un gráfico dirigido etiquetado. El diagrama de transición de estado consta de los siguientes elementos:

- Estados—representan posibles situaciones en las que el sistema puede estar • Transiciones—representan posibles (correctos) cambios de estados • Eventos —representan fenómenos, generalmente externos al sistema, la ocurrencia de que desencadena las transiciones correspondientes
- Acciones: acciones que el sistema puede realizar durante la transición entre estados. • Condiciones de guardia: condiciones lógicas asociadas con las transiciones; una transición puede ejecutarse sólo si la condición de guardia asociada es verdadera

La Figura 4.9 muestra un ejemplo de un diagrama de transición de estados. No es trivial, aunque en la práctica a menudo se utilizan modelos aún más complicados, utilizando una notación más rica que la analizada en el programa de estudios. Sin embargo, este diagrama nos permite mostrar todos los elementos esenciales del modelo de transición de estado descrito en el programa de estudios manteniendo el ejemplo práctico.

El diagrama representa un modelo del comportamiento del sistema al realizar una llamada telefónica a un usuario de teléfono celular con un número específico. El usuario marca un número de teléfono de nueve dígitos presionando las teclas correspondientes a los dígitos sucesivos del número uno por uno. Cuando se ingresa el noveno dígito, el sistema intenta realizar la llamada automáticamente.

El diagrama de transición de estados que modela este sistema consta de cinco estados (rectángulos). Las posibles transiciones entre ellos están indicadas por flechas. El sistema se inicia en la pantalla de bienvenida del estado inicial y espera un evento (etiquetado EnterDigit) en el que el usuario selecciona el primer dígito de un número de teléfono de nueve dígitos. Cuando ocurre este evento, el sistema pasa al estado Entrando y, además, durante esta transición, el sistema realiza una acción para establecer el valor de la variable x en 1. Esta variable representará la cantidad de dígitos del número de teléfono marcado. ingresado por el usuario hasta el momento.

En el estado Entrando, solo puede ocurrir el evento EnterDigit, pero dependiendo de cuántos dígitos se hayan ingresado hasta el momento, son posibles transiciones a dos estados diferentes. Mientras el usuario no haya ingresado el noveno dígito, el sistema permanece en el estado Entrando, aumentando cada vez la variable x en uno. Esto se debe a que la condición de guardia “ $x < 8$ ” es verdadera en esta situación. Justo antes de que el usuario ingrese el último noveno dígito, la variable x tiene un valor de 8. Esto significa que la condición de protección “ $x < 8$ ” es falsa y la condición de protección “ $x = 8$ ” es verdadera. Por lo tanto, al seleccionar el último noveno dígito del número se cambiará del estado Entrando bajo el evento EnterDigit al estado Conectar.

Cuando la llamada tiene éxito (la ocurrencia del evento ConnectionOK), el sistema ingresa al estado de Llamada, en el que permanece hasta que el usuario finaliza la llamada, lo que será señalado por la ocurrencia del evento EndConnection. En este punto, el sistema pasa al estado final Fin y finaliza su funcionamiento. Si el sistema está en el estado Conectar y ocurre el evento ConnectionError, el sistema pasa al estado final, pero a diferencia de la transición análoga desde el estado Llamada, realizará adicionalmente la

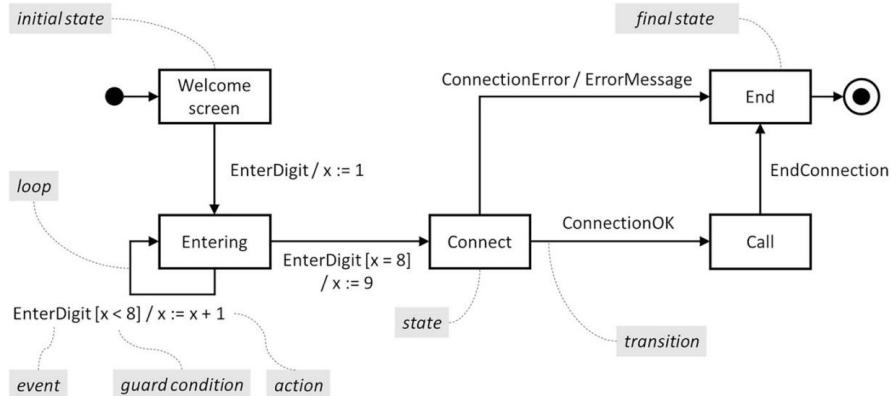


Fig. 4.9 Ejemplo de diagrama de transición de estado

Acción de ErrorMessage, que indica al usuario la imposibilidad de conectarse al número seleccionado.

El sistema en cada momento se encuentra exactamente en uno de los estados, y el cambio de estado se produce como resultado de la ocurrencia de los eventos correspondientes. El concepto de Estado es abstracto. Un estado puede denotar una situación de muy alto nivel (por ejemplo, el sistema está en una determinada pantalla de aplicación), pero también puede describir situaciones de bajo nivel (por ejemplo, el sistema ejecuta una determinada instrucción de programa). El nivel de abstracción depende del modelo adoptado, es decir, lo que el modelo realmente describe y en qué nivel de generalidad. Se supone que cuando ocurre un determinado evento, el cambio de estados es instantáneo (se puede asumir que es un evento de duración cero). Las etiquetas de transición (es decir, las flechas en el diagrama) tienen en general la forma:

evento [condición de guardia] / acción

Si, en un caso dado, la condición o acción de guardia no existe o no es relevante desde el punto de vista del evaluador, se pueden omitir. Por tanto, las etiquetas de transición también pueden adoptar una de las tres formas siguientes:

Evento

Evento/acción

Evento [condición de guardia]

Una condición de guardia para una transición determinada permite que esa transición se ejecute solo si la condición se cumple. Las condiciones de guardia nos permiten definir dos transiciones diferentes bajo el mismo evento evitando el no determinismo. Por ejemplo, en el diagrama de la Fig. 4.9 del estado Entrando, tenemos dos transiciones bajo el evento EnterDigit, pero solo una de ellas se puede ejecutar en cualquier momento porque las condiciones de protección correspondientes son independientes (ya sea x es menor que 8 o es igual a 8).

Un caso de prueba de ejemplo para el diagrama de transición de estado en la Fig. 4.9, verificando la exactitud de una conexión exitosa, podría parecerse al de la Tabla 4.7.

Tabla 4.7 Ejemplo de una secuencia de transiciones en el diagrama de transición de estados de la Fig. 4.9

Paso	Estado	Evento	Acción	siguiente estado
1	Pantalla de bienvenida	Ingresar dígito	$x := 1$	Entrando
2	Entrando	Ingresar dígito	$x := x + 1$	Entrando
3	Entrando	Ingresar dígito	$x := x + 1$	Entrando
4	Entrando	Ingresar dígito	$x := x + 1$	Entrando
5	Entrando	Ingresar dígito	$x := x + 1$	Entrando
6	Entrando	Ingresar dígito	$x := x + 1$	Entrando
7	Entrando	Ingresar dígito	$x := x + 1$	Entrando
8	Entrando	Ingresar dígito	$x := x + 1$	Entrando
9	Entrando	Ingresar dígito	$x := 9$	Conectar
10	Conectar	Conexión OK		Llamar
11	Llamar	Conexión final		Fin

El escenario se desencadena por una secuencia de 11 eventos: EnterDigit (9 veces), Conexión OK y Finalizar conexión. En cada paso, activamos el correspondiente evento y comprobar si, después de su ocurrencia, el sistema realmente pasa al estado descrito en la columna Siguiete estado.

Diagrama de formas equivalentes de transición de estado

Existen al menos dos formas equivalentes de representación de la transición estatal. diagrama. Considere una máquina simple de cuatro estados con los estados S1, S2, S3 y S4 y eventos A, B y C. El diagrama de transición de estados se muestra en la figura 4.10a.

Sin embargo, de manera equivalente, se puede presentar en forma de tabla de estados, donde Las filas individuales (respectivamente, columnas) de la tabla representan estados sucesivos. (respectivamente, eventos, junto con las condiciones de guardia si existen), y en la celda en la intersección de la columna y la fila correspondientes al estado especificado S y El evento E se escribe como el estado objetivo al que el sistema debe hacer la transición si, estando en estado S, ocurre el evento E. Por ejemplo, si el sistema se encuentra actualmente en el estado S2 y el evento B ha ocurrido, el sistema debe pasar al estado S1. Si nuestro diagrama usara acciones, Habría que anotarlos en las celdas correspondientes de las tablas de la Fig. 4.10b, c.

Una forma más de representar la máquina de estados es mediante el uso de la transición completa. tabla que se muestra en la figura 4.10c. Aquí la tabla representa todas las combinaciones posibles de estados, eventos y condiciones de guardia (si existen). Como tenemos cuatro estados, tres eventos diferentes y sin condiciones de guardia, la tabla contendrá $4 * 3 = 12$ filas. El La última columna contiene el estado objetivo al que la máquina debe realizar la transición si, cuando se en el estado definido en la primera columna de la tabla, el evento descrito en la segunda ocurre la columna de la tabla. Si la transición en cuestión no está definida, el estado Siguiete La columna indica esto, por ejemplo, con un guión u otro símbolo fijo.

La ausencia de una transición (es decir, la ausencia de una combinación estado/evento) en el El diagrama de transición de estado se representa simplemente por la ausencia del correspondiente flecha. Por ejemplo, al estar en el estado S1, la máquina no tiene un comportamiento definido para el ocurrencia del evento B. Por lo tanto, no hay ninguna flecha saliente desde S1 etiquetada por B.

Tanto la tabla de estado como la tabla de transición completa nos permiten mostrar directamente los llamados transiciones no válidas, que también pueden probarse, y en algunas situaciones deben probarse. Las transiciones no válidas (las "flechas que faltan" en el diagrama de transición de estado) son

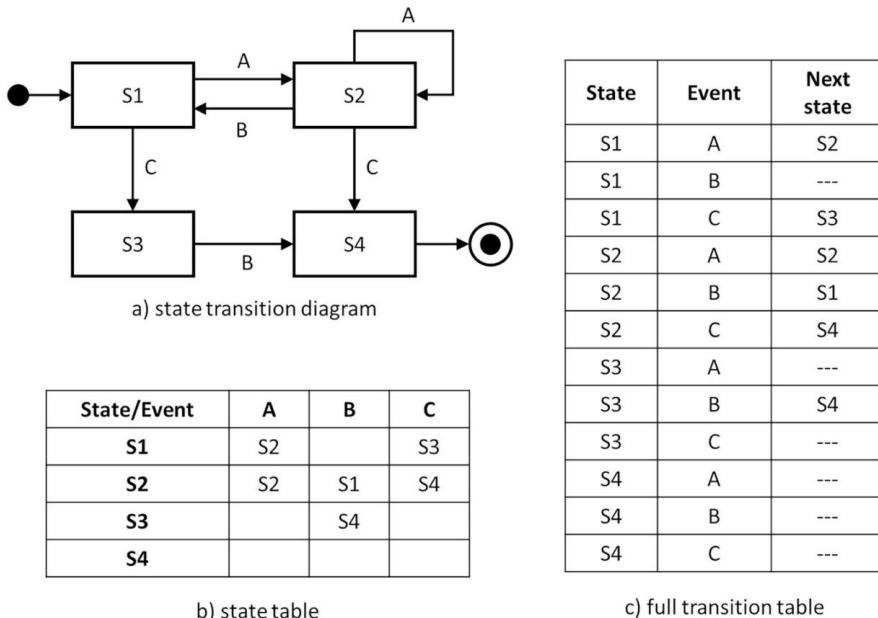


Fig. 4.10 Diferentes formas de presentación de máquinas de estados

representado por celdas vacías en las tablas. En otras palabras, una transición no válida es cualquier combinación de estado y evento que no aparece en el diagrama de transición de estado.

Por ejemplo, al diagrama de la figura 4.10 le faltan seis transiciones: (S1, B), (S3, A), (S3, C), (S4, A), (S4, B) y (S4, C). . Por lo tanto, tenemos un total de seis transiciones no válidas, correspondientes a seis celdas vacías en la tabla de la figura 4.10b o seis celdas vacías en la columna "Siguiiente estado" de la tabla de la figura 4.10c.

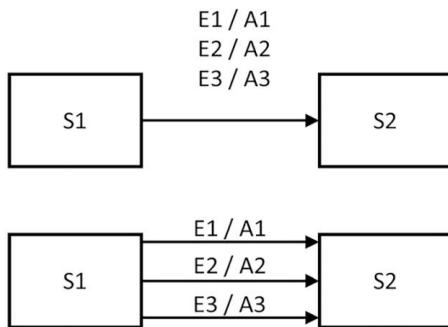
A veces, por simplicidad, solo se dibuja una flecha entre dos estados en el diagrama de transición de estados, incluso si hay más transiciones paralelas entre estos estados. Es importante prestar atención a esto, porque a veces la cuestión del número de transiciones en el diagrama es muy importante. La Figura 4.11 muestra un ejemplo de una forma "simplificada" de transiciones de dibujo y una forma "completa" equivalente.

Diseño de la prueba: Cobertura

Para las pruebas de transición estatales, existen muchos criterios de cobertura diferentes. Estos son los tres criterios más populares, descritos en el programa de estudios de Foundation Level:

- Cobertura de todos los estados: el criterio de cobertura más débil. Los elementos de cobertura son los estados. Por lo tanto, la cobertura de todos los estados requiere que el sistema haya estado en cada estado al menos una vez.
- Cobertura de transiciones válidas (también llamada cobertura de cambio 0 o cobertura de Chow): el criterio de cobertura más popular. Los elementos de cobertura son transiciones entre estados. Por lo tanto, la cobertura de transiciones válidas requiere que cada transición definida en el diagrama de transición de estado se ejecute al menos una vez.

Fig. 4.11 Formas equivalentes de representación gráfica de transiciones paralelas



- Cobertura de todas las transiciones: en la que los elementos de cobertura son todas las transiciones indicadas en la tabla estatal. Por lo tanto, este criterio requiere que se cubran todas las transiciones válidas y, además, se intente la ejecución de cada transición no válida. Es una buena práctica probar uno de estos eventos por prueba (para evitar el enmascaramiento de defectos).

Se pueden definir otros criterios de cobertura, como la cobertura de pares de transiciones (también denominada cobertura de 1 conmutador), que requiere que se prueben todas las secuencias posibles de dos transiciones válidas consecutivas. Los criterios de este tipo se pueden generalizar: podemos exigir la cobertura de las tres transiciones, de las cuatro transiciones, etc. En general, podemos definir toda una familia de criterios de cobertura con respecto a $N+1$ transiciones consecutivas, para cualquier N no negativo. N (llamada cobertura de conmutador N). También es posible considerar la cobertura de algunos caminos particulares, la cobertura de circuitos, etc. (todos estos criterios de cobertura no están dentro del alcance del programa de estudios de nivel básico).

Por lo tanto, en el caso de las pruebas de transición de estados, estamos tratando con un número potencialmente infinito de posibles criterios de cobertura. Desde un punto de vista práctico, los criterios más utilizados son la cobertura de transiciones válidas y la cobertura de todas las transiciones, ya que lo principal que normalmente nos preocupa al probar el diagrama de transición de estados es la verificación de la implementación válida de las transiciones entre estados.

La cobertura se define como el número de elementos cubiertos por las pruebas en relación con el número de todos los elementos de cobertura definidos por el criterio. Por ejemplo, para el criterio de cobertura de todos los estados aplicado al diagrama de transición de la figura 4.10, tenemos cuatro elementos que cubrir: los estados S_1 , S_2 , S_3 y S_4 ; para el criterio de cobertura de transiciones válido, tenemos tantos elementos como transiciones entre estados (seis).

El requisito habitual es diseñar el menor número posible de casos de prueba que sean suficientes para lograr una cobertura total. Veamos cómo se pueden realizar diferentes tipos de cobertura para el diagrama de transición de estados que se muestra en la figura 4.10a.

Para el criterio de cobertura de todos los estados, tenemos cuatro estados para cubrir: S_1 , S_2 , S_3 y S_4 . Tenga en cuenta que esto se puede lograr dentro de un único caso de prueba, por ejemplo:

$S_1(A) S_2(B) S_1(C) S_3(B) S_4$.

La convención utilizada anteriormente describe la secuencia de transiciones entre estados bajo la influencia de eventos (indicados entre paréntesis). La notación " $S(E)T$ " denota la transición del estado S bajo la influencia del evento E al estado T . En el

Tabla 4.8 Escenario de prueba para pruebas de transición de estado

Paso	Condición inicial	Evento	Resultado Esperado
1	T1	A	Transición a S2
2	T2	B	Transición a S1
3	T1	C	Transición a S3
4	T3	B	Transición a S4
5	T4		

En el ejemplo anterior, pasamos por los cuatro estados dentro de un caso de prueba, por lo que este único El caso de prueba logró una cobertura estatal del 100%. En la práctica, el resultado esperado de la prueba es el paso real de los estados uno por uno como lo describe el modelo. El escenario de prueba correspondiente al caso de prueba:

S1 (A) S2 (B) S1 (C) S3 (B) S4

podría verse así, como se describe en la Tabla 4.8.

Tenga en cuenta que un caso de prueba no se equipara con una condición de prueba. En nuestro ejemplo, la prueba Las condiciones eran estados individuales, pero un solo caso de prueba podría cubrirlos a todos. Una prueba El caso es una secuencia de transiciones entre estados, comenzando con el estado inicial y terminando con el estado final (posiblemente interrumpiendo este viaje antes si es necesario). Por lo que Es posible cubrir más de una condición de prueba dentro de un solo caso de prueba.

Para el criterio de cobertura de transiciones válido, tenemos seis transiciones que cubrir. Dejar los denotamos como T1–T6:

- T1: S1 (A) S2
- T2: T1 (C) S3
- T3: S2 (A) S2
- T4: S2 (B) S1
- T5: S2 (C) S4
- T6: S3 (B) S4

Queremos diseñar la menor cantidad de pruebas posible para cubrir estas seis transiciones. La estrategia es tratar de cubrir tanto como sea posible los elementos previamente descubiertos dentro de cada prueba sucesiva. Por ejemplo, si comenzamos el primer caso de prueba con la secuencia S1 (A) S2, en este punto no valdría la pena activar el evento C y pasar a la fase final. estado S4, cuando todavía podemos cubrir varias otras transiciones, como S2 (A) S2 (B) S1.

Tenga en cuenta que es imposible cubrir, dentro de un solo caso de prueba, las transiciones S2 (C) S4 y S3 (B) S4, porque tan pronto como se alcanza S4, el caso de prueba debe finalizar. Entonces, Necesitaremos al menos dos casos de prueba para cubrir todas las transiciones. De hecho, dos casos son suficiente para lograr una cobertura completa de las transiciones válidas. Un conjunto de ejemplo de estas dos pruebas. casos se muestra en la Tabla 4.9.

En esta tabla, la primera columna proporciona la secuencia de transiciones y la segunda La columna proporciona las transiciones cubiertas correspondientes. Transiciones cubiertas por primera vez. El tiempo se indica en negrita (por ejemplo, en la segunda prueba, cubrimos T1, que ya estaba cubierto previamente con el primer caso de prueba).

Tabla 4.9 Casos de prueba que logran una cobertura total de transiciones válidas

Caso de prueba	Transiciones cubiertas
S1 (A) S2 (A) S2 (B) S1 (C) S3 (B) S4	T1, T3, T4, T2, T6
S1 (A) S2 (C) S4	T1, T5

Para lograr la cobertura de todas las transiciones, debemos proporcionar casos de prueba que cubren todas transiciones válidas (ver arriba) y, además, intentar ejercer cada transición no válida. De acuerdo con las buenas prácticas descritas anteriormente, probaremos cada uno de estos transición en un caso de prueba separado. Por lo tanto, el número de casos de prueba será igual a el número de casos de prueba que cubren todas las transiciones válidas más el número de casos no válidos transiciones. En nuestro modelo, estas serán las siguientes seis transiciones no válidas (puede Escribalos rápidamente analizando la tabla de estado completa (consulte la figura 4.10c):

S1(B)?

T3 (A)?

T3 (C)?

T4 (A)?

T4 (B)?

T4 (C)?

Recuerde que cada caso de prueba comienza con un estado inicial. Así, por cada inválido transición, primero debemos llamar a la secuencia de eventos que alcanza el estado dado y luego, una vez que estemos en él, intente llamar a la transición no válida. Para las seis transiciones inválidas descrito anteriormente, los seis casos de prueba correspondientes podrían verse como en la Tabla 4.10.

Si logramos desencadenar un evento que no está definido en el modelo, podemos interpretar hacerlo al menos de dos maneras:

- Si el sistema cambia de estado, esto debe considerarse una falla, ya que desde el modelo no permite dicha transición, no debería ser posible desencadenarla.
- Si el sistema no cambió su estado, entonces esto puede considerarse un comportamiento correcto. (ignorando el evento). Sin embargo, debemos estar seguros de que semánticamente se trata de una situación aceptable.

También existen al menos dos posibles soluciones a situaciones tan problemáticas:

- Arreglar el sistema para que el evento no sea posible (la transición no válida es imposible que se va a activar).

Tabla 4.10 Casos de prueba que cubren transiciones no válidas

Caso de prueba	Transición incorrecta cubierta
S1(B)?	S1 (B)?
S1 (C) S3 (A)?	T3 (A)?
S1 (C) S3 (C) ?	T3 (C)?
S1 (C) S3 (B) S4 (A) ?	T4 (A)?
S1 (C) S3 (B) S4 (B) ?	T4 (B)?
S1 (C) S3 (B) S4 (C) ?	T4 (C)?

- Agregar una transición bajo la influencia de este evento al modelo para que modele el “ignorar” el evento por parte del sistema (por ejemplo, un bucle al mismo estado).

Volvamos al ejemplo práctico del diagrama de transición de estados de la figura 4.9 que modela una llamada telefónica. Un ejemplo de una transición no válida es la transición del estado Conectar en respuesta al evento EnterDigit. Esta situación se desencadena de forma muy sencilla: en el momento de establecer una llamada, simplemente pulsamos una de las teclas que representan dígitos. Si el sistema no reacciona de ninguna manera, consideramos que la prueba ha pasado.

Finalmente, analicemos uno de los criterios de cobertura no descritos en el programa: la cobertura de pares de transiciones válidas (cobertura de 1 cambio). Este es un material adicional, no examinable en el examen de certificación de nivel básico.

Consideraremos que este ejemplo muestra que cuanto más fuerte es el criterio que adoptamos, más difícil es y generalmente requiere diseñar más casos de prueba que para un criterio más débil (en este caso, el criterio más débil es la cobertura de transiciones válidas).

Para satisfacer la cobertura de 1 interruptor, debemos definir todos los pares permitidos de transiciones válidas. Podemos hacer esto de la siguiente manera: para cada transición única y válida, consideramos todas sus posibles continuaciones en la forma de la siguiente transición única y válida. Por ejemplo, para la transición S1 (A) S2, las posibles continuaciones son todas las transiciones que salen de S2, es decir, S2 (A) S2, S2 (B) S1 y S2 (C) S4. Por lo tanto, todos los pares posibles de transiciones válidas se ven así:

PP1: S1 (A) S2 (A) S2
 PP2: T1 (A) S2 (B) S1
 PP3: S1 (A) S2 (C) S4
 PP4: T1 (C) S3 (B) T4
 PP5: T2 (A) S2 (A) S2
 PP6: S2 (A) S2 (B) S1
 PP7: S2 (A) S2 (C) S4
 PP8: T2 (B) S1 (A) S2
 PP9: T2 (B) S1 (C) T3

Tenga en cuenta (análogo al caso de cobertura de transiciones válidas) que los pares de transiciones PP3, PP4 y PP7 terminan en el estado final, por lo que no pueden ocurrir dos de ellos en un solo caso de prueba, ya que la ocurrencia de cualquiera de estos pares de transiciones resulta en la terminación del caso de prueba. Por lo tanto, necesitamos al menos tres casos de prueba para cubrir pares de transiciones válidas y, de hecho, tres casos de prueba son suficientes. En la Tabla 4.11 se proporciona un conjunto de pruebas de ejemplo (los pares cubiertos por primera vez se indican en negrita). Por lo tanto, necesitamos un caso de prueba más en comparación con el caso de cobertura de transiciones válidas.

Tabla 4.11 Casos de prueba que cubren pares de transiciones válidas

Caso de prueba	Pares cubiertos de transiciones válidas.
S1 (A) S2 (A) S2 (B) S1 (A) S2 (B) S1 (C) S3 (B) T4	PP1, PP5, PP6, PP8, PP2, PP9, PP4
S1 (A) S2 (C) S4	PP3
S1 (A) S2 (A) S2 (C) S4	PP1, PP7

4.2.5 (*) Pruebas de casos de uso

Aplicación Un

caso de uso es un documento de requisitos que describe la interacción entre los llamados actores, que suelen ser el usuario y el sistema.

Un caso de uso es una descripción de una secuencia de pasos que realizan el usuario y el sistema, que en última instancia lleva a que el usuario obtenga algún beneficio. Cada caso de uso debe describir un escenario único y bien definido. Por ejemplo, si documentamos los requisitos para un cajero automático (un cajero automático), los casos de uso podrían, entre otras cosas, describir los siguientes escenarios:

- Rechazar una tarjeta de cajero automático
- no válida • Iniciar sesión en el sistema ingresando el PIN correcto •
- Retirar dinero correctamente de un cajero automático •
- Intento fallido de retiro de dinero debido a fondos insuficientes en la cuenta • Bloquear la tarjeta ingresando el PIN incorrecto tres veces

Para cada caso de uso, el evaluador puede construir un caso de prueba correspondiente, así como un conjunto de casos de prueba para verificar la ocurrencia de eventos inesperados durante la ejecución del escenario.

Construcción de casos de uso

Un caso de uso correctamente construido, además de información "técnica", como un número y un nombre únicos, debe consistir en:

- Condiciones previas (incluidos los datos de entrada) • Exactamente un escenario principal secuencial • (Opcional) Marcas de las ubicaciones de los llamados flujos alternativos y excepciones • Condiciones posteriores (que describen el estado del sistema después de la ejecución exitosa de el escenario y el beneficio obtenido para el usuario)

Desde el punto de vista del evaluador, el propósito principal del caso de uso es probar el escenario principal, es decir, verificar que realizar realmente todos los pasos como describe el escenario conduce al beneficio definido para el usuario. Sin embargo, al hacerlo, también es necesario probar el comportamiento del sistema en busca de eventos "insospechados", es decir, flujos alternativos y excepciones. La diferencia entre estos dos es la siguiente:

- Un flujo alternativo provoca que ocurra un evento inesperado pero permite al usuario completar el caso de uso, es decir, le permite regresar al escenario principal y completarlo felizmente.
- Una excepción interrumpe la ejecución del escenario principal; si se produce una excepción, no es posible completar el escenario principal correctamente. El usuario no obtiene ningún beneficio y el caso de uso finaliza con un mensaje de error o se cancela. La aparición de la excepción en sí se puede controlar mediante un escenario definido en un caso de uso independiente.

Un caso de uso no debe contener lógica de negocios, es decir, el escenario principal debe ser lineal y no contener ramificaciones. La existencia de tal ramificación sugiere que

en realidad están tratando con más de un caso de uso. Cada una de estas posibles rutas debe describirse en un caso de uso independiente.

Ejemplo Consideremos un ejemplo del escenario “Retiro correcto de dinero (\$100) de un cajero automático con una comisión de \$5”. Un caso de uso que describa este escenario podría parecerse al que se muestra a continuación.

Este caso de uso consta de un escenario principal (pasos 1 a 11), tres excepciones (etiquetadas 3A, 4A y 9A) y un flujo alternativo (9B). La numeración de estos eventos hace referencia al paso en el que pueden ocurrir. Si es posible más de una excepción o flujo alternativo en un solo paso, se indican con letras consecutivas: A, B, C, etc.

Caso de uso: UC-003-02

Nombre: retiro exitoso de \$100 de un cajero automático con una comisión de \$5.

Requisitos previos: el usuario ha iniciado sesión, la tarjeta de pago se reconoce como una tarjeta con una tarifa de retiro de \$5.

Pasos del escenario principal:

1. El usuario selecciona la opción de Retirar dinero.
2. El sistema muestra un menú con los montos de pago disponibles.
3. El usuario selecciona la opción \$100.
4. El sistema verifica si hay al menos \$105 en la cuenta del usuario.
5. El sistema solicita una copia impresa de la confirmación.
6. El usuario selecciona la opción NO.
7. El sistema muestra el mensaje Quitar tarjeta.
8. El sistema devuelve la tarjeta.
9. El usuario toma la tarjeta.
10. El sistema retira \$100 y transfiere \$5 de comisión a la cuenta bancaria.
11. El sistema muestra el mensaje Tomar dinero. El caso de uso finaliza, el usuario cierra la sesión, el sistema regresa a la pantalla inicial.

Condiciones finales: se retiraron \$100 del usuario, el saldo de la cuenta del usuario se redujo en \$105, el saldo de efectivo del cajero automático se redujo en \$100.

Fujos alternativos y excepciones:

3A - el usuario no selecciona ninguna opción durante 30 segundos (el sistema devuelve la tarjeta y vuelve a la pantalla inicial).

4A - el saldo de la cuenta es inferior a \$105 (mensaje Fondos insuficientes, devolver la tarjeta, registrar al usuario, volver a la pantalla inicial y finalizar el caso de uso).

9A - el usuario no toma la tarjeta durante 30 segundos (el sistema “extrae” la tarjeta, envía una notificación por correo electrónico al cliente y vuelve a la pantalla inicial).

9B - el usuario vuelve a insertar la tarjeta después de recibirla (el sistema devuelve la tarjeta).

Derivar casos de prueba a partir de un caso de uso

Hay un mínimo de lo que un evaluador debe hacer para probar un caso de uso. Este mínimo (entendido como cobertura total, 100% del caso de uso) es diseñar:

- Un caso de prueba para ejercitarse el escenario principal, sin eventos inesperados.
- Suficientes casos de prueba para ejercitarse cada excepción y flujo alternativo.

Entonces, en nuestro ejemplo, podríamos diseñar los siguientes cinco casos de prueba:

CT1: implementación de los pasos 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

TC2: implementación de los pasos 1, 2, 3A

TC3: implementación de los pasos 1, 2, 3, 4A

TC4: implementación de los pasos 1, 2, 3, 4, 5, 6, 7, 8, 9A

TC5: implementación de los pasos 1, 2, 3, 4, 5, 6, 7, 8, 9B, 10, 11

Cada excepción y flujo alternativo debe probarse en casos de prueba separados para evitar el enmascaramiento de defectos. Discutimos este fenómeno en detalle en la Sección. [4.2.1](#).

Sin embargo, en algunas situaciones, por ejemplo, presión de tiempo, se puede permitir probar más de un flujo alternativo dentro de un caso de prueba.

Destacar que TC5 (flujo alternativo) nos permite conseguir el objetivo (llegamos al paso 11).

TC2, TC3 y TC4 finalizan antes debido a la ocurrencia de situaciones incorrectas durante la ejecución del escenario. Podemos diseñar los casos de prueba a partir del caso de uso, en forma de escenarios, describiendo la respuesta esperada del sistema en cada paso. Por ejemplo, el escenario para el caso de prueba TC3 puede verse como en la Tabla [4.12](#).

Observemos que los pasos consecutivos (acciones del usuario y respuestas esperadas) corresponden a los pasos 1, 2, 3 y 4A del caso de uso consecutivo. Observe también que el evaluador verificó el valor límite para el monto del retiro en este escenario: se suponía que toda la operación reduciría el saldo de la cuenta del usuario en \$105, mientras que el saldo de la cuenta declarado era \$104,99. Éste es un ejemplo típico de combinación

Tabla 4.12 Caso de prueba construido a partir de un caso de uso

Caso de prueba TC3: Retiro en cajero automático con fondos insuficientes en la cuenta (relativo al caso de uso UC-003-02, ocurrencia de la excepción 4A).

Condiciones

previas: • El usuario ha

iniciado sesión • El sistema está en el menú

principal • El retiro se reconoce como cargado con una comisión de \$5 (debido al tipo de tarjeta) • Saldo en cuenta: \$104.99 Paso Evento

		Resultado Esperado
1	Seleccione la opción para Retirar dinero	El sistema pasa al menú de selección de importes.
2	Elije la opción \$100	El sistema encuentra que no hay fondos en la cuenta, muestra el mensaje Sin fondos, devuelve la tarjeta

Condiciones

posteriores: • El saldo de la cuenta del usuario y el saldo de la cuenta bancaria no han cambiado • El cajero automático no retiró dinero •

El cajero automático devolvió la tarjeta al usuario

• El sistema volvió a la pantalla de bienvenida

técnicas de prueba para incluir la verificación de muchos tipos diferentes de condiciones en un pequeño número de casos de prueba. Aquí, para verificar el correcto funcionamiento del sistema en una situación de fondos insuficientes en la cuenta, utilizamos la técnica del análisis del valor límite, de modo que la cantidad insuficiente era sólo 1 centavo menos que la cantidad que ya habría permitido el retiro correcto de dinero.

Cobertura

El estándar ISO/IEC 29119 no define una medida de cobertura para las pruebas de casos de uso. Sin embargo, el antiguo plan de estudios sí proporciona dicha medida, asumiendo que los elementos de cobertura son flujos de escenarios de casos de uso. Por lo tanto, la cobertura se puede definir como la relación entre el número de rutas de flujo verificadas y todas las rutas de flujo posibles descritas en el caso de uso.

Por ejemplo, en el caso de uso descrito anteriormente, hay cinco rutas de flujo diferentes: la ruta principal, tres flujos con manejo de excepciones y uno con flujo alternativo. Si el conjunto de pruebas contuviera solo los casos de prueba TC1 y TC5, la cobertura de casos de uso con estas pruebas sería $2/5 = 40\%$.

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.

Las pruebas de caja blanca se basan en la estructura interna del objeto de prueba. Muy a menudo, las pruebas de caja blanca están asociadas con las pruebas de componentes, en las que el modelo del objeto de prueba es la estructura interna del código, representada, por ejemplo, en forma del llamado gráfico de flujo de control (CFG). Sin embargo, es importante señalar que las técnicas de prueba de caja blanca se pueden aplicar a todos los niveles de prueba, por ejemplo:

- En las pruebas de componentes (estructura de ejemplo: CFG)
- En las pruebas de integración (estructuras de ejemplo: gráfico de llamadas,
- API) • En las pruebas de sistemas (estructuras de ejemplo: proceso de negocio modelado en BPMN idioma,³ menú del programa) •

En las pruebas de aceptación (estructura de ejemplo: estructura de páginas del sitio web)

El programa de estudios de Foundation Level analiza dos técnicas de prueba de caja blanca: prueba de declaraciones y prueba de rama. Ambas técnicas están asociadas por naturaleza con el código, por lo que su área de aplicación es principalmente la prueba de componentes. Además de éstas, existen muchas otras técnicas de caja blanca (y normalmente más potentes), como:

³ Modelo y notación de procesos de negocio, BPMN: una notación gráfica utilizada para describir el negocio. procesos

- Pruebas MC/DC •
- Pruebas de condiciones múltiples •
- Pruebas de bucle •
- Pruebas de ruta básica

Sin embargo, estas técnicas no se analizan en el programa de estudios del nivel básico.

Algunos de ellos se analizan en el programa de estudios de Nivel avanzado: Analista de pruebas técnicas [29]. Estas técnicas de prueba de caja blanca más potentes se utilizan a menudo cuando se prueban sistemas críticos para la seguridad (por ejemplo, software de instrumentos médicos o software aeroespacial).

4.3.1 Pruebas de declaraciones y cobertura de declaraciones

La prueba de declaraciones es la más simple y también la más débil de todas las técnicas de prueba de caja blanca. Implica cubrir declaraciones ejecutables en el código fuente de un programa.

Ejemplo Consideré un fragmento de código simple:

```

1. ENTRADA x, y // dos números naturales 2. SI (x > y) ENTONCES
3.
    z := x - y
    DEMÁS
4.     z := y - x
5. SI (x > 1) ENTONCES 6.
    z := z*2
7. REGRESAR z

```

Las declaraciones ejecutables están marcadas con los números del 1 al 7 en este código. El texto que comienza con los caracteres “//” es un comentario y no se ejecuta cuando se ejecuta el programa. En general, el código se ejecuta secuencialmente, línea por línea, a menos que se produzcan algunos saltos en el flujo de control (por ejemplo, en las sentencias de decisión 2 y 5 de nuestro código). La palabra clave ELSE es sólo una parte de la sintaxis de la instrucción IF-THEN-ELSE y por sí sola no se trata como una instrucción ejecutable (porque no hay nada que ejecutar aquí).

Después de tomar de la entrada dos variables, x e y, en la declaración 1, el programa verifica en la declaración 2 si x es mayor que y. Si es así, se ejecuta la declaración 3 (asignación a z de la diferencia x - y), seguida de un salto a la declaración 5. Si no, se ejecuta la declaración 4 (la parte “else” de IF-THEN-ELSE), en la que la diferencia y - x se asigna a la variable z, seguido de un salto a la declaración 5. En la declaración 5, se verifica si la variable x tiene un valor mayor que 1. Si es así, el cuerpo de la declaración IF-THEN (declaración 6) se ejecuta (duplicando el valor de z). Si la decisión en la declaración 5 es falsa, se omite la declaración 6 y el flujo de control salta después del bloque SI-ENTONCES, es decir, a la declaración 7, donde se devuelve el resultado (el valor de la variable z) y el programa termina. su funcionamiento.

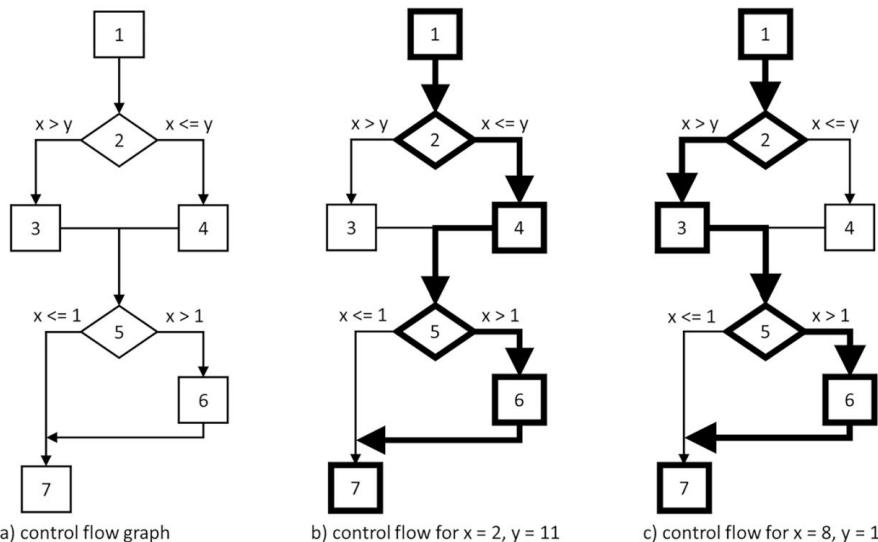


Fig. 4.12 CFG y ejemplos de flujo de control para diferentes datos de entrada

El código fuente se puede representar como un CFG. En la figura 4.12a se muestra un gráfico de este tipo para el código antes mencionado. Es un gráfico dirigido, en el que los vértices representan declaraciones y las flechas representan el posible flujo de control entre declaraciones. Las declaraciones de decisión se indican aquí con rombos, mientras que otras declaraciones se indican con cuadrados.

Considere dos casos de prueba para el código del ejemplo anterior:

- TC1: entrada: $x = 2, y = 11$; salida esperada: 18. • TC2:
entrada: $x = 8, y = 1$; resultado esperado: 14.

En la figura 4.12b, las flechas en negrita muestran el flujo de control cuando a la entrada se le dan los valores $x = 2, y = 11$. El flujo de control para TC1 será el siguiente (entre paréntesis, mostramos el resultado de la decisión en un momento dado). declaración de decisión):

$$1 \rightarrow 2 (x \leq y) \rightarrow 4 \rightarrow 5 (x > 1) \rightarrow 6 \rightarrow 7.$$

Por otro lado, para el TC2 con entrada $x = 8, y = 1$, se muestra el flujo de control en la Fig. 4.12c y será el siguiente:

$$1 \rightarrow 2 (x > y) \rightarrow 3 \rightarrow 5 (x > 1) \rightarrow 6 \rightarrow 7.$$

Enfatizemos una cosa muy importante en el contexto de las técnicas de prueba de caja blanca: los resultados esperados en los casos de prueba no se derivan del código. Esto se debe a que el código es precisamente lo que estamos probando, por lo que no puede ser su propio oráculo. Los resultados esperados se derivan de una especificación externa al código. En nuestro caso, podría ser así: "El programa toma dos números naturales y luego les resta los

uno más pequeño del más grande. Si el primer número es mayor que 1, el valor se duplica aún más. El programa devuelve el valor calculado de esta manera".

Cobertura

En las pruebas de declaraciones, los elementos de cobertura son las declaraciones ejecutables. Por de cobertura de  lo tanto, es el cociente de instrucciones ejecutables cubiertas por casos declaraciones de prueba por el número de todas las instrucciones ejecutables en el código analizado, generalmente expresado como porcentaje. Tenga en cuenta que esta métrica no tiene en cuenta declaraciones que no son ejecutables (por ejemplo, comentarios o encabezados de funciones).

Reconsideremos el código fuente anterior y los casos de prueba TC1 y TC2. Si nuestro conjunto de pruebas contiene sólo TC1, su ejecución alcanzará una cobertura de 6/7 (aprox. 86%), ya que este caso de prueba ejercita seis instrucciones ejecutables diferentes de las siete que componen nuestro código (éstas son 1, 2, 4, 5, 6 y 7). TC2 también ejercita seis de los siete enunciados (1, 2, 3, 5, 6, 7), por lo que logra la misma cobertura de ca. 86%. Sin embargo, si nuestro conjunto de prueba contiene TC1 y TC2, la cobertura será del 100%, porque juntos, estos dos casos de prueba ejercitan las siete declaraciones (1, 2, 3, 4, 5, 6, 7).

4.3.2 Pruebas de sucursales y cobertura de sucursales

Una rama es un flujo de control entre dos vértices de un CFG. Una rama puede ser incondicional o condicional. Una rama incondicional entre los vértices A y B significa que después de que se completa la ejecución de la declaración A, el control debe pasar a la declaración B. Una rama condicional entre A y B significa que después de que se completa la ejecución de la declaración A, el control puede pasar a la declaración B, pero no necesariamente.

Las ramas condicionales surgen de vértices de decisión, es decir, lugares del código donde se toma alguna decisión de la que depende el curso de control posterior. Ejemplos de declaraciones de decisión son las declaraciones condicionales IF-THEN, IF-THEN-ELSE y SWITCH-CASE, así como declaraciones que verifican la llamada condición de bucle en los bucles WHILE, DO-WHILE o FOR.

Por ejemplo, en la figura 4.12a, las ramas incondicionales son 1→2, 3→5, 4→5 y 6→7, porque, por ejemplo, si se ejecuta la declaración 3, la siguiente declaración debe ser la declaración 5. otras ramas, 2→3, 2→4, 5→6 y 5→7, son condicionales. Por ejemplo, después de la ejecución de la sentencia 2, el control puede pasar a la sentencia 3 o a la sentencia 4. Cuál de estos casos ocurría dependerá del valor lógico de la decisión SI en la sentencia 2. Si $x > y$, el control irá a 3, pero si $x \leq y$, irá a 4.

Cobertura

En las pruebas de sucursales, los elementos de cobertura son sucursales, tanto condicionales como incondicionales. Por lo tanto, la  cobertura de sucursales se calcula como el número de sucursales que se ejercieron durante la ejecución de la prueba dividido por el número total de sucursales en el código. El objetivo de las pruebas de sucursales es diseñar una cantidad suficiente de casos de prueba que alcancen el nivel requerido (aceptado) de cobertura de sucursales. Como en otros casos, esta cobertura suele expresarse en porcentaje. Cuando se logra una cobertura completa de sucursales del 100%, significa que todas las sucursales en el código, tanto condicionales como

incondicional: se ejecutó durante las pruebas al menos una vez. Esto significa que hemos probado todas las transiciones directas posibles entre declaraciones en el código.

Ejemplo Consideremos nuevamente el código de ejemplo de la Sección [4.3.1](#):

```
1. ENTRADA x, y // dos números naturales 2. SI (x > y) ENTONCES 3. z := x - y
MÁS
```

```
4.           z := y - x 5. SI
(x > 1) ENTONCES 6.
z := z * 2
7. REGRESAR z
```

En este código, tenemos ocho ramas:

$1 \rightarrow 2, 2 \rightarrow 3, 2 \rightarrow 4, 3 \rightarrow 5, 4 \rightarrow 5, 5 \rightarrow 6, 5 \rightarrow 7, 6 \rightarrow 7.$

El caso de prueba TC1 ($x = 2, y = 11$, resultado esperado: 18) cubre las siguientes ramas:

- $1 \rightarrow 2$ (incondicional) • $2 \rightarrow 4$
(condicional) • $4 \rightarrow 5$
(incondicional) • $5 \rightarrow 6$
(condicional) • $6 \rightarrow 7$
(incondicional)

y logra una cobertura de $5/8 = 62,5\%$.

El caso de prueba TC2 ($x = 8, y = 1$, resultado esperado: 14) cubre las siguientes ramas:

- $1 \rightarrow 2$ (incondicional) • $2 \rightarrow 3$
(condicional) • $3 \rightarrow 5$
(incondicional) • $5 \rightarrow 6$
(condicional) • $6 \rightarrow 7$
(incondicional)

y logra también una cobertura de $5/8 = 62,5\%$.

Los dos casos juntos logran $7/8 = 87,5\%$ de cobertura de sucursales, porque todos las ramas están cubiertas excepto una, $5 \rightarrow 7$.

Ejemplo Considere el código y su CFG de la figura [4.13](#).

Un solo caso de prueba con entrada $y = 3$ logra una cobertura de declaración del 100%. Esto se debe a que el control pasará por las siguientes declaraciones (el valor de la variable x asignado en la declaración 4 y la decisión verificada en la declaración 3 se dan entre paréntesis):

$1 \rightarrow 2 (x := 1) \rightarrow 3 (1 < 3) \rightarrow 4 (x := 2) \rightarrow 3 (2 < 3) \rightarrow 4 (x := 3) \rightarrow 3 (3 < 3) \rightarrow 5.$

```

1. INPUT Y
2. x=1;
3. WHILE (x < y) DO
4.     x=x+1;
5. PRINT ("End of the loop")
END

```

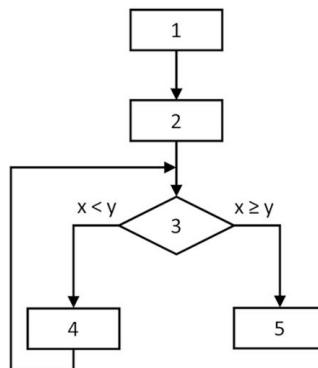


Fig. 4.13 El código fuente con un bucle while y su CFG

El mismo caso de prueba también logra una cobertura de sucursales del 100%. Hay cinco ramas en el código: $1 \rightarrow 2$, $2 \rightarrow 3$, $3 \rightarrow 4$, $3 \rightarrow 5$ y $4 \rightarrow 3$. Según el flujo de control, las ramas se cubrirán secuencialmente de la siguiente manera:

- $1 \rightarrow 2$ (incondicional) 2
- $\rightarrow 3$ (incondicional) 3 \rightarrow
- 4 (condicional) 4 \rightarrow 3
- (incondicional) 3 \rightarrow 4
- (condicional, cubierto anteriormente) 4
- $\rightarrow 3$ (incondicional, cubierto antes) 3 \rightarrow 5
- (condicional)

Ejemplo Considere el código de la figura 4.14a.

Su CFG, en el que cada vértice corresponde a un único enunciado, se muestra en la figura 4.14b. Hay nueve ramas en este gráfico:

- $1 \rightarrow 2$ (incondicional) •
- $2 \rightarrow 3$ (incondicional) • $3 \rightarrow 4$
- (condicional) • $3 \rightarrow 5$
- (condicional) • $5 \rightarrow 6$
- (condicional) • $5 \rightarrow 9$
- (condicional) • $6 \rightarrow 7$
- (incondicional) • $7 \rightarrow 8$
- (incondicional) • $8 \rightarrow 5$
- (incondicional)

Para lograr una cobertura de sucursales del 100%, necesitamos al menos dos casos de prueba, por ejemplo:

CT1: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$

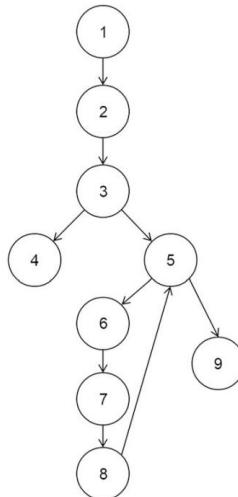
CT2: $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 5 \rightarrow 9$

```

1 INPUT a, b, c
2 d = a + b + c
3 IF (d>0) THEN
4   RETURN d
ELSE
5   WHILE (d<0) DO
6     a = a + 1
7     b = b - 1
8     d = d + 1
END WHILE
9 RETURN a * b * c
END IF
END

```

a) source code



b) control flow graph

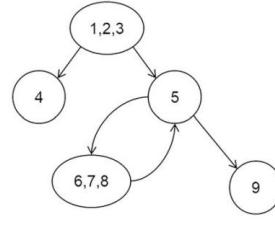
c) control flow graph
with basic blocks

Fig. 4.14 Dos CFG para el mismo código

TC1 cubre tres de nueve sucursales, por lo que logra una cobertura de sucursales de $3/9 \approx 33,3\%$. TC2 cubre ocho de las nueve sucursales, por lo que logra una cobertura de sucursales de $8/9 \approx 89\%$. Las dos pruebas juntas cubren las nueve ramas, por lo que el conjunto de pruebas {TC1, TC2} logra una cobertura completa de las ramas, $9/9 = 100\%$.

En ocasiones, un CFG se dibuja de manera que incluya varias proposiciones en un único vértice, constituyendo lo que se denomina bloque básico. Un bloque básico es una secuencia de declaraciones tal que cuando una de ellas se ejecuta, todas las demás deben ejecutarse. Un CFG modificado de esta manera es más pequeño y más legible (no contiene "largas cadenas de vértices"), pero su uso afecta la medida de cobertura, ya que habrá menos aristas que en un CFG sin bloques básicos. Por ejemplo, la gráfica de la figura 4.14c es equivalente a la gráfica de la figura 4.14b, pero tiene sólo cinco aristas. El TC1 mencionado anteriormente logra en este caso $1/5 = 20\%$ de cobertura de sucursales, mientras que el TC2 logra $4/5 = 80\%$ de cobertura.

Si la cobertura se mide directamente sobre el código, equivale a la cobertura calculada para un CFG, en el que cada vértice representa una sola instrucción, ya que las ramas representan el flujo directo de control entre instrucciones individuales, no entre sus grupos (bloques básicos).

4.3.3 El valor de las pruebas de caja blanca

La prueba de sentencias se basa en la siguiente hipótesis de error: si hay un defecto en el programa, éste debe localizarse en una de las sentencias ejecutables. Y si logramos una cobertura del 100% del extracto, entonces estamos seguros de que un extracto con un defecto

Fue ejecutado. Esto, por supuesto, no garantizará que se produzca un fallo, pero al menos creará esa posibilidad. Considere un ejemplo simple de código que toma dos números como entrada y devuelve su multiplicación (si el primer número es positivo) o devuelve el primer número (si el primer número no es positivo). El código tiene el siguiente aspecto:

```

1. ENTRADA x, y 2. SI
x>0 ENTONCES
3.      x:= x+y
4. REGRESAR x

```

En este programa, hay un defecto que involucra el uso incorrecto del operador de suma en lugar de multiplicación en la línea 3. Para los datos de entrada $x = 2$, $y = 2$, el programa revisará todas las declaraciones, por lo que cubrirá completamente la declaración al 100%. se logrará. En particular, la declaración 3 se ejecutará incorrectamente, pero dicho sea de paso, $2 * 2$ es igual a $2 + 2$, por lo que el resultado devuelto será correcto a pesar del defecto. Este sencillo ejemplo muestra que la cobertura de declaraciones no es una técnica sólida y que vale la pena utilizar otras más potentes, como las pruebas de cobertura de sucursales.

En cambio, en las pruebas de rama, la hipótesis del error es la siguiente: si hay un defecto en el programa, provoca un flujo de control erróneo. En una situación en la que hemos logrado una cobertura de sucursal del 100%, debemos haber ejercido al menos una vez una transición incorrecta (por ejemplo, el valor de decisión debería ser verdadero, pero era falso), para que el programa ejecute las declaraciones incorrectas, y esto posiblemente resultará en una falla.

Por supuesto, como en el caso de las pruebas de declaraciones, la cobertura total de la rama no garantiza que se produzca una falla, incluso si el programa toma el camino equivocado. En nuestro ejemplo, cuando agregamos un segundo caso de prueba con $x = 0$, $y = 5$, cubriremos adicionalmente la rama $2 \rightarrow 3$. Los dos casos de prueba juntos logran una cobertura de rama del 100%, pero aún así, el defecto en el código desaparecerá. no se identificará: los resultados reales en ambos casos de prueba serán exactamente los mismos que los resultados esperados.

Sin embargo, existe una relación importante entre las pruebas de declaraciones y las pruebas de rama:

Lograr una cobertura del 100% en sucursales garantiza el logro de una cobertura del 100% en el estado de cuenta.

En jerga científica, decimos que la cobertura de sucursal incluye la cobertura de declaración. Esto significa que cualquier conjunto de prueba que logre una cobertura de sucursal del 100%, por definición, también logra una cobertura de estado de cuenta del 100%. Por lo tanto, en tal caso no necesitamos verificar la cobertura del estado de cuenta por separado. Sabemos que debe ser al 100%.

La relación de subsunción inversa no es cierta: lograr una cobertura del 100% del estado de cuenta no garantiza que se logre una cobertura del 100% de las sucursales. Para demostrar esto, consideremos el fragmento de código proporcionado anteriormente. Este programa tiene cuatro declaraciones y cuatro ramas: $1 \rightarrow 2$, $2 \rightarrow 3$, $2 \rightarrow 4$ y $3 \rightarrow 4$. Para los datos de entrada $x = 3$, $y = 1$, el programa revisará las cuatro declaraciones, 1, 2, 3 y 4, por lo que este caso de prueba logrará una cobertura de declaración del 100%. Sin embargo, no hemos cubierto todas las ramas.

después de la ejecución de la sentencia 2, el programa pasa a la sentencia 3 (porque $3 > 0$), por lo que no cubre la rama $2 \rightarrow 4$ (que se ejecutaría en el caso de que la decisión en la sentencia 2 fuera falsa). Entonces, la prueba para $x = 3$, $y = 1$ logra una cobertura del 100% del estado de cuenta pero solo del 75% de cobertura de la sucursal.

Tenga en cuenta también que cada declaración de decisión es una declaración y, como tal, es parte del conjunto de declaraciones ejecutables. Por ejemplo, en el código anterior, la línea 2 es una declaración (y se trata como tal cuando utilizamos la técnica de cobertura de declaraciones). Entonces podemos decir que la cobertura del 100% de la declaración garantiza la ejecución de cada decisión en el código, pero no podemos decir que garantiza el logro de cada resultado de cada decisión (y por lo tanto no podemos decir que garantiza la cobertura de todas las ramas del código).

La ventaja clave de todas las técnicas de prueba de caja blanca es que durante las pruebas se tiene en cuenta toda la implementación del software, lo que facilita la detección de defectos incluso cuando la especificación del software no es clara o está incompleta. El diseño de la prueba es independiente de la especificación. Una debilidad correspondiente es que si el software no implementa uno o más requisitos, las pruebas de caja blanca pueden no detectar defectos resultantes que impliquen falta de funcionalidad [50].

Las técnicas de prueba de caja blanca se pueden utilizar en pruebas estáticas. Son muy adecuados para revisiones de código que aún no está listo para su ejecución [51], así como pseudocódigo y otra lógica de alto nivel que se puede modelar utilizando un diagrama de flujo de control.

Si sólo se realizan pruebas de caja negra, no hay forma de medir la cobertura real del código. Las pruebas de caja blanca proporcionan una medida objetiva de la cobertura y proporcionan la información necesaria para permitir que se generen pruebas adicionales para aumentar esa cobertura y posteriormente aumentar la confianza en el código.

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.

Además de las técnicas de prueba basadas en especificaciones y de caja blanca, existe una tercera familia de técnicas: técnicas de prueba basadas en la experiencia. Esta categoría contiene técnicas que se consideran menos formales que las discutidas anteriormente, donde la base de las acciones del evaluador siempre fue algún modelo formal (un modelo de dominio, lógica, comportamiento, estructura, etc.).

Las técnicas de prueba basadas en la experiencia utilizan principalmente el conocimiento, las habilidades, la intuición y la experiencia de los evaluadores trabajando con productos similares o incluso con el mismo producto anteriormente. Este enfoque facilita que el evaluador identifique fallas o defectos que serían difíciles de detectar utilizando técnicas más estructuradas. A pesar de las apariencias, los evaluadores suelen utilizar técnicas de prueba basadas en la experiencia. Sin embargo, es importante tener en cuenta que la eficacia de estas técnicas, al

su propia naturaleza dependerá en gran medida del enfoque y la experiencia de cada evaluador. El programa de estudios de Foundation Level enumera los siguientes tres tipos de técnicas de prueba basadas en la experiencia, que analizaremos en las siguientes secciones:

- Error al adivinar •
- Pruebas exploratorias •
- Pruebas basadas en listas de verificación

4.4.1 Error al adivinar

Descripción de la técnica La adivinación

de errores es quizás conceptualmente la más simple de todas las técnicas de prueba basadas en la experiencia. No está asociado a ninguna planificación previa, ni es necesario gestionar las actividades del evaluador asociadas al uso de esta técnica. El evaluador simplemente predice los tipos de errores posiblemente cometidos por los desarrolladores, defectos o fallas en un componente o sistema haciendo referencia, entre otras cosas, a los siguientes aspectos:

- Cómo ha funcionado hasta el momento el componente o sistema bajo prueba (o su versión anterior que ya esté funcionando en producción). •
- Cuáles son los errores típicos que conoce el evaluador, cometidos por desarrolladores, arquitectos, analistas y otros miembros del equipo de producción. • Conocimiento
- de Fallos que han ocurrido previamente en aplicaciones similares probadas.
por el evaluador o que el evaluador ha oído hablar
- Los errores, defectos y fallas en general pueden estar relacionados con:
 - Entrada (p. ej., entrada válida no aceptada, entrada no válida aceptada, parámetro incorrecto valor, parámetro de entrada faltante)
 - Salida (p. ej., formato de salida incorrecto, salida incorrecta, salida correcta en el momento equivocado, salida incompleta, salida faltante, errores gramaticales o de puntuación) • Lógica (p. ej., faltan casos a considerar, casos duplicados a considerar, operador lógico no válido, falta condición, iteración de bucle no válida) • Cálculos (p. ej., algoritmo incorrecto, algoritmo ineficaz, cálculo faltante, operando no válido, operador no válido, error de horquillado, precisión insuficiente del resultado, función incorporada no válida)
 - Interfaz (p. ej., procesamiento incorrecto de eventos desde la interfaz, fallas relacionadas con el tiempo en el procesamiento de entrada/salida, llamada a una función incorrecta o inexistente, parámetro faltante, tipos de parámetros incompatibles) • Datos (p. ej., inicialización, definición o declaración de una variable, acceso incorrecto a una variable, valor incorrecto de una variable, uso de una variable no válida, referencia incorrecta a un dato, escala o unidad incorrecta de un dato, dimensión incorrecta de los datos, índice incorrecto, tipo incorrecto de una variable, rango incorrecto de una variable, "error por uno" (ver Apartado 4.2.2))

Fig. 4.15 Calculadora de IMC
(fuente: calculadorasworld.com)

The screenshot shows a mobile application titled "BMI CALCULATOR". At the top, there is a radio button group for "Imperial" and "Metric", with "Metric" selected. Below this are two input fields: "Height" with the value "180" and unit "cm", and "Weight" with the value "80" and unit "kg". A large green box displays the result: "Your BMI is 24.69 (normal BMI)". At the bottom is a blue "Reset" button, and at the very bottom of the screen, the text "Powered by CalculatorsWorld.com".

Existe una variación más organizada y metódica de esta técnica, llamada ataque de falla o ataque de software. Este enfoque implica la creación de una lista de posibles errores, defectos y fallas. El evaluador analiza la lista punto por punto e intenta hacer que cada error, defecto o falla sea apropiadamente visible para el objeto bajo prueba.

Muchos ejemplos de este tipo de ataques se describen en [52-54]. Esta técnica se diferencia de las demás en que su punto de partida es un evento “negativo”, un defecto o falla específica, en lugar de algo “positivo” a verificar (como el dominio de entrada o la lógica de negocios del sistema).

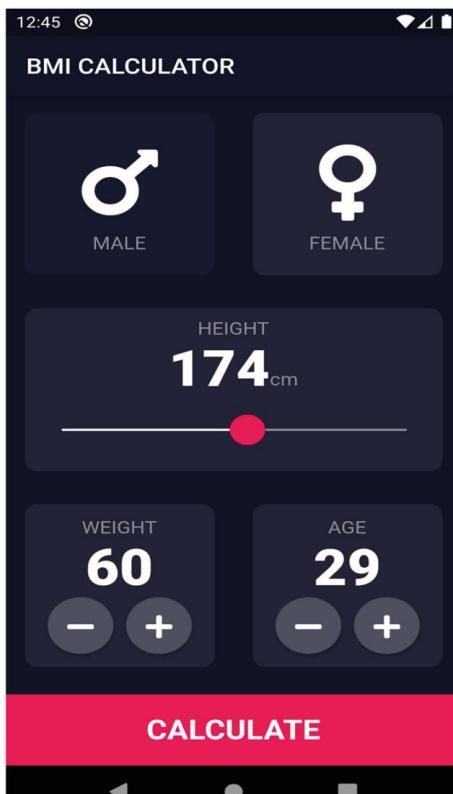
Un evaluador puede crear listas de errores, defectos y fallas basadas en su propia experiencia; entonces serán las más efectivas. También pueden utilizar varios tipos de listas que están disponibles públicamente, por ejemplo, en Internet.

Ejemplo El evaluador está probando un programa para calcular el IMC (índice de masa corporal), que toma dos valores del usuario como entrada, peso y altura, y luego calcula el IMC. La interfaz de la aplicación se muestra en la Fig. 4.15.

El probador utiliza un ataque de falla utilizando la siguiente lista de defectos y fallas:

1. La ocurrencia de un desbordamiento de un campo de formulario.
2. La ocurrencia de la división por cero.
3. Forzar la aparición de un valor no válido.

Fig. 4.16 Calculadora de IMC con una interfaz diferente
 (fuente: <https://www.amazon.com/Meet-shingala-Bmi-calculator/dp/B08FJ4 8KYQ>)



Para intentar forzar una falla desde el elemento 1 de la lista, el evaluador puede intentar ingresar una cadena muy larga de dígitos en el campo de entrada "peso".⁴

Para intentar forzar un fallo en el punto 2 de la lista, el evaluador puede dar una altura cero, ya que el IMC se calcula como el cociente del peso y el cuadrado de la altura.

Para intentar forzar una falla del punto 3, el evaluador puede dar, por ejemplo, un valor de peso negativo para forzar un valor de IMC negativo (incorrecto).

Tenga en cuenta que estos ataques se pueden realizar para la aplicación anterior porque su interfaz permite ingresar valores de peso y altura directamente desde el teclado. Es una buena idea diseñar interfaces de manera que se evite la introducción de valores incorrectos. Por ejemplo, al usar campos con controles de rango de valores incorporados, permitimos al usuario ingresar la entrada solo a través de botones que permiten aumentar o disminuir los valores en un rango controlado. Otra idea es utilizar mecanismos como listas desplegables o controles deslizantes. La Figura 4.16 muestra una interfaz de este tipo para una aplicación de calculadora de IMC análoga.

⁴ La fórmula del IMC divide el peso por el cuadrado de la altura. Para obtener un desbordamiento de los valores del IMC, el numerador de la fracción debe ser lo más grande posible. Por lo tanto, no tiene sentido dar valores grandes para la altura, porque cuanto mayor es el denominador, menor es el IMC.

4.4.2 Pruebas exploratorias

Descripción de la técnica Las pruebas

exploratorias son el enfoque de diseñar, ejecutar y registrar pruebas no escritas (es decir, pruebas que no han sido diseñadas de antemano) y evaluarlas dinámicamente durante la ejecución. Esto significa que el evaluador no ejecuta pruebas preparadas en una fase separada de diseño o análisis, sino que cada uno de sus próximos pasos en el escenario de prueba actual es dinámico y depende de:

- Conocimiento e intuición del evaluador •

Experiencia con esta o aplicaciones similares • La forma

en que se comportó el sistema en el paso anterior del escenario

De hecho, la estricta división entre pruebas exploratorias y programadas no refleja del todo la verdad. Esto se debe a que cada actividad de prueba incluye algún elemento de planificación y algún elemento de dinámica y exploración. Por ejemplo, incluso en pruebas puramente exploratorias, es habitual planificar una sesión exploratoria con antelación, asignando, por ejemplo, un tiempo específico para que el evaluador la execute. Un evaluador exploratorio también puede preparar varios datos de prueba necesarios para la prueba antes de que comience la sesión.

Pruebas exploratorias basadas en sesiones

Las pruebas exploratorias basadas en sesiones son un enfoque que permite a los evaluadores y administradores de pruebas obtener una mayor estructuración de la actividad del evaluador, así como la posibilidad de gestionar mejor las actividades de pruebas exploratorias. Consta de tres pasos básicos:

1. El evaluador se reúne con el gerente para determinar el alcance de la prueba y asignar tiempo. El gerente entrega la carta de prueba (ver Tabla 4.13) al evaluador o la escribe en colaboración con el evaluador. La carta de prueba debe crearse durante el análisis de la prueba (ver Sección 1.4.3).
2. Realizar una sesión de prueba exploratoria por parte del evaluador, durante la cual toma notas de cualquier observación relevante (problemas observados, recomendaciones para el futuro, información sobre lo que falló durante la sesión, etc.).
3. Reunirse nuevamente al final de la sesión, donde se presentan los resultados de la misma.

Se discuten y se toman decisiones sobre posibles próximos pasos.

Las pruebas exploratorias basadas en sesiones se llevan a cabo en un período de tiempo bien definido (generalmente de 1 a 4 h). El evaluador se concentra en la tarea especificada en la carta de prueba, pero, por supuesto, si es necesario, puede desviarse en cierta medida de la tarea esencial, en una situación en la que se observe algún defecto grave en otra área de la aplicación.

El evaluador documenta los resultados de la sesión en la carta de prueba.

Se recomienda realizar pruebas exploratorias basadas en sesiones de forma colaborativa, por ejemplo, mediante emparejamiento. Un evaluador puede asociarse con un representante comercial, un usuario final o un propietario del producto para explorar la aplicación mientras la prueba. Un evaluador también podría asociarse con un desarrollador para evitar probar funciones que son inestables, para abordar cierto problema técnico o para demostrar un comportamiento no deseado de inmediato, sin tener que proporcionar evidencia extensa en el informe de defectos.

Tabla 4.13 Carta de prueba

Carta de prueba—TE 02-001-01	
Meta	Pruebe la funcionalidad de inicio de sesión
Áreas	Inicie sesión como usuario existente con un nombre de usuario y contraseña correctos Inicie sesión como usuario existente a través de una cuenta de Google Inicie sesión como usuario existente a través de una cuenta de Facebook Inicio de sesión incorrecto: ningún usuario Inicio de sesión incorrecto: contraseña incorrecta Acciones que resultan con el bloqueo de la cuenta Uso de la función de recordatorio de contraseña Ataque de inyección SQL y otros ataques a la seguridad
Ambiente	Sitio accesible a través de varios navegadores (Chrome, FF, IE) 2019-06-12, 11:00–
Tiempo	13:15
Ensayador	Bob cazador de insectos
Notas del probador	
Archivos	(Capturas de pantalla, datos de pruebas, etc.).
Defectos encontrados	
división del tiempo	20% preparación para la sesión 70% conducción de la sesión 10% análisis de problemas

^a La inyección SQL es un tipo de ataque que implica la llamada inyección de un código malicioso. Es un ataque a la seguridad al insertar una sentencia SQL maliciosa en un campo de entrada con la intención de ejecutarla.

¿Cuándo utilizar pruebas exploratorias?

Las pruebas exploratorias serán una solución buena, efectiva y eficiente si se cumplen una o más de las siguientes premisas:

- Las especificaciones del producto bajo prueba están incompletas, son de mala calidad o inexistentes. • Hay presión de tiempo; Los evaluadores tienen poco tiempo para realizar las pruebas. • Los evaluadores conocen bien el producto y tienen experiencia en pruebas exploratorias.

Las pruebas exploratorias están fuertemente relacionadas con la estrategia de prueba reactiva (ver Sección 5.1.1).

Las pruebas exploratorias pueden utilizar otras técnicas de caja negra, de caja blanca y basadas en la experiencia.

Nadie puede dictarle al evaluador cómo llevar a cabo la sesión. Si, por ejemplo, al evaluador le resulta útil crear un diagrama de transición de estado que describa cómo funciona el sistema y luego, en una sesión exploratoria, diseña casos de prueba y los ejecuta, tiene derecho a hacerlo. Este es un ejemplo del uso de pruebas programadas como parte de pruebas exploratorias.

Ejemplo La organización planea probar la funcionalidad básica de un sitio web. Se tomó la decisión de realizar pruebas exploratorias y se preparó para el evaluador una carta de prueba que se muestra en la Tabla 4.13. El administrador de pruebas, basándose en los resultados recopilados de múltiples sesiones de pruebas exploratorias, deriva las siguientes métricas de muestra para su posterior análisis:

- Número de sesiones realizadas y completadas • Número de defectos reportados • Tiempo dedicado a prepararse para la sesión • Tiempo de la sesión de prueba real • Tiempo dedicado a analizar problemas • Número de funcionalidades cubiertas

Tours de pruebas exploratorias

Andrew Whittaker [55] propone un conjunto de enfoques de pruebas exploratorias inspirados en las visitas turísticas y el turismo. Esta metáfora permite a los evaluadores aumentar su creatividad al realizar sesiones exploratorias. Los objetivos de estos enfoques son:

- Comprender cómo funciona la aplicación, cómo es su interfaz, qué funcionalidad ofrece al usuario • Obligar al software a demostrar sus capacidades • Encontrar defectos

La metáfora del turista permite dividir las áreas de software de manera análoga a las partes de la ciudad visitadas por el turista:

- Distrito de negocios—corresponde a aquellas partes del software que están relacionadas con su parte “negocio”, es decir, las funcionalidades y características que el software ofrece a los usuarios.
- Distrito histórico (centro histórico): corresponde al código heredado y al historial de funcionalidad defectuosa. • Distrito turístico: corresponde a aquellas partes del software que atraen a nuevos usuarios (turistas), funciones que es poco probable que un usuario avanzado (residente de la ciudad) vuelva a utilizar.
- Distrito de entretenimiento: corresponde a funciones y características de soporte relacionados con la usabilidad y la interfaz de usuario.
- Distrito hotelero: el lugar donde descansa el turista; Corresponde a los momentos en los que el usuario no utiliza activamente el software pero el software aún hace su trabajo.
- Barrios sospechosos: lugares donde es mejor no aventurarse; en el software, corresponden a lugares donde se pueden lanzar varios tipos de ataques contra la aplicación.

Whittaker describe una variedad de tipos de exploración (turismo) para cada distrito.

Para obtener más información sobre las pruebas exploratorias, consulte [56, 57].

4.4.3 Pruebas basadas en listas de verificación

Descripción de la técnica Las pruebas

basadas en listas de verificación, al igual que las dos técnicas anteriores descritas en esta sección, utilizan el conocimiento y la experiencia del evaluador, pero la base para la ejecución de la prueba son los elementos contenidos en la llamada lista de verificación. La lista de verificación contiene las condiciones de prueba que se deben verificar. La lista de verificación no debe contener elementos que puedan verificarse automáticamente, elementos que funcionen mejor como criterios de entrada/salida o elementos que sean demasiado generales [58].

Las pruebas basadas en listas de verificación pueden parecer una técnica similar a los ataques de fallas. La diferencia entre estas técnicas es que un ataque de falla comienza a partir de defectos y fallas, y la acción del evaluador es revelar problemas, dada una falla o defecto en particular.

En las pruebas basadas en listas de verificación, el evaluador también actúa de forma sistemática, pero verifica las características "positivas" del software. La base de prueba en esta técnica es la propia lista de verificación.

Los elementos de la lista de verificación a menudo se formulan en forma de pregunta. La lista de verificación debería permitirle verificar cada uno de sus elementos por separado y directamente. Los elementos de la lista de verificación pueden hacer referencia a requisitos, características de calidad u otras formas de condiciones de prueba. Se pueden crear listas de verificación para admitir varios tipos de pruebas, incluidas pruebas funcionales y no funcionales (por ejemplo, 10 heurísticas para pruebas de usabilidad [59]).

Algunos elementos de la lista de verificación pueden volverse gradualmente menos efectivos con el tiempo a medida que quienes la desarrollan aprenden a evitar cometer los mismos errores. También es posible que sea necesario agregar nuevos elementos para reflejar defectos de alta gravedad que se hayan descubierto recientemente. Por lo tanto, las listas de verificación deben actualizarse periódicamente en función del análisis de defectos. Sin embargo, se debe tener cuidado de que la lista de verificación no sea demasiado larga [60].

En ausencia de casos de prueba detallados, las pruebas basadas en listas de verificación pueden proporcionar cierto grado de coherencia a las pruebas. Dos evaluadores que trabajen con la misma lista de verificación probablemente realizarán su tarea de manera ligeramente diferente, pero en general probarán las mismas cosas (las mismas condiciones de prueba), por lo que necesariamente sus pruebas serán similares. Si las listas de verificación son de alto nivel, es probable que haya cierta variabilidad en las pruebas reales, lo que dará como resultado una cobertura potencialmente mayor pero una menor repetibilidad de las pruebas.

Tipos de listas de verificación

Existen muchos tipos diferentes de listas de verificación para diferentes aspectos del software. Además, las listas de verificación pueden tener distintos grados de generalidad y un campo de aplicación más amplio o más limitado. Por ejemplo, las listas de verificación para el uso de pruebas de código (por ejemplo, en pruebas de componentes) tenderán a ser muy detalladas y normalmente incluirán muchos detalles técnicos sobre aspectos del desarrollo de código en un lenguaje de programación particular. Por el contrario, una lista de verificación para pruebas de usabilidad puede ser de muy alto nivel y general. Por supuesto, esto no es una regla: los evaluadores siempre deben ajustar el nivel de detalle de la lista de verificación para adaptarla a sus propias necesidades.

Las listas de

verificación de aplicaciones se pueden utilizar básicamente para cualquier tipo de prueba. En particular, pueden aplicarse a pruebas funcionales y no funcionales.

En las pruebas basadas en listas de verificación, el evaluador diseña, implementa y ejecuta pruebas para cubrir las condiciones de prueba que se encuentran en la lista de verificación. Los evaluadores pueden utilizar listas de verificación existentes (por ejemplo, disponibles en Internet) o pueden modificarlas y adaptarlas a sus necesidades. También pueden crear dichas listas ellos mismos, basándose en su propia experiencia y la de su organización con defectos y fallas, su conocimiento de las expectativas de los usuarios del producto desarrollado o su conocimiento de las causas y síntomas de las fallas del software. Hacer referencia a la propia experiencia puede hacer que esta técnica sea más eficaz, porque normalmente, las mismas personas, en la misma organización, trabajando en productos similares, cometerán errores similares. Normalmente, los evaluadores menos experimentados trabajan con listas de verificación ya existentes.

Cobertura

Para una prueba basada en una lista de verificación, no se definen medidas específicas de cobertura. Por supuesto, como mínimo, se debe cubrir cada elemento de la lista de verificación. Sin embargo, dado que es difícil saber hasta qué punto se ha cubierto dicho elemento (debido al hecho de que la técnica apela al conocimiento, la experiencia y la intuición del evaluador individual), esto tiene ventajas y desventajas.

La desventaja, por supuesto, es la falta de información detallada sobre la cobertura y una menor repetibilidad que con las técnicas formales. La ventaja, por otro lado, es que se puede lograr una mayor cobertura si dos o más evaluadores utilizan la misma lista de pruebas.

Esto se debe a que lo más probable es que cada uno de ellos realice un conjunto de pasos ligeramente diferente para cubrir el mismo elemento de la lista de verificación. Así, habrá cierta variabilidad en las pruebas, lo que se traducirá en una mayor cobertura pero a costa de una menor repetibilidad.

Ejemplo A continuación presentamos dos listas de verificación de muestra. El primero contiene las denominadas heurísticas de usabilidad de Nielsen. Junto a cada elemento de la lista de verificación, se proporciona además un comentario entre paréntesis, que puede ayudar al evaluador con las pruebas.

Heurísticas de Nielsen para la usabilidad del sistema

1. Visibilidad del estado del sistema: ¿se muestra el estado del sistema en cada punto de su funcionamiento? (El usuario siempre debe saber dónde se encuentra; el sistema debe utilizar las llamadas rutas de navegación, que muestran el camino que ha seguido el usuario, así como títulos claros para cada pantalla)
2. Coincidencia entre el sistema y el mundo real: ¿Existe compatibilidad entre el sistema y la realidad? (La aplicación no debe utilizar un lenguaje técnico sino un lenguaje sencillo utilizado todos los días por los usuarios del sistema, relacionado con el dominio empresarial en el que opera el programa)
3. Control y libertad del usuario: ¿tiene el usuario control sobre el sistema? (por ejemplo, debería ser posible deshacer una acción que el usuario realizó por error, como eliminar del carrito de compras un producto colocado allí por error)
4. Coherencia y estándares: ¿mantiene el sistema coherencia y estándares? (Las soluciones de apariencia deben ser consistentes en toda la aplicación, por ejemplo, el mismo formato de enlaces, uso de la misma fuente; también se deben utilizar enfoques familiares, por ejemplo, colocar el logotipo de la empresa en la esquina superior izquierda de la pantalla)

5. Prevención de errores: ¿el sistema previene adecuadamente los errores? (El usuario no debe tener la impresión de que algo ha salido mal; por ejemplo, no debemos darle la opción al usuario de seleccionar una versión de un producto que no está disponible en la tienda, sólo para ver más tarde un error de "no stock").
6. Reconocimiento en lugar de recuerdo: ¿el sistema te permite seleccionar en lugar de obligarte a recordar? (por ejemplo, las descripciones de los campos en un formulario no deben desaparecer cuando comienza a completarlos; esto sucede cuando la descripción se coloca inicialmente dentro de ese campo)
7. Flexibilidad y eficiencia de uso: ¿el sistema proporciona flexibilidad y eficiencia? (por ejemplo, las opciones de búsqueda avanzada deberían estar ocultas de forma predeterminada si la mayoría de los usuarios no las utilizan)
8. Diseño estético y minimalista: ¿el sistema es estéticamente agradable y no está sobrecargado de contenido? (La aplicación debe atraer a los usuarios; debe tener un buen diseño, combinación de colores adecuada, disposición de los elementos de la página, etc.).
9. Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores: ¿se proporciona un manejo eficaz de errores? (una vez que ocurre un error, el usuario no debe recibir un mensaje técnico al respecto o que es culpa del usuario; también debe haber información sobre lo que debe hacer el usuario en esta situación)
10. Ayuda y documentación: ¿hay ayuda disponible en el sistema? ¿Existe documentación de usuario? (Algunos usuarios necesitarán una función de ayuda; dicha opción debería estar disponible en la aplicación; también debería incluir información de contacto para soporte técnico)

El segundo ejemplo de lista de verificación se refiere a la revisión del código. Esta lista, a modo de ejemplo, incluye sólo aspectos técnicos seleccionados de buenas prácticas de escritura de código y está lejos de ser completa.

Lista de verificación para inspecciones de códigos

1. ¿El código está escrito según los estándares actuales?
2. ¿El código tiene un formato coherente?
3. ¿Hay funciones que nunca se llaman?
4. ¿Son eficientes los algoritmos implementados y cuentan con recursos computacionales adecuados? ¿complejidad?
5. ¿Se utiliza la memoria de forma eficaz?
6. ¿Existen variables que se utilizan sin haber sido declaradas primero?
7. ¿Está el código debidamente documentado?
8. ¿Es coherente la forma de comentar?
9. ¿Toda operación de división está protegida contra la división por cero?
10. En las declaraciones IF-THEN, ¿son los bloques de declaraciones que se ejecutan con mayor frecuencia? comprobado primero?
11. ¿Cada declaración CASE tiene un bloque predeterminado?
12. ¿Se libera alguna memoria asignada cuando ya no está en uso?

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 empresariales.

FL-4.5.2 (K2) Clasificar las diferentes opciones para redactar criterios de aceptación.

FL-4.5.3 (K2) Utilice el desarrollo basado en pruebas de aceptación (ATDD) para derivar pruebas casos.

La popularidad de las metodologías ágiles ha llevado al desarrollo de métodos de prueba específicos para este enfoque, teniendo en cuenta los artefactos utilizados por estas metodologías y enfatizando la colaboración entre clientes (negocios), desarrolladores y evaluadores. Este capítulo analiza las siguientes cuestiones relacionadas con las pruebas en el contexto de metodologías ágiles, utilizando un enfoque de prueba basado en la colaboración:



- Historias de usuarios como contraparte ágil de los requisitos del usuario (Sección 4.5.1)
- Criterios de aceptación como contraparte ágil de las condiciones de prueba, proporcionando la base para el diseño de pruebas (Sección 4.5.2)
- Desarrollo basado en pruebas de aceptación (ATDD) como una forma de prueba de alto nivel (pruebas de sistema y pruebas de aceptación) que se utiliza a menudo en metodologías ágiles, basadas en un enfoque de "prueba primero" (Sección 4.5.3).

Cada una de las técnicas descritas en las Secciones. 4.2, 4.3 y 4.4 tienen un objetivo específico con respecto a la detección de defectos de un tipo específico. Los enfoques colaborativos, por otro lado, también se centran en evitar defectos mediante la cooperación y la comunicación.

4.5.1 Escritura colaborativa de historias de usuario

En el desarrollo ágil de software, una historia de usuario representa un incremento funcional que será de valor para el usuario, el comprador del sistema o software, o cualquier otra parte interesada. Las historias de usuarios se escriben para capturar los requisitos desde la perspectiva de desarrolladores, evaluadores y representantes comerciales. En los modelos SDLC secuenciales, esta visión compartida de una característica o función de software específica se logra mediante revisiones formales después de que se hayan escrito los requisitos. En los enfoques ágiles, por otro lado, la visión compartida se logra a través de revisiones informales frecuentes durante la redacción de requisitos o escribiendo requisitos juntos, de manera colaborativa, por parte de evaluadores, analistas, usuarios, desarrolladores y cualquier otra parte interesada. Las historias de usuario constan de tres aspectos, conocidos como las "3C":

- Tarjeta
- Conversación •
- Confirmación

Tarjeta

Una tarjeta es un medio que describe la historia de un usuario (por ejemplo, puede ser una tarjeta física en forma de una hoja de papel colocada en un tablero scrum⁵ o una entrada en un tablero electrónico). La tarjeta identifica el requisito, su importancia o prioridad, las limitaciones, el tiempo esperado de desarrollo y prueba y, muy importante, los criterios de aceptación de la historia. La descripción debe ser precisa, ya que se utilizará en la cartera de productos.

Los equipos ágiles pueden documentar historias de usuarios de diversas formas. Un formato popular es:

Como (usuario previsto)
Quiero (acción prevista), de
modo que (propósito/resultado de la acción, beneficio obtenido),
seguido de los criterios de aceptación. Estos criterios también se pueden documentar de diferentes maneras (ver Sección 4.5.2). Independientemente del enfoque adoptado para documentar historias de usuarios, la documentación debe ser concisa y suficiente para el equipo que la implementará y probará.

Las tarjetas representan los requisitos del cliente en lugar de documentarlos. Si bien la tarjeta puede contener el texto de la historia, los detalles se resuelven en la conversación y se registran en la confirmación [61].

Conversación La

conversación explica cómo se utilizará el software. La conversación puede ser documentada o verbal. Los evaluadores, que tienen un punto de vista diferente al de los desarrolladores y representantes comerciales, realizan valiosas contribuciones al intercambio de pensamientos, opiniones y experiencias. La conversación comienza durante la fase de planificación del lanzamiento y continúa cuando se planifica la historia. Se lleva a cabo entre partes interesadas con tres perspectivas principales sobre el producto: el cliente/usuario, el desarrollador y el evaluador.

Confirmación La

confirmación se presenta en forma de criterios de aceptación, que representan elementos de cobertura que transmiten y documentan los detalles de la historia de un usuario y que pueden usarse para determinar cuándo está completa una historia. Los criterios de aceptación suelen ser el resultado de una conversación. Los criterios de aceptación pueden verse como condiciones de prueba que el evaluador debe verificar para verificar que la historia esté completa.

Las historias de usuarios deben abordar características tanto funcionales como no funcionales. Normalmente, la perspectiva única del evaluador mejorará la historia del usuario al identificar detalles faltantes o requisitos no funcionales. El evaluador puede brindar información haciendo preguntas abiertas a los representantes comerciales sobre la historia del usuario, sugiriendo formas de probarla y confirmando los criterios de aceptación.

La autoría compartida de historias de usuarios puede utilizar técnicas como la lluvia de ideas o los mapas mentales.⁶ Las buenas historias de usuario cumplen con las llamadas propiedades INVEST, es decir, son [62]:

⁵ En Scrum, un tablero de scrum es un tablero físico de uso opcional que muestra el estado actual de una iteración. Proporciona una representación visual del cronograma de trabajo a realizar en una iteración determinada.

⁶ Para obtener más información sobre mapas mentales, consulte, por ejemplo, <https://www.mindmapping.com/>.

- Independientes: no se superponen y pueden desarrollarse en cualquier orden. • Negociables: no son contratos de características explícitas; más bien, los detalles serán creados conjuntamente por el cliente y el desarrollador durante el desarrollo.
- Valiosos: una vez implementados, deberían aportar valor agregado al cliente. • Estimable: el cliente debería poder priorizarlos y el equipo debería poder estimar su tiempo de finalización para gestionar más fácilmente el proyecto y optimizar los esfuerzos del equipo.
- Pequeño: esto hace que su alcance sea fácil de entender para el equipo y la creación de las historias mismas son más manejables.
- Comprobable: ésta es una característica general de un buen requisito; la corrección de la implementación de la historia debe ser fácilmente verificable.

Si un cliente no sabe cómo probar algo, puede indicar que la historia no es lo suficientemente clara o que no refleja algo de valor para el cliente o que el cliente simplemente necesita ayuda con las pruebas [62].

Ejemplo El siguiente ejemplo muestra una historia de usuario escrita desde la perspectiva de un cliente de una aplicación web de banca electrónica. Tenga en cuenta que los criterios de aceptación no necesariamente se derivan directamente del contenido de la historia en sí, sino que pueden ser el resultado de una conversación entre el equipo y los clientes. El evaluador utiliza estos criterios para diseñar pruebas de aceptación que verificarán que la historia se haya implementado total y correctamente.

Historia de usuario US-001-03

Como cliente del banco

Quiero poder iniciar sesión en el sistema.

Para poder utilizar productos bancarios

Criterios de aceptación

- El inicio de sesión debe ser una dirección de correo electrónico válida. • El sistema rechaza el intento de inicio de sesión con una contraseña incorrecta. • El sistema rechaza el intento de inicio de sesión por un inicio de sesión de usuario inexistente. • El sistema permite al usuario ingresar en los campos "login" y "contraseña" únicamente caracteres alfanuméricos, así como un punto y el símbolo "@".
- Los campos de inicio de sesión y contraseña pueden tener hasta 32 caracteres. • Después de hacer clic en el enlace "recordar contraseña" e ingresar la dirección de correo electrónico del usuario, se envía a esta dirección un enlace al sistema de recordatorio de contraseña. • El sistema inicia el proceso de inicio de sesión cuando se hace clic en el botón "iniciar sesión" o cuando se presiona la tecla Enter; en el último caso, el proceso de inicio de sesión comienza cuando se cumplen tres condiciones simultáneamente: (1) el campo "iniciar sesión" no está vacío, (2) la ventana activa es la ventana "contraseña", (3) el campo "contraseña" no está vacío.

4.5.2 Criterios de aceptación

Los criterios de aceptación  son condiciones que un producto (en la medida descrita por la historia del usuario o el Elemento de la cartera de productos del que forman parte estos criterios de aceptación) debe cumplir para ser aceptado por el cliente. Desde esta perspectiva, los criterios de aceptación pueden verse como condiciones de prueba o elementos de cobertura que deben verificarse mediante pruebas de aceptación.

Se utilizan criterios de aceptación:

- Definir los límites de la historia del usuario • Llegar a un consenso entre el equipo de desarrollo y el cliente • Describir escenarios de prueba tanto positivos como negativos • Como base para las pruebas de aceptación de la historia del usuario (ver Sección 4.5.3) • Como herramienta para una planificación y estimación precisas

Los criterios de aceptación pueden considerarse de gran ayuda para determinar y evaluar la Definición de Listo (DoR) o Definición de Hecho (DoD). Un equipo puede decidir no iniciar la implementación cuando los criterios de aceptación de una historia de usuario no se obtienen de manera exhaustiva. De manera similar, un equipo puede decidir que una Historia de Usuario no se considera candidata para lanzamiento (o demostración) cuando no se cumplen todos los Criterios de Aceptación (cobertura de los Criterios de Aceptación por debajo del 100%).

Los criterios de aceptación se discuten durante la conversación (ver Sección 4.5.1) y se definen en colaboración entre representantes comerciales, desarrolladores y evaluadores.

Los criterios de aceptación, si se cumplen, se utilizan para confirmar que la historia del usuario se ha implementado completamente y de acuerdo con la visión compartida de todas las partes interesadas. Proporcionan a los desarrolladores y evaluadores una visión ampliada de la función que validarán los representantes comerciales (o sus representantes). Se deben utilizar pruebas tanto positivas como negativas para cubrir los criterios. Durante la confirmación, diferentes partes interesadas desempeñan el papel de probador. Estos pueden variar desde desarrolladores hasta especialistas centrados en el rendimiento, la seguridad, la interoperabilidad y otros atributos de calidad. El equipo ágil considera que una tarea está completa cuando se considera que se ha cumplido un conjunto de criterios de aceptación.

No existe una forma única y establecida de escribir criterios de aceptación para una historia de usuario. Los dos formatos más comunes son:

- Criterios de aceptación orientados a escenarios. • Criterios de aceptación orientados a reglas.

Criterios de aceptación orientados a escenarios Este formato para escribir criterios de aceptación suele utilizar el formato Dado/Cuándo/Entonces conocido por la técnica BDD. Sin embargo, en algunos casos, resulta difícil encajar los criterios de aceptación en dicho formato, por ejemplo, cuando el público objetivo no necesita los detalles exactos de los casos de prueba. En ese caso, se puede utilizar un formato orientado a reglas.

Criterios de aceptación orientados a reglas

Normalmente, en este formato, los criterios de aceptación toman la forma de una lista de verificación con viñetas o una forma tabular de asignación de entradas a salidas. Algunos frameworks y lenguajes

para escribir los criterios de estilo Dado/Cuándo/Entonces (p. ej., Gherkin) proporcionan mecanismos que Le permite crear un conjunto de reglas dentro de un escenario, cada una de las cuales será probada. por separado (esta es una forma de prueba basada en datos).

La mayoría de los criterios de aceptación se pueden documentar en uno de los dos formatos mencionados. arriba. Sin embargo, el equipo puede utilizar cualquier otro formato no estándar, siempre y cuando el Los criterios de aceptación están bien definidos y son inequívocos.

Junto a los criterios de aceptación orientados a escenarios y a reglas mencionados arriba, los equipos menos maduros generalmente obtienen y documentan los criterios de aceptación de forma gratuita. formato, como adiciones o subsecciones a una Historia de Usuario. Cualquiera que sea el formato que decida un equipo de uso, la trazabilidad bidireccional entre la historia de usuario y sus criterios de aceptación Es vital.

Ejemplo La historia de usuario descrita en la sección anterior (iniciar sesión en el sistema de banca electrónica) tiene criterios de aceptación descritos en forma de reglas. Algunos de pueden detallarse y representarse en forma de mapeo de entradas y salidas mediante definir ejemplos específicos de datos de prueba. Por ejemplo, en Gherkin, los primeros tres puntos de la lista de criterios de aceptación para esta historia:

- El inicio de sesión debe ser una dirección de correo electrónico válida.
- El sistema rechaza el intento de inicio de sesión con una contraseña incorrecta.
- El sistema rechaza el intento de inicio de sesión por un inicio de sesión de usuario inexistente.

se puede especificar de la siguiente manera:

Esquema del escenario: inicios de sesión y contraseñas correctos e incorrectos

El usuario dado ingresa <login> como inicio de sesión

Y el usuario ingresa <contraseña> como contraseña

Cuando el usuario hace clic en el botón "iniciar sesión"

Entonces el resultado de inicio de sesión es <resultado>.

Ejemplos:

```
| iniciar sesión | contraseña | resultado |
| edina.monsoon@gmail.com | abFab | Aceptar |
| patsy.stone@gmail | martini | NO ESTÁ BIEN |
| azafrán@mail.abc.com | NO ESTÁ BIEN |
| mods→london@gmail.com | balanceándose | NO ESTÁ BIEN |
|| algunaContraseña | NO ESTÁ BIEN |
```

En el ejemplo anterior, hay referencias a variables (entre paréntesis "<" y ">") en la descripción de los criterios de aceptación presentados en formato Dado/Cuándo/Entonces, y los datos de prueba específicos se encuentran en la sección Ejemplos a continuación. Esta prueba se ejecutará cinco veces, cada vez con un conjunto diferente de datos de prueba. El nombre de la sección donde los datos de la prueba se colocan sugiere que las pruebas están en forma de ejemplos, que muestran en datos específicos cómo se supone que debe comportarse el sistema. Por ejemplo, un inicio de sesión es Se supone que es correcto si el nombre de usuario y la contraseña cumplen con las condiciones establecidas (el nombre de usuario es un La dirección de correo electrónico, el nombre de usuario y la contraseña válidos no están vacíos y contienen únicamente

caracteres alfanuméricos). Por otro lado, si la dirección de correo electrónico no es válida (línea 2), la contraseña está en blanco (línea 3), se utilizan caracteres no alfanuméricos (línea 4) o la El inicio de sesión está en blanco (línea 5), el inicio de sesión no se realizará correctamente.

Ejemplo Consideremos otro ejemplo de una historia de usuario, esta vez para un CRM. sistema para un determinado banco. La historia se refiere a la implementación de ciertos negocios. reglas relacionadas con la oferta de tarjetas de crédito a clientes que cumplan ciertos requisitos.

Como institución financiera

Quiero asegurarme de que sólo los clientes con ingresos anuales suficientes obtengan un tarjeta de crédito

Para que no se ofrezcan tarjetas de crédito a clientes que no podrán pagar el débito en la tarjeta

Escenario: Hay dos tipos de tarjetas: una con un límite de débito de \$2500, la otra con un límite de débito de \$5000. El límite máximo de la tarjeta de crédito depende del ganancias del cliente (redondeadas al dólar más cercano). El cliente debe tener un salario. de más de \$10,000 por mes para obtener el límite de débito más bajo. Si el salario excede \$15,000 por mes, el cliente obtiene un límite de débito más alto.

Dado un cliente con ganancias mensuales <monto de ganancias>.

Cuando un cliente solicita una tarjeta de crédito

Entonces la solicitud debe ser <aceptada> o <rechazada> y si es aceptada, el límite máximo de la tarjeta de crédito debe ser <límite máximo>.

En la Tabla 4.14 se muestran ejemplos de casos de prueba escritos basados en esta historia . Nota que en este ejemplo, el evaluador, mientras crea casos de prueba para verificar la regla descrita en la historia, al mismo tiempo intentó cubrir los valores límite del "salario mensual" dominio.

4.5.3 Desarrollo basado en pruebas de aceptación (ATDD)

El desarrollo basado en pruebas de aceptación (ATDD) es un enfoque que da prioridad a las pruebas (consulte Secta. 2.1.3). Los casos de prueba se crean antes de implementar la historia del usuario. Casos de prueba son creados por miembros del equipo con diferentes perspectivas sobre el producto, como clientes, desarrolladores y evaluadores [63]. Los casos de prueba pueden ser manuales o automatizados.

El primer paso es el llamado taller de especificación, durante el cual los miembros del equipo analizan, discuten y escriben la historia del usuario y sus criterios de aceptación. Durante este proceso, todo tipo de problemas en la historia, como falta de completitud, ambigüedades,

Cuadro 4.14 Resultados empresariales

Ganancias de CT	Resultado esperado	Límite máximo	Comentarios
1	\$10 000	Rechazado	\$0
2	\$10 001	Aceptado	\$2500
3	\$15 000	Aceptado	\$2500
4	\$15 001	Aceptado	\$5000

se solucionan contradicciones u otro tipo de defectos. El siguiente paso es crear pruebas.

Esto lo puede hacer colectivamente un equipo o individualmente un evaluador. En cualquier caso, una persona independiente, como un representante comercial, realiza la validación de las pruebas. Las pruebas son ejemplos, basados en criterios de aceptación, que describen características específicas de la historia del usuario (ver Sección 4.5.2). Estos ejemplos ayudan al equipo a implementar la historia del usuario correctamente. Debido a que los ejemplos y las pruebas son lo mismo, los términos a menudo se usan indistintamente.

Una vez diseñadas las pruebas, las técnicas de prueba descritas en las Secciones. Se pueden aplicar 4.2, 4.3 y 4.4 . Normalmente, las primeras pruebas son positivas, confirman el comportamiento correcto sin excepciones ni errores, e implican una secuencia de acciones realizadas si todo sale como se esperaba. Se dice que este tipo de escenarios implementan los llamados caminos felices, es decir, caminos de ejecución donde todo va según lo planeado sin que se produzcan fallas. Después de ejecuciones de pruebas positivas, el equipo debe realizar pruebas negativas y pruebas relacionadas con los atributos no funcionales (por ejemplo, rendimiento, usabilidad). Las pruebas deben expresarse en términos que las partes interesadas puedan entender. Normalmente, las pruebas incluyen oraciones en lenguaje natural que contienen las condiciones previas necesarias (si las hay), entradas y salidas asociadas.

Los ejemplos (es decir, pruebas) deben cubrir todas las características de la historia del usuario y no deben ir más allá. Sin embargo, los criterios de aceptación pueden detallar algunos de los problemas descritos en la historia del usuario. Además, no deben haber dos ejemplos que describan las mismas características de la historia del usuario (es decir, las pruebas no deben ser redundantes).

Cuando las pruebas se escriben en un formato compatible con el marco de automatización de pruebas de aceptación, los desarrolladores pueden automatizar estas pruebas escribiendo código de soporte durante la implementación de la función descrita por la historia del usuario. Las pruebas de aceptación se convierten entonces en requisitos ejecutables. Un ejemplo de dicho código de soporte se muestra en la Sección. 2.1.3 al discutir el enfoque BDD.

Ejemplo Supongamos que un equipo tiene la intención de probar la siguiente historia de usuario.

Historia de usuario US-002-02

Como cliente bancario registrado

Quiero poder transferir dinero a otra cuenta

Para poder transferir fondos entre cuentas.

Criterios de aceptación

- (AC1) el monto de la transferencia no puede exceder el saldo de la cuenta • (AC2) la cuenta de destino debe ser correcta • (AC3) para datos válidos (monto, números de cuenta) el saldo de la cuenta de origen disminuye y el saldo de la cuenta de destino aumenta en el monto de la transferencia

- (AC4) el monto de la transferencia debe ser positivo y representar la cantidad correcta de dinero (es decir, tener una precisión máxima de dos decimales)

Teniendo en cuenta que las pruebas deben verificar que se cumplan los criterios de aceptación, los ejemplos de pruebas funcionales positivas aquí podrían ser las siguientes pruebas relacionadas con (AC3) y verificando también (AC4):

- TC1: realizar una transferencia “típica”, por ejemplo, una transferencia por \$1500 desde una cuenta con un saldo de \$2735.45 a otra cuenta correcta; resultado esperado: saldo de la cuenta = \$1235.45, el saldo de la cuenta objetivo aumenta en \$1500
- TC2: realizar una transferencia correcta de todo el saldo de la cuenta a otra cuenta correcta (por ejemplo, una transferencia por \$21,37 de una cuenta con un saldo de \$21,37 a otra cuenta correcta); Resultado esperado: saldo de la cuenta = \$0, el saldo de la cuenta objetivo aumenta en \$21,37

El segundo caso de prueba utiliza el análisis de valores límite para la diferencia entre los saldo de la cuenta y el monto de la transferencia.

El siguiente paso es crear pruebas negativas, teniendo en cuenta que las pruebas son para verificar que se cumplan los criterios de aceptación. Por ejemplo, un evaluador podría considerar las siguientes situaciones:

- TC3: intentar realizar una transferencia a otra cuenta correcta, cuando el monto de la transferencia es mayor que el saldo de la cuenta de origen (verificación de AC1) • TC4: intentar realizar una transferencia con el monto y saldo de la transferencia correctos, pero a una cuenta inexistente (verificación de AC2)
- TC5: intentar realizar una transferencia con el monto y saldo de transferencia correctos a la misma cuenta (verificación de AC2)
- TC6: intento de realizar una transferencia por importe incorrecto (verificación de AC4)

Por supuesto, para cada una de estas pruebas, el evaluador puede definir una serie de datos de prueba para verificar el comportamiento específico del programa. Al crear estos casos de prueba, el evaluador puede utilizar técnicas de caja negra (por ejemplo, partición de equivalencia o análisis de valores límite). Por ejemplo, para TC1, vale la pena considerar transferencias por los siguientes montos: el mínimo posible (p. ej., \$0,01), “típico” (p. ej., \$1500), montos con diferentes niveles de precisión (p. ej., \$900,2, \$8321,06), o un cantidad muy grande (por ejemplo, \$8,438,483,784).

Para TC3, por otro lado, vale la pena comprobar las siguientes situaciones (aquí utilizamos la partición de equivalencia y el análisis de valores límite):

- Cuando el monto de la transferencia es mucho mayor que el saldo de la cuenta • Cuando el monto de la transferencia es mayor en 1 centavo (el incremento mínimo posible) que el saldo de la cuenta

Para TC4, una cuenta no válida se puede representar como:

- Una cadena de caracteres vacía • Un número con la estructura correcta (26 dígitos), pero que no corresponde a ninguna cuenta física • Un número con estructura incorrecta: demasiado corto (p. ej., 25 dígitos) • Un número con estructura incorrecta: demasiado largo (p. ej., 27 dígitos) • Un número con estructura no válida: contiene caracteres prohibidos, como letras

Finalmente, para TC6, vale la pena considerar en particular:

- Un número negativo que representa la cantidad correcta (por ejemplo, -\$150,25) • Número 0 • Una cadena de caracteres que contiene letras (por ejemplo, \$15B.20)

- Un número representado como resultado de una operación matemática (por ejemplo, \$15+\$20)
- Un número con más de dos decimales (por ejemplo, \$0,009)

Tenga en cuenta que cada uno de los conjuntos de datos de prueba proporcionados anteriormente verifica un riesgo potencial significativamente diferente que existe en el sistema que en los otros casos. Por tanto, las pruebas no son redundantes; no marcan "lo mismo".

Preguntas de muestra

Pregunta 4.1

(FL-4.1.1, K2)

Diseño de pruebas, implementación de pruebas y ejecución de pruebas basadas en la entrada del software.

El análisis de dominio es un ejemplo de:

- A. Técnica de prueba de caja blanca.
- B. Técnica de prueba de caja negra.
- C. Técnica de prueba basada en la experiencia.
- D. Técnica de prueba estática.

Elija una respuesta.

Pregunta 4.2

(FL-4.1.1, K2)

¿Cuál es una característica común de técnicas como la partición de equivalencia, el estado?

¿Pruebas de transición o tablas de decisión?

- R. Al utilizar estas técnicas, las condiciones de prueba se derivan en función de la información sobre la estructura interna del software bajo prueba.
- B. Para proporcionar datos de prueba para casos de prueba diseñados en base a estas técnicas, el evaluador debe analizar el código fuente.
- C. La cobertura en estas técnicas se mide como la proporción de elementos del código fuente probados (por ejemplo, declaraciones) con respecto a todos los elementos identificados en el código.
- D. Los casos de prueba diseñados con estas técnicas pueden detectar discrepancias entre requisitos y su implementación real.

Elija una respuesta.

Pregunta 4.3

(FL-4.2.1, K3)

El sistema asigna un descuento por compras dependiendo del monto de las compras expresado en \$, que es un número positivo con dos decimales. Las compras hasta el monto de \$99.99 no dan derecho a descuento. Las compras de \$100 a \$299,99 dan derecho a un 5% de descuento. Las compras superiores a \$299,99 tienen derecho a un 10% de descuento.

Indique el conjunto mínimo de valores (dados en \$) para lograr una técnica de partición del 100% de equivalencia.

- R. 0,01; 100,99; 500.
- B. 99,99; 100; 299,99; 300.
- C.1; 99; 299.
- D.0; 5; 10.

Elija una respuesta.

Pregunta 4.4

(FL-4.2.1, K3)

La máquina expendedora de café acepta monedas de 25 céntimos, 50 céntimos y 1 dólar. El café cuesta 75c. Después de insertar la primera moneda, la máquina espera hasta que la cantidad de monedas insertadas por el usuario sea igual o superior a 75c. Cuando esto sucede, se bloquea la ranura para monedas, se dispensa café y (si es necesario) se da cambio. Supongamos que la máquina siempre tiene suficientes números y denominaciones de monedas para dar cambio. Considere los siguientes escenarios de prueba:

Escenario 1:

Inserte monedas en orden: 25c, 25c, 25c.

Comportamiento esperado: la máquina expendedora dispensa café y no da cambio.

Escenario 2:

Inserte monedas en orden: 25c, 50c.

Comportamiento esperado: la máquina expendedora dispensa café y no da cambio.

Escenario 3:

Inserte una moneda de \$1.

Comportamiento esperado: la máquina expendedora dispensa café y cambio de 25 céntimos.

Escenario 4:

Inserte monedas en orden: 25c, 25c, 50c.

Comportamiento esperado: la máquina expendedora dispensa café y cambio de 25 céntimos.

Quiere comprobar si la máquina realmente da cambio cuando el usuario ha insertado monedas por más de 75 céntimos y si no da ningún cambio si el usuario ha insertado exactamente 75 céntimos.

¿Cuál de estos escenarios representa el conjunto mínimo de casos de prueba que logra este objetivo?

- A. Escenario 1, Escenario 2.
- B. Escenario 3, Escenario 4.
- C. Escenario 1, Escenario 3.
- D. Escenario 1, Escenario 2, Escenario 3.

Elija una respuesta.

Pregunta 4.5

(FL-4.2.1, K3)

Estás probando un sistema de gestión de tarjetas que te da derecho a descuentos en compras. Hay cuatro tipos de tarjetas, regulares, plateadas, doradas y diamantes, y tres posibles descuentos: 5%, 10% y 15%. Se utiliza la partición de equivalencia.

Técnica para comprobar si el sistema funciona correctamente para todo tipo de tarjetas y todos los descuentos posibles. Ya tienes preparados los siguientes casos de prueba:

- TC1: tarjeta normal, descuento: 10%
- TC2: tarjeta plata, descuento: 15%
- TC3: tarjeta oro, descuento: 15%
- TC4: tarjeta plata, descuento: 10%

¿Cuál es la MENOR cantidad de casos de prueba que debe preparar ADICIONALMENTE?

¿Cómo lograr una cobertura del 100% en "cada opción" tanto para tipos de tarjetas como para tipos de descuento?

- A. 1
- B. 3
- C. 2
- D. 8

Elija una respuesta.

Pregunta 4.6

(FL-4.2.2, K3)

El usuario define la contraseña ingresándola en el campo de texto y haciendo clic en el botón "Confirmar". De forma predeterminada, el campo de texto está en blanco al principio. El sistema considera correcto el formato de la contraseña si la misma tiene al menos 6 caracteres y no más de 11 caracteres. Identificó tres particiones de equivalencia: contraseña demasiado corta, longitud de contraseña correcta y contraseña demasiado larga. Los casos de prueba existentes representan un conjunto mínimo de casos de prueba que logran una cobertura BVA de 2 valores del 100 %. El director de pruebas decidió que, debido a la criticidad del componente sometido a prueba, sus pruebas deberían lograr una cobertura BVA de 3 valores del 100 %.

Contraseñas de qué longitudes deben probarse ADICIONALMENTE para lograr el
cobertura requerida?

- A. 5, 12 B.
- 0, 5, 12 C. 4,
- 7, 10 D. 1, 4,
- 7, 10, 13

Elija una respuesta.

Pregunta 4.7

(FL-4.2.2, K3)

Los usuarios del lavadero disponen de tarjetas electrónicas que registran cuántas veces han utilizado el lavadero hasta el momento. El lavado de coches ofrece una promoción: cada décimo lavado es gratuito. Está probando la corrección de ofrecer la promoción utilizando una técnica de análisis de valor límite.

¿Cuál es el conjunto MÍNIMO de casos de prueba que logra una cobertura BVA del 100% de 2 valores? Los valores en las respuestas indican el número del lavado dado (actual).

Tabla 4.15 Tabla de decisiones para reglas de asignación de tarifa gratuita

	R1	R2	R3
Condiciones			
¿Miembro del Parlamento?	Sí	-	-
¿Desactivado?	-	Sí	-
¿Alumno?	-	-	Sí
Acción			
¿Paseo libre?	Sí	Sí	NO

- R. 9, 10
 B.1, 9, 10
 C.1, 9, 10, 11
 D. No se puede lograr la cobertura requerida

Elija una respuesta.

Pregunta 4.8
 (FL-4.2.3, K3)

El analista de negocios preparó una tabla de decisión minimizada (Tabla 4.15) para describir las reglas comerciales para conceder viajes gratuitos en autobús. Sin embargo, la tabla de decisiones es defectuosa. ¿Cuál de los siguientes casos de prueba, que representan combinaciones de condiciones, muestra que las reglas de negocio en esta tabla de decisiones son CONTRADICTORIAS?

- A. Miembro del parlamento = Sí, discapacitado = NO, estudiante = Sí.
 B. Miembro del parlamento = Sí, discapacitado = Sí, estudiante = NO.
 C. Miembro del parlamento = NO, discapacitado = NO, estudiante = Sí.
 D. Miembro del parlamento = NO, discapacitado = NO, estudiante = NO.

Elija una respuesta.

Pregunta 4.9
 (FL-4.2.3, K3)

Crea una tabla de decisiones completa que tiene las siguientes condiciones y acciones:

- Condición "edad"—valores posibles: (1) hasta 18; (2) 19–40; (3) 41 o más.
- Condición "lugar de residencia"—valores posibles: (1) ciudad; (2) pueblo.
- Condición "salario mensual"—valores posibles: (1) hasta \$4000; (2) \$4001 o más.
- Acción "otorgar crédito"—valores posibles: (1) Sí; (2) NO.
- Acción "ofrecer seguro de crédito"—valores posibles: (1) Sí; (2) NO.

¿Cuántas columnas tendrá la tabla de decisión completa para este problema?

- R.6
 B.11
 C.12
 D.48

Elija una respuesta.

Tabla 4.16 Transiciones válidas
del sistema para el inicio de sesión
proceso

Transición	Estado	Evento	siguiente estado
1	Inicial	Acceso	Inicio sesión
2	Inicio sesión	Iniciar sesiónOK	registrado
3	Inicio sesión	Error de inicio de sesión	Inicial
4	registrado	Cerrar sesión	Inicial

Pregunta 4.10

(FL-4.2.4, K3)

La Tabla 4.16 muestra en sus filas TODAS las transiciones válidas entre estados en el sistema para manejar el proceso de inicio de sesión. El sistema contiene tres estados (Inicial, Registro, Registrado) y cuatro posibles eventos (Iniciar sesión, Iniciar sesión OK, Iniciar sesiónError, Cerrar sesión).

¿Cuántas transiciones INVÁLIDAS hay en este sistema?

- R.8
- b.0
- C.4
- D.12

Elija una respuesta.

Pregunta 4.11

(FL-4.2.4, K3)

La figura 4.17 muestra un diagrama de transición de estado para un determinado sistema.

¿Cuál es el número MÍNIMO de casos de prueba que alcanzarán el 100% de validez?
cobertura de transiciones?

- R.2
- B.3
- C.6
- D.7

Elija una respuesta.

Pregunta 4.12

(FL-4.3.1, K2)

Después de la ejecución de la prueba, sus casos de prueba lograron una cobertura de declaración del 100%. cual de
¿Las siguientes afirmaciones describen la consecuencia correcta de este hecho?

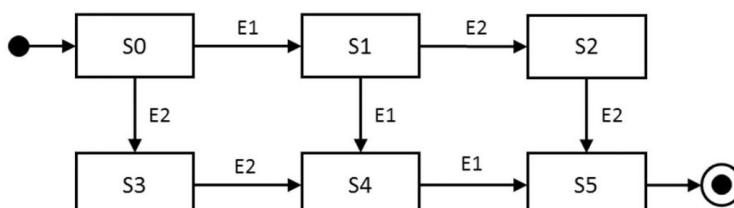


Fig. 4.17 Diagrama de transición de una máquina de estados

- R. Se logra una cobertura del 100% en sucursales.
- B. Se ejerció cada valor lógico de cada decisión en el código.
- C. Se aplicaron todos los resultados posibles que el programa bajo prueba puede devolver.
- D. Se ejerció cada declaración que contenía un defecto.

Elija una respuesta.

Pregunta 4.13

(FL-4.3.2, K2)

Considera un programa simple de tres declaraciones (suponga que las declaraciones 1 y 2 toman números enteros no negativos (0, 1, 2, etc.) y que estas declaraciones siempre se ejecutarán correctamente):

- 1. Obtener el valor de entrada x * 2.

Obtener el valor de entrada y 3.

Devolver x + y

¿Cuántos casos de prueba se necesitan para lograr una cobertura de sucursal del 100% en este código y por qué?

- R. Se necesitan dos casos de prueba, porque uno debe probar la situación en la que el valor devuelto es 0 y el otro debe probar la situación en la que el valor devuelto es un número positivo.
- B. No es necesario ejecutar ninguna prueba, porque la cobertura de bifurcaciones para este código se cumple por definición, ya que el programa consta de un pasaje secuencial de tres declaraciones y no hay bifurcaciones condicionales para probar.
- C. Se necesita un caso de prueba con valores de entrada arbitrarios x e y, porque cada prueba dará como resultado la ejecución de la misma ruta que cubre todas las ramas.
- D. Es imposible lograr cobertura de rama con un número finito de casos de prueba, porque para hacerlo, tendríamos que forzar todos los valores posibles de la suma de x+y en el enunciado 3, lo cual no es factible.

Elija una respuesta.

Pregunta 4.14

FL-4.3.3 (K2)

¿Cuál de las siguientes opciones describe MEJOR el beneficio de utilizar técnicas de prueba de caja blanca?

- A. La capacidad de detectar defectos cuando las especificaciones están incompletas.
- B. La capacidad de los desarrolladores para realizar pruebas, ya que estas técnicas requieren programación habilidades de minería.
- C. Asegurarse de que las pruebas alcancen una cobertura del 100% de cualquier técnica de caja negra, ya que esto es implica lograr una cobertura de código del 100%.
- D. Mejor control del nivel de riesgo residual en el código, ya que está directamente relacionado con medidas como como cobertura de estados de cuenta y cobertura de sucursales.

Elija una respuesta.

Pregunta 4.15

(FL-4.4.1, K2)

El evaluador utiliza el siguiente documento para diseñar los casos de prueba:

1. Ocurrencia de un error aritmético provocado al dividir por cero.
2. Ocurrencia de error de redondeo.
3. Forzar un resultado de valor negativo.

¿Qué técnica utiliza el probador?

- A. Análisis de valores en la frontera.
- B. Pruebas basadas en listas de verificación.
- C. Pruebas basadas en casos de uso.
- D. Error al adivinar

Elija una respuesta.

Pregunta 4.16

(FL-4.4.2, K2)

¿Cuál de las siguientes es la oración correcta con respecto al uso de técnicas de prueba formales (es decir, de caja negra y de caja blanca) como parte de una prueba exploratoria basada en sesiones?

- R. Se permiten todas las técnicas de prueba formales, porque las pruebas exploratorias no imponen ningún método de operación específico.
- B. Todas las técnicas de prueba formales están prohibidas, porque las pruebas exploratorias se basan en el conocimiento, las habilidades, la intuición y la experiencia del evaluador.
- C. Todas las técnicas de prueba formales están prohibidas, porque los pasos realizados en la exploración Las pruebas teóricas no están planificadas con antelación.
- D. Se permiten todas las técnicas de prueba formales, porque un evaluador exploratorio necesita una base de prueba de la cual derivar casos de prueba.

Elija una respuesta.

Pregunta 4.17

(FL-4.4.3, K2)

¿Qué beneficio se puede lograr mediante el uso de pruebas basadas en listas de verificación?

- A. Valoración de las pruebas no funcionales, que a menudo se subestiman. B. Permitir una medición precisa de la cobertura del código.
- C. Aprovechar la experiencia del evaluador.
- D. Consistencia de prueba mejorada.

Elija una respuesta.

Pregunta 4.18

(FL-4.5.1, K2)

Durante una reunión de planificación de iteraciones, el equipo comparte ideas sobre la historia del usuario. El propietario del producto quiere que el cliente tenga un formulario de pantalla única para ingresar información. El desarrollador explica que esta característica tiene algunas

limitaciones técnicas, relacionadas con la cantidad de información que se puede capturar en el pantalla.

¿Cuál de las siguientes opciones representa MEJOR la continuación de la escritura de esta historia de usuario?

- R. El evaluador decide que el formulario debe caber en la pantalla y describe esto como uno de los criterios de aceptación de la historia, ya que será él quien realizará las pruebas de aceptación más adelante.
- B. El evaluador escucha los puntos de vista del propietario del producto y del desarrollador y crea dos pruebas de aceptación para cada una de las dos soluciones propuestas.
- C. El evaluador informa al desarrollador que los criterios de aceptación del desempeño deben basarse en un estándar: un máximo de 1 segundo por registro de datos. El desarrollador describe esto como uno de los criterios de aceptación, ya que el desarrollador es responsable del rendimiento de la aplicación.
- D. El evaluador negocia con el desarrollador y el propietario del producto la cantidad de información de entrada necesaria y juntos deciden reducir esta información a la más importante para que quepa en la pantalla.

Elija una respuesta.

Pregunta 4.19

(FL-4.5.2, K2)

Considere la siguiente historia de usuario:

Como cliente potencial de la tienda electrónica,
quiero poder registrarme completando el formulario de registro, para poder utilizar
todas las funciones de la tienda electrónica.

¿Cuáles DOS de los siguientes son ejemplos de criterios de aceptación comprobables para esta historia?

- R. El proceso de registro debe realizarse con la suficiente rapidez.
- B. Después del registro, el usuario tiene acceso a la función “pedidos a domicilio”.
- C. El sistema rechaza el registro si el usuario ingresa como inicio de sesión un correo electrónico ya existente en la base de datos.
- D. El operador del sistema puede enviar el pedido realizado por el usuario para su procesamiento.
- E. Los criterios de aceptación para esta historia de usuario deben estar en formato Dado/Cuándo/Entonces.

Seleccione DOS respuestas.

Pregunta 4.20

(FL-4.5.3, K3)

La regla comercial para el préstamo de libros en el sistema de bibliotecas universitarias establece que un lector puede pedir prestados libros nuevos si se cumplen las dos condiciones siguientes:

- El tiempo de conservación del libro más antiguo no supera los 30 días. • Despues del préstamo, el número total de libros prestados no excederá los cinco libros.
para un estudiante y diez libros para un profesor.

El equipo debe implementar una historia de usuario para este requisito. La historia se crea. utilizando el marco ATDD, y los siguientes criterios de aceptación están escritos como Ejemplos en el formato Dado/Cuándo/Entonces:

Dado que <UserType> ya ha tomado prestados <Number> de libros
 Y el tiempo del libro retenido por más tiempo es <Días> días.
 Cuando el usuario quiere pedir prestado <Solicitar> libros nuevos
 Luego el sistema <Decisión> de prestar los libros.

Ejemplos:

No | Tipo de usuario | Número | Días | Solicitar | Decisión

1	Estudiante	3	30	2	no permite
2	Estudiante	4	1	1	permite
3	Profesor	6	32	3	no permite
4	Profesor	0	0	6	permite

¿Cuántos de los cuatro casos de prueba anteriores están definidos INCORRECTAMENTE, es decir, ¿violar las reglas comerciales del préstamo de libros?

R. Ninguno: todos siguen la regla empresarial.

Hueso.

C. Dos.

D. Tres.

Ejercicios

Ejercicio 4.1

(FL-4.2.1, K3)

El usuario rellena un formulario web para adquirir entradas para conciertos. Los boletos están disponibles para tres conciertos: Iron Maiden, Judas Priest y Black Sabbath. Para cada uno de los conciertos se pueden adquirir dos tipos de entradas: un sector frente al escenario y un sector alejado del escenario. El usuario confirma la elección marcando el cuadros en las listas desplegables correspondientes (una de las cuales contiene los nombres de los bandas, el otro, el tipo de entrada). Sólo se puede adquirir una entrada para una sola banda. comprado por sesión.

Quiere comprobar la corrección del sistema para cada banda y, independientemente deadamente, para el tipo de billete.

A) Identificar los dominios y sus particiones de equivalencia.

B) ¿Hay particiones no válidas? Justifica tu respuesta.

C) Diseñar el conjunto más pequeño posible de casos de prueba que alcancen el 100% de equivalencia. Cobertura de partición.

Ejercicio 4.2

(FL-4.2.1, K3)

Tabla 4.17 Reglas de descuento

Cantidad	Descuento concedido
Hasta \$300	No
Más de \$300, hasta \$800	5%
Más de \$800	10%

Un número natural mayor que 1 se llama primo si es divisible sólo por dos números: por 1 y por sí mismo. El sistema toma un número natural (ingresado por el usuario en el campo del formulario) como entrada y devuelve si es primo o no. El campo del formulario para los datos de entrada tiene un mecanismo de validación y no permitirá ingresar ninguna cadena que no represente una entrada válida (es decir, un número natural mayor que uno).

- A) Identificar el dominio y su partición de equivalencia.
- B) ¿Hay particiones no válidas en el problema? Justifica tu respuesta.
- C) Diseñe el conjunto más pequeño posible de casos de prueba que logren una cobertura de partición de equivalencia del 100%.

Ejercicio 4.3

(FL-4.2.2, K3)

Estás probando la funcionalidad de pago de la tienda electrónica. El sistema recibe una cantidad positiva de compras (en \$ con una precisión de 1 centavo). Luego, este monto se redondea al número entero más cercano y, en función de este valor redondeado, se calcula un descuento de acuerdo con las reglas descritas en la Tabla 4.17: Desea aplicar un BVA de 2 valores para

verificar la exactitud del cálculo del descuento. Los datos de entrada en el caso de prueba son la cantidad antes del redondeo. Realice particiones de equivalencia, determine los valores límite y diseñe los casos de prueba.

Ejercicio 4.4

(FL-4.2.2, K3)

El sistema calcula el precio del marco del cuadro basándose en los parámetros dados: ancho y alto del cuadro (en centímetros). El ancho correcto del cuadro está entre 30 y 100 cm inclusive. La altura correcta del cuadro está entre 30 y 60 cm inclusive.

El sistema toma ambos valores de entrada a través de una interfaz que acepta sólo valores válidos. Valores de los rangos especificados anteriormente.

El sistema calcula el área de la imagen como el producto del ancho y el alto. Si el precio del marco es La superficie supera los 1600 cm² El precio, \$500. De lo contrario, el encuadre es de \$450.

Aplique BVA de 2 valores al problema anterior:

- A) Identificar las particiones y sus valores límite.
- B) Diseñe el menor número posible de casos de prueba que cubran los valores límite para todos los parámetros relevantes. ¿Es posible lograr una cobertura total de BVA de 2 valores? Justifica tu respuesta.

Ejercicio 4.5

(FL-4.2.3, K3)

El operador del sistema de soporte para el examen de licencia de conducir ingresa la siguiente información en el sistema para un candidato que realiza los exámenes por primera vez:

- El número de puntos del examen teórico (número entero de 0 a 100). • El número de errores cometidos por el candidato durante el examen práctico (número entero número 0 o mayor).

El candidato deberá realizar ambos exámenes. A un candidato se le concede una licencia de conducir si cumple las dos condiciones siguientes: obtuvo al menos 85 puntos en el examen teórico y no cometió más de dos errores en el examen práctico. Si un candidato no aprueba uno de los exámenes, deberá repetirlo. Además, si el candidato no aprueba ambos exámenes, deberá realizar horas adicionales de lecciones de conducción.

Utilice pruebas de tablas de decisión para llevar a cabo el proceso de diseño de casos de prueba para el problema anterior. Siga el procedimiento de cinco pasos descrito en la Sección. 4.2.3, es decir:

- A) Identifique todas las condiciones posibles y enumérelas en la parte superior de la tabla.
- B) Identifique todas las posibles acciones que pueden ocurrir en el sistema y enumérelas en la parte inferior de la tabla.
- C) Generar todas las combinaciones de condiciones. Elimine las combinaciones no factibles (si las hay) y enumere todas las combinaciones factibles restantes en columnas individuales en la parte superior de la matriz.
- D) Para cada combinación de condiciones así identificadas, determine, con base en la especificación, qué acciones deben realizarse en el sistema e intodúzcalas en las columnas correspondientes en la parte inferior de la tabla.
- E) Para cada columna, diseñe un caso de prueba que incluya un nombre (que describa lo que prueba el caso de prueba), condiciones previas, datos de entrada, resultados esperados y condiciones posteriores.

Ejercicio 4.6

(FL-4.2.3, K3)

La Figura 4.18 (según Graham et al., Foundations of Software Testing) muestra el proceso de asignación de asientos a los pasajeros en función de si tienen una tarjeta dorada y si hay asientos disponibles en las clases ejecutiva y económica, respectivamente.

Describe este proceso con una tabla de decisiones. ¿Notaste algún problema con el especificación al crear la tabla? Si es así, sugiera una solución al problema.

Ejercicio 4.7

(FL-4.2.4, K3)

El cajero automático se encuentra inicialmente en estado de espera (pantalla de bienvenida). Después de que el usuario inserta la tarjeta, se lleva a cabo la validación de la tarjeta. Si la tarjeta no es válida, el sistema la devuelve y finaliza con el mensaje "Error de tarjeta". De lo contrario, el sistema solicita al usuario que ingrese un PIN. Si el usuario proporciona un PIN válido, el sistema cambia al estado "Registrado" y finaliza la operación. Si el usuario ingresa un PIN incorrecto, el sistema solicita ingresar nuevamente. Si el usuario ingresa el PIN incorrecto tres veces, la tarjeta se bloquea, el usuario recibe el mensaje "Tarjeta bloqueada" y el sistema pasa al estado final que representa la tarjeta bloqueada.

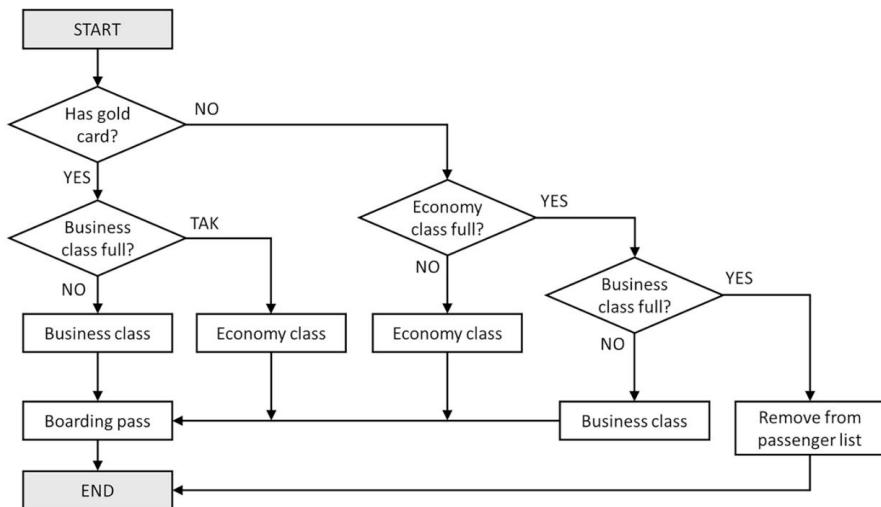


Fig. 4.18 El proceso de asignación de asientos en un avión.

- A) Identificar posibles estados, eventos y acciones, y diseñar un diagrama de transición de estados.
para el escenario anterior sin utilizar condiciones de guardia.
- B) Diseñar un diagrama de transición de estado para el mismo escenario pero usando condiciones de guardia. Debería obtener un modelo con menos estados.
- C) Para el diagrama de transición de estado de (A), diseñe el menor número posible de casos de prueba que logren:
- Cobertura del 100% en todos los estados.
 - Cobertura de transiciones 100% válida.

Ejercicio 4.8

(FL-4.2.4, K3)

El funcionamiento del perro robot mecánico se describe mediante la transición de estado. diagrama que se muestra en la Fig. 4.19. El estado inicial es "S".

- A) ¿Cuántas transiciones no válidas hay en este diagrama?
- B) Diseñar casos de prueba que logren una cobertura total de todas las transiciones. Adopte la regla de que al probar transiciones no válidas, un caso de prueba prueba solo una transición no válida.

Ejercicio 4.9

(FL-4.5.3, K3)

Como tester, comienzas a trabajar en el diseño de la prueba de aceptación para la siguiente historia de usuario:

US-01-002 Registro de nuevo usuario Como:

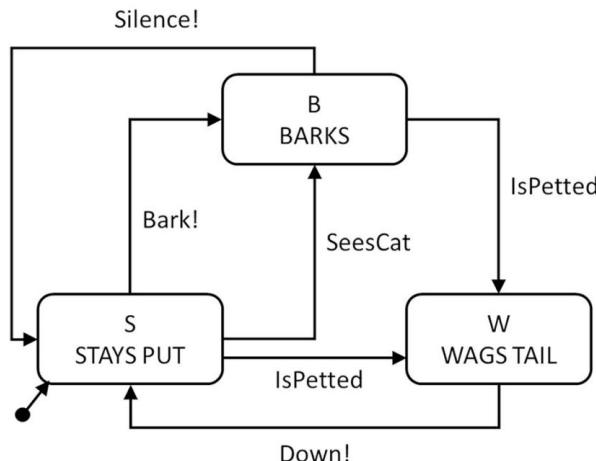
Estimación del esfuerzo: 3

cualquier usuario—cliente potencial Quiero:

poder completar el formulario de registro web Para que: Soy poder

utilizar las ofertas del proveedor de servicios Criterios de aceptación:

Fig. 4.19 Diagrama de transición
de estado del perro robot



AC1 El inicio de sesión del usuario es una dirección de correo electrónico válida.

El usuario AC2 no puede registrarse utilizando un inicio de sesión existente

AC3 Por motivos de seguridad, el usuario deberá introducir la contraseña en dos campos independientes, y dichas contraseñas deben ser idénticas; el sistema no permite al usuario pegar una cadena en estos campos; debe ingresarse manualmente

La contraseña AC4 debe contener al menos 6 y como máximo 12 caracteres, al menos 1 número

y al menos 1 letra mayúscula

AC5 El registro exitoso da como resultado el envío de un correo electrónico que contiene un enlace, el Al hacer clic se confirma el registro y se activa la cuenta del usuario.

Proponer algunas pruebas de aceptación de muestra con datos de prueba específicos y el comportamiento esperado del sistema.

Capítulo 5 Gestión de las actividades de prueba



Palabras clave

Gestión de defectos	El proceso de reconocer, registrar, clasificar, investigar, resolver y eliminar defectos
Informe de defectos	Documentación de la ocurrencia, naturaleza y estado de un defecto. Sinónimos: informe de error
Criterio para entrar	El conjunto de condiciones para iniciar oficialmente una tarea definida. Referencias: Gilb y Graham
Criterio de salida	El conjunto de condiciones para completar oficialmente un determinado tarea. Después de Gilb y Graham. Sinónimos: finalización de la prueba criterios, criterios de finalización
Riesgo del producto	Un riesgo que afecta la calidad de un producto.
Riesgo del proyecto	Un riesgo que impacta el éxito del proyecto
Riesgo	Un factor que podría resultar en futuros negativos. consecuencias
Análisis de riesgo	El proceso general de identificación de riesgos y riesgos. evaluación
Evaluación de riesgos	El proceso para examinar los riesgos identificados y determinar el nivel de riesgo
Control de riesgo	El proceso general de mitigación de riesgos y riesgos. supervisión
Identificación de riesgo	El proceso de encontrar, reconocer y describir. riesgos. Referencias: ISO 31000
Nivel de riesgo	La medida de un riesgo definida por el impacto y la probabilidad. Sinónimos: exposición al riesgo
Gestión de riesgos	El proceso de gestión de riesgos. Después de ISO 24765
Mitigación de riesgos	El proceso mediante el cual se toman decisiones y Se implementan medidas de protección para reducir o mantener los riesgos a niveles específicos

Monitoreo de riesgos	La actividad que verifica e informa el estado de los conocidos riesgos para las partes interesadas
Pruebas basadas en riesgos	Pruebas en las que la gestión, selección, priorización y uso de actividades y recursos de prueba se basan en los tipos y niveles de riesgo correspondientes.
	Después de ISO 29119-1
Enfoque de prueba	La forma de implementar las tareas de prueba.
Informe de finalización de la prueba	Un tipo de informe de prueba elaborado en los hitos de finalización que proporciona una evaluación de la prueba correspondiente elementos según los criterios de salida. Sinónimos: resumen de la prueba informe
Control de prueba	La actividad que desarrolla y aplica acciones correctivas para encaminar un proyecto de prueba cuando se desvía de lo que fue planeado
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
Plan de prueba	Documentación que describa los objetivos de la prueba que se deben conseguir y los medios y el calendario para lograrlo. ellos, organizados para coordinar las actividades de prueba.
	Referencias: ISO 29119-1
Planificación de pruebas	La actividad de establecer o actualizar un plan de pruebas.
Informe de progreso de la prueba	Un tipo de informe de prueba periódico que incluye el progreso de las actividades de prueba frente a una línea de base, riesgos y alternativas requiriendo una decisión. Sinónimos: informe de estado de prueba
Pirámide de prueba	Un modelo gráfico que representa la relación de la cantidad de pruebas por nivel, con más en la parte inferior que en la cima
Cuadrantes de prueba	Un modelo de clasificación de tipos de pruebas/niveles de pruebas en cuatro cuadrantes, relacionándolos con dos dimensiones de la prueba. objetivos: apoyar al equipo de producto versus criticar el producto y la tecnología frente a frente a los negocios

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 iteración y la planificación de lanzamientos
- FL-5.1.3 (K2) Comparar y contrastar criterios de entrada y criterios de salida
- FL-5.1.4 (K3) Usar técnicas de estimación para calcular el esfuerzo de prueba requerido

FL-5.1.5 (K3) Aplicar la 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

5.1.1 Propósito y contenido de un plan de prueba

El plan de prueba es un documento que proporciona una descripción detallada de los objetivos del proyecto de prueba, los medios necesarios para lograr esos objetivos y un cronograma de actividades de prueba. En proyectos típicos, generalmente se crea un único plan de prueba, a veces llamado plan maestro de prueba o plan de prueba del proyecto. Sin embargo, en proyectos más grandes, puede encontrar varios planes, como un plan maestro de pruebas y planes correspondientes a los niveles de prueba definidos en el proyecto (plan de pruebas de nivel o plan de pruebas de fases). En tal situación, puede haber, por ejemplo, un plan de pruebas de integración de componentes, un plan de pruebas del sistema, un plan de pruebas de aceptación, etc. Los planes de pruebas detallados también pueden estar relacionados con los tipos de pruebas que se planean llevar a cabo en el proyecto (por ejemplo, un plan de pruebas de rendimiento).

El plan de prueba describe el enfoque de la prueba y ayuda al equipo a asegurarse de que se puedan iniciar las actividades de prueba y, cuando se completen, que se hayan realizado correctamente. Esto se logra definiendo criterios específicos de entrada y salida (ver Sección 5.1.3) para cada actividad de prueba o actividad en el plan de prueba. El plan de pruebas también confirma que, si se sigue, las pruebas se realizarán de acuerdo con la estrategia de pruebas del proyecto y la política de pruebas de la organización.

Muy raramente el proyecto saldrá 100% como lo planeamos. En la mayoría de las situaciones, serán necesarias modificaciones menores o mayores. Entonces surge la pregunta natural: ¿por qué perder el tiempo planificando en un caso así? Dwight Eisenhower, general del ejército estadounidense y más tarde presidente de los Estados Unidos, dijo una vez: "Al prepararme para la batalla, siempre me he dado cuenta de que los planes son inútiles, pero la planificación es indispensable".

¹ De hecho,

la parte más valiosa de la creación de un plan de pruebas es el proceso de planificación de pruebas.

En cierto sentido, la planificación "obliga" a los evaluadores a centrar su pensamiento en los desafíos y riesgos futuros relacionados con el cronograma, los recursos, las personas, las herramientas, el costo, el esfuerzo, etc.

Durante la planificación, los evaluadores pueden identificar riesgos y pensar en el enfoque de prueba que será más efectivo. Sin planificación, los evaluadores no estarían preparados para los diferentes eventos indeseables que sucederán durante el proyecto. Esto, a su vez, crearía un riesgo muy alto de fracaso del proyecto, es decir, no completarlo o hacerlo tarde, por encima del presupuesto o con un alcance reducido del trabajo completado.

¹ https://pl.wikiquote.org/wiki/Dwight_Eisenhower

El proceso de planificación está influenciado por muchos factores que los evaluadores deben considerar para planificar mejor todas las actividades del proceso de prueba. Entre estos factores, se deben considerar en particular los siguientes:

- Política de pruebas y estrategia de pruebas •

SDLC • Alcance de las

pruebas • Objetivos

• Riesgos •

Limitaciones •

Criticidad •

Capacidad de

prueba • Disponibilidad de recursos

Un plan de prueba típico incluye la siguiente información: Contexto de la prueba. Alcance, objetivos, limitaciones e información de la base de la prueba Supuestos y limitaciones del proyecto de prueba Partes interesadas. Roles, responsabilidades, influencia en el proceso de prueba (por ejemplo, poder versus interés) y necesidades de contratación y capacitación de personas. Comunicación. Tipos y frecuencia de comunicación y plantillas de documentos utilizados.

Lista de riesgos. Riesgos del producto y riesgos del proyecto (ver Sección 5.2)

Enfoques para las pruebas. Niveles de prueba (ver Sección 2.2.1), tipos de prueba (ver Sección 2.2.2), técnicas de prueba (ver Capítulo 4), productos del trabajo de prueba, criterios de entrada y salida (ver Sección 5.1.3), nivel de independencia de la prueba (ver Sección 1.5.3), definición de las métricas de prueba utilizadas (ver Sección 5.3.1), requisitos de datos de prueba, requisitos del entorno de prueba y desviaciones de las mejores prácticas organizacionales (con justificación)

Horario (ver Apartado 5.1.5)

A medida que se realiza el proyecto y se implementa el plan de prueba, aparece información adicional que detalla el plan. Por tanto, la planificación de pruebas es una actividad continua y se realiza durante todo el ciclo de vida del producto; a veces incluye una fase de mantenimiento. Es importante darse cuenta de que el plan de prueba inicial se actualizará, ya que la retroalimentación de las actividades de prueba se utilizará para identificar riesgos cambiantes y ajustar los planes en consecuencia.

Los resultados del proceso de planificación se pueden documentar en un plan maestro de pruebas y en Planes separados para niveles de prueba y tipos de prueba específicos.

Los siguientes pasos se realizan durante la planificación de la prueba (ver Fig. 5.1):

- Definir el alcance y los objetivos de las pruebas y los riesgos asociados • Determinar el enfoque general de las pruebas • Integrar y coordinar las actividades de prueba dentro de las actividades del SDLC • Decidir qué probar, qué personal y otros recursos se necesitarán para realizar las diversas actividades de prueba y cómo se deben realizar las actividades

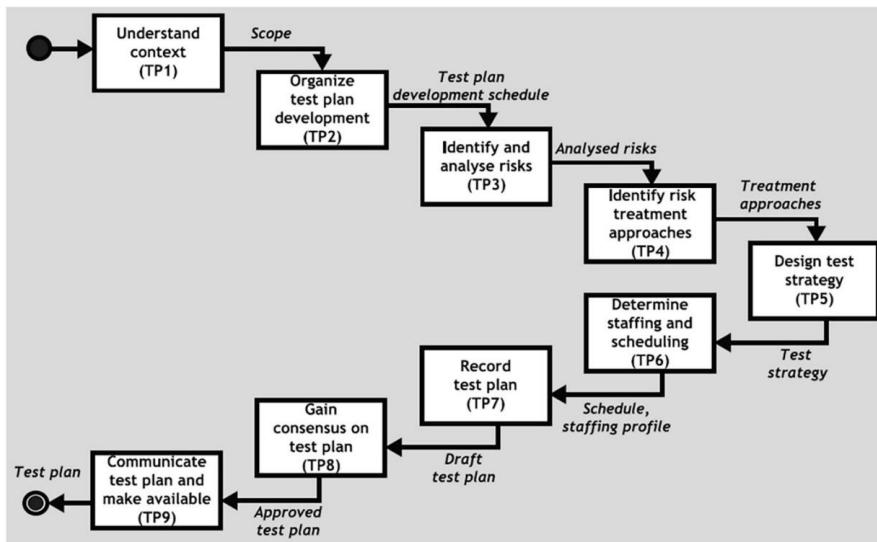


Fig. 5.1 Proceso de planificación de pruebas (según norma ISO/IEC/IEEE 29119-2)

- Planificar las actividades de análisis, diseño, implementación, ejecución y evaluación de pruebas estableciendo plazos específicos (enfoque secuencial) y colocando estas actividades en el contexto de iteraciones individuales (enfoque iterativo) • Hacer una selección de medidas para el seguimiento y control de las pruebas • Determinar el presupuesto para la recopilación y evaluación de métricas. • Determinar el nivel de detalle y estructura de la documentación (plantillas o documentos de muestra).

Una parte del plan de prueba (el enfoque de prueba) será en la práctica la implementación de una estrategia de prueba específica vigente en la organización o proyecto. La estrategia de prueba es una descripción general del proceso de prueba que se está implementando, generalmente a nivel de producto u organización.

Estrategias de prueba

Las estrategias de prueba típicas son:

- Estrategia analítica. Esta estrategia se basa en el análisis de un factor específico (por ejemplo, requisitos o riesgo). Un ejemplo de enfoque analítico son las pruebas basadas en riesgos, en las que el punto de partida para el diseño y la priorización de las pruebas es el nivel de riesgo.
- Estrategia basada en modelos. Con esta estrategia, las pruebas se diseñan sobre la base de un modelo de un aspecto específico requerido del producto, por ejemplo, función, proceso de negocio, estructura interna o características no funcionales (como

(continuado)

como confiabilidad). Se pueden crear modelos basados en modelos de procesos de negocio, modelos de estado, modelos de crecimiento de confiabilidad, etc.

- Estrategia metódica. La base de esta estrategia es la aplicación sistemática de un conjunto predeterminado de pruebas o condiciones de prueba, por ejemplo, un conjunto estándar de pruebas o una lista de verificación que contiene tipos de fallas típicas o probables. Según este enfoque, se realizan, entre otras cosas, pruebas basadas en listas de verificación, ataques de fallas y pruebas basadas en características de calidad.
- Estrategia compatible con el proceso (o compatible con el estándar). Esta estrategia implica la creación de casos de prueba basados en reglas y estándares externos (derivados, por ejemplo, de estándares de la industria), documentación de procesos o identificación y uso riguroso de la base de prueba, así como cualquier proceso o estándar impuesto por la organización. • Estrategia dirigida (o consultiva). Esta estrategia se basa principalmente en el asesoramiento y la orientación de las partes interesadas, expertos en la materia o expertos técnicos, incluidos aquellos ajenos al equipo de prueba o a la organización. • Estrategia adversa a la regresión. Esta estrategia, motivada por el deseo de evitar la regresión de funcionalidades ya existentes, prevé la reutilización del software de prueba heredado (especialmente casos de prueba), la automatización extensiva de las pruebas de regresión y el uso de conjuntos de pruebas estándar. • Estrategia reactiva. Con esta estrategia, las pruebas están más orientadas a reaccionar ante eventos que a seguir un plan predeterminado (como ocurre con las estrategias descritas anteriormente), y las pruebas se diseñan y pueden ejecutarse inmediatamente en función del conocimiento adquirido a partir de los resultados de pruebas anteriores.

Un ejemplo de este enfoque son las pruebas exploratorias, en las que las pruebas se ejecutan y evalúan en respuesta al comportamiento del software bajo prueba.

En la práctica, se pueden e incluso se deben combinar diferentes estrategias. Los conceptos básicos de la selección de estrategias de prueba incluyen:

- Riesgo de fracaso del proyecto:
 - Riesgos para el producto
 - Peligro para las personas, el medio ambiente o la empresa causado por fallos del producto
 - Falta de habilidades y experiencia de las personas en el proyecto.
- Regulaciones (externas e internas) sobre el proceso de desarrollo • Propósito de la empresa de prueba • Misión del equipo de prueba • Tipo y características específicas del producto

5.1.2 Contribución del evaluador a la planificación de iteraciones y lanzamientos

En los enfoques iterativos para el desarrollo de software, existen dos tipos de planificación relacionada con los productos: planificación de lanzamiento y planificación de iteraciones. Existen otros tipos de planificación, a nivel superior, organizacional o estratégico (como la planificación de "gran sala" o la planificación de "incremento de producto"). Estos tipos de planificación no se analizan más.

Planificación de lanzamientos

La planificación de lanzamientos anticipa el lanzamiento de un producto. La planificación del lanzamiento define y redefine el trabajo pendiente del producto y puede incluir refinar historias de usuarios más grandes en una colección de historias más pequeñas. La planificación de lanzamiento proporciona la base para un enfoque de prueba y un plan de prueba que cubra todas las iteraciones del proyecto. Durante la planificación del lanzamiento, los representantes comerciales (en colaboración con el equipo) determinan y priorizan las historias de usuarios para un lanzamiento. Sobre la base de estas historias de usuarios, se identifican los riesgos del proyecto y del producto (ver Sección 5.2) y se realiza una estimación del esfuerzo de alto nivel (ver Sección 5.1.4).

Los evaluadores participan en la planificación del lanzamiento y agregan valor, especialmente en las siguientes actividades:

- Definir historias de usuario comprobables, incluidos los criterios de aceptación.
- Participar en el análisis de riesgos (de calidad) del proyecto y del producto.
- Estimar el esfuerzo de prueba relacionado con las historias de usuario.
- Definir los niveles de prueba necesarios.
- Planificar las pruebas para el lanzamiento.

Planificación de iteraciones

Una vez que se completa la planificación del lanzamiento, comienza la planificación de iteraciones para la primera iteración. La planificación de iteraciones mira hacia el final de una sola iteración y aborda el trabajo pendiente de iteración. Durante la planificación de la iteración, el equipo selecciona historias de usuarios de la cartera de productos priorizada, las refina (aclara) y las divide (cuando sea necesario), las desarrolla, realiza análisis de riesgos para las historias de usuarios y estima el trabajo necesario para implementar cada historia seleccionada (ver sección 5.1.4). Si una historia de usuario no está clara y los intentos de aclararla han fallado, el equipo puede rechazarla y utilizar la siguiente historia de usuario según la prioridad. Los representantes comerciales deben responder las preguntas del equipo sobre cada historia para que el equipo comprenda qué debe implementar y cómo probar cada historia.

El número de historias seleccionadas se basa en la llamada velocidad del equipo² y el tamaño estimado de las historias de usuario seleccionadas, así como en limitaciones técnicas. Una vez el

² La velocidad del equipo es la cantidad de trabajo determinada empíricamente que un equipo es capaz de realizar durante una sola iteración. Generalmente se expresa en términos de los llamados puntos de historia de usuario. El tamaño de cada historia también se estima en estas unidades, de modo que el equipo sepa cuántas historias de usuario puede incluir en el trabajo pendiente de iteración; la suma de su complejidad no puede exceder la velocidad del equipo. Esto reduce el riesgo de que el equipo no tenga tiempo para completar todo el trabajo a realizar en una iteración y que el equipo no termine el trabajo antes del final de la iteración, lo que provoca las llamadas ejecuciones vacías.

Una vez finalizado el contenido de la iteración, las historias de usuario se dividen en tareas que deben ejecutar los miembros del equipo correspondientes.

Los probadores participan en la planificación de iteraciones y agregan valor, especialmente en las siguientes actividades:

- Participar en análisis de riesgo detallados de historias de usuarios •

Determinar la capacidad de prueba de las historias de usuarios

- Co-crear pruebas de aceptación para historias de usuarios •

Dividir historias de usuarios en tareas (especialmente tareas de prueba) •

Estimar el esfuerzo de prueba para todas las tareas de prueba •

Identificar funciones funcionales y no -Aspectos de prueba funcionales del sistema bajo prueba. • Apoyar y participar en la automatización de pruebas en múltiples niveles de prueba.

5.1.3 Criterios de entrada y criterios de salida

Criterios de entrada

Los criterios de entrada (más o menos similares a la Definición de Listo en un enfoque ágil) definen las condiciones previas que deben cumplirse antes de que pueda comenzar una actividad de prueba. Si no se cumplen, las pruebas pueden ser más difíciles y requerir más tiempo, costosas y riesgosas.

Los criterios de entrada incluyen la disponibilidad de recursos o software de prueba:

- Requisitos, historias de usuario y/o modelos comprobables • Elementos

de prueba que cumplieron con los criterios de salida aplicables a niveles anteriores de prueba, principalmente en el enfoque en cascada • Entorno

de prueba • Herramientas

de prueba necesarias • Datos

de prueba y otros recursos necesarios

así como el nivel de calidad inicial de un objeto de prueba (por ejemplo, todas las pruebas de humo pasan). Los criterios de entrada nos protegen de iniciar tareas para las que aún no estamos completamente preparados.

Criterios de salida

Los criterios de salida (más o menos similares a la Definición de Hecho en un enfoque ágil) definen las condiciones que deben cumplirse para que la ejecución de un nivel de prueba o conjunto de pruebas se considere completa.

Estos criterios deben definirse para cada nivel de prueba y tipo de prueba. Pueden variar dependiendo de los objetivos de la prueba.

Los criterios de salida típicos son:

- Finalización de la ejecución de las pruebas programadas. • Lograr el nivel adecuado de cobertura (por ejemplo, requisitos, historias de usuarios, aceptación).
- criterios, código) •

No exceder el límite acordado de defectos no reparados

- Obtener una densidad de defectos estimada suficientemente baja.
- Lograr índices de confiabilidad suficientemente altos.

Tenga en cuenta que a veces las actividades del examen pueden acortarse debido a:

- El uso de todo el presupuesto
- El paso del tiempo programado
- La presión de sacar un producto al mercado

En tales situaciones, las partes interesadas del proyecto y los propietarios de empresas deben conocer y aceptar los riesgos de ejecutar el sistema sin realizar más pruebas. Dado que estas situaciones ocurren a menudo en la práctica, los evaluadores (especialmente la persona que desempeña el rol de líder del equipo de pruebas) deben proporcionar información que describa el estado actual del sistema, destacando los riesgos.

Los criterios de salida suelen adoptar la forma de medidas de minuciosidad o integridad.

Expresan el grado deseado de realización de un trabajo determinado. Los criterios de salida nos permiten determinar si hemos realizado determinadas tareas según lo previsto.

Ejemplo El equipo adoptó los siguientes criterios de salida de las pruebas del sistema:

- (EX1) logró una cobertura del 100% de los requisitos mediante casos de prueba
- (EX2) logró al menos un 75% de cobertura de declaración para cada componente probado
- (EX3) no hay defectos abiertos (no reparados) con el nivel más alto de gravedad
- (EX4) como máximo dos defectos abiertos de gravedad media o baja

Después de analizar el informe al final de la fase de prueba del sistema, quedó claro que:

- Para cada requisito, se diseñó al menos un caso de prueba.
- Para dos de los cuatro módulos, se logró una cobertura completa de la declaración; para el tercero, 80%; y para el cuarto, el 70%
- Una de las pruebas detectó un defecto de prioridad media que aún no está cerrado

Este análisis muestra que se cumplen los criterios (EX1), (EX3) y (EX4), mientras que no se cumple el criterio (EX2) para uno de los componentes. El equipo decidió analizar qué partes del código de este componente no están cubiertas y agregó un caso de prueba que cubre una ruta de flujo de control adicional en este módulo, lo que aumentó la cobertura de este módulo del 70 al 78 %. En este punto, se ha cumplido el criterio (EX2). Dado que en este punto se cumplen todos los criterios de salida de la fase de prueba del sistema, el equipo considera formalmente que esta fase está completa.

5.1.4 Técnicas de estimación

El esfuerzo de prueba es la cantidad de trabajo relacionado con las pruebas necesario para lograr los objetivos del proyecto de prueba. El esfuerzo de prueba es a menudo un factor de costo importante o significativo durante el desarrollo de software, y a veces consume más del 50% del esfuerzo total de desarrollo [64]. Una de las preguntas más comunes que los miembros del equipo escuchan de sus gerentes es "¿cuánto tiempo llevará realizar esta tarea?" Los problemas descritos anteriormente hacen la prueba.

La estimación es una actividad muy importante desde la perspectiva de la gestión de pruebas. Cada evaluador debe tener la capacidad de estimar el esfuerzo de la prueba, ser capaz de utilizar técnicas de estimación apropiadas, comprender las ventajas y desventajas de los diferentes enfoques y ser consciente de cuestiones como la precisión de la estimación (error de estimación).

El esfuerzo de prueba se puede medir como intensidad de mano de obra, una medida del producto (tiempo * recursos). Por ejemplo, un esfuerzo de 12 días-persona significa que un trabajo determinado será realizado en 12 días por una persona, o en 6 días por dos personas, o en 4 días por tres personas, y así sucesivamente. En general, será un trabajo realizado en x días por y personas, donde $x \times y = 12$.

Sin embargo, hay que tener cuidado porque, en la práctica, las medidas de los productos "no escalan". Por ejemplo, si un equipo de tres personas realiza algún trabajo en 4 días, la intensidad laboral es de 12 días-persona. Sin embargo, si un gerente agrega una nueva persona al equipo, con la esperanza de que cuatro personas hagan el mismo trabajo en 3 días, puede sentirse decepcionado: un equipo tan grande aún puede hacer el trabajo en 4 días o incluso más. Esto se debe a una serie de factores, como un aumento en los esfuerzos de comunicación (más personas en el equipo) o la necesidad de que los miembros del equipo con más años de servicio dediquen tiempo a formar a nuevos miembros, lo que puede causar retrasos. Los directores de proyecto tienen un dicho que ilustra este fenómeno: "una mujer tendrá un bebé en nueve meses, pero nueve mujeres no tendrán un bebé en un mes".

Al realizar una estimación del esfuerzo, a menudo no se dispone de todo el conocimiento sobre el objeto de prueba, por lo que la precisión de la estimación puede ser limitada. Es importante explicar a las partes interesadas que la estimación se basa en muchos supuestos. La estimación se puede refinar y posiblemente ajustar más adelante (por ejemplo, cuando haya más datos disponibles). Una posible solución es utilizar un rango de estimación para proporcionar una estimación inicial (por ejemplo, el esfuerzo será de 10 meses-persona con una desviación estándar de 3 meses-persona, lo que significa que con alta probabilidad, el esfuerzo real será de entre 10 y 3 meses). = 7 y 10 + 3 = 13 meses-persona).

La estimación de tareas pequeñas suele ser más precisa que la de las grandes. Por lo tanto, al estimar una tarea compleja, se puede utilizar una técnica de descomposición llamada estructura de desglose del trabajo (WBS). En esta técnica, la tarea o actividad principal (compleja) que se va a estimar se descompone jerárquicamente en subtareas más pequeñas (más simples). El objetivo es dividir la tarea principal en componentes fácilmente estimables pero ejecutables. La tarea debe dividirse con la mayor precisión posible basándose en la información actual, pero sólo en la medida necesaria para comprender y estimar con precisión una sola subtarea. Después de estimar todas las subtareas en el nivel más bajo, las tareas de nivel superior se estiman (sumando las estimaciones de las subtareas relevantes) de manera ascendente. En el último paso se obtiene la estimación de la tarea principal.

En la figura 5.2 se muestra un ejemplo de uso del método WBS .

Queremos estimar el esfuerzo de todo el proyecto de prueba. Sin embargo, es tan grande que no tiene sentido estimar todo el esfuerzo de una vez, ya que el resultado estará sujeto a un error muy grande. Entonces dividimos el proyecto de prueba en las tareas principales: planificación, definición del entorno de prueba, pruebas de integración y pruebas del sistema. Supongamos que podemos estimar con bastante precisión el esfuerzo para las dos primeras tareas (1 y 4 días-persona, respectivamente). Sin embargo, las pruebas de integración requerirán mucho esfuerzo, por lo que dividimos jerárquicamente esta tarea en tareas más pequeñas. Identificamos dos

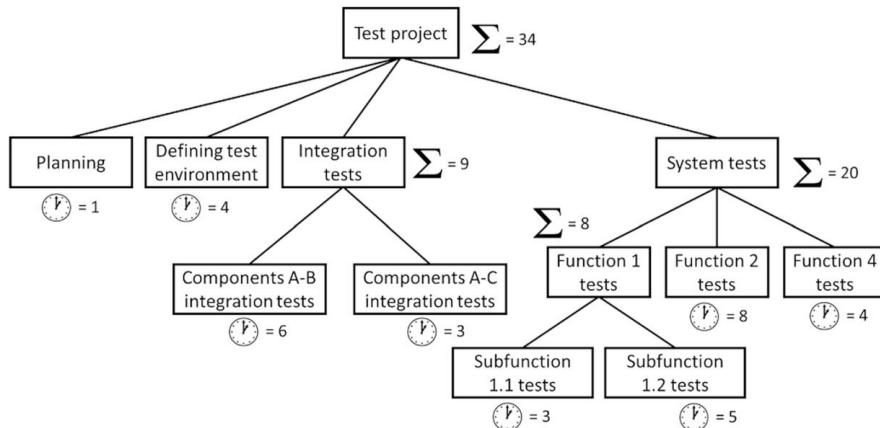


Fig. 5.2 Uso del método WBS para estimar el esfuerzo del proyecto de prueba

subtareas dentro de las pruebas de integración: integración de los componentes A y B (6 días-persona) e integración de los componentes A y C (3 días-persona). En las pruebas del sistema, identificamos tres funciones, F1, F2 y F3, como objetos de prueba. Para F2 y F3, estimamos que el esfuerzo de prueba será de 8 y 4 días-persona, respectivamente. La función F1 es demasiado grande, por lo que la dividimos en dos subtareas y estimamos el esfuerzo relacionado en 3 y 5 días-persona, respectivamente. Ahora podemos recopilar los resultados de los niveles más bajos y estimar las tareas en los niveles superiores sumando las subtareas correspondientes. Por ejemplo, el esfuerzo para las pruebas de integración (9 días-persona) será la suma del esfuerzo de dos subtareas con un esfuerzo de 6 y 3 días-persona, respectivamente. De manera similar, el esfuerzo para F1 es $3+5 = 8$ días-persona, y el esfuerzo para probar el sistema es $8+8+4 = 20$ días-persona. Al final, calculamos el esfuerzo de todo el proyecto de prueba como la suma de sus subtareas: $1+4+9+20 = 34$ días-persona.

Las técnicas de estimación se pueden dividir en dos grupos principales:

- Técnicas basadas en métricas: donde el esfuerzo de prueba se basa en métricas de proyectos similares anteriores, en datos históricos del proyecto actual o en valores típicos (las llamadas líneas de base de la industria).
- Técnicas basadas en expertos: donde el esfuerzo de la prueba se basa en la experiencia de los propietarios de las tareas de prueba o de los expertos en la materia.

El programa de estudios describe las siguientes cuatro técnicas de estimación utilizadas con frecuencia en la práctica.

Estimación basada en ratios

En esta técnica basada en métricas, se recopila la mayor cantidad posible de datos históricos de proyectos anteriores, lo que permite derivar ratios "estándar" de varios indicadores para proyectos similares. Los ratios propios de una organización suelen ser la mejor fuente para utilizar en el proceso de estimación. Estos índices estándar se pueden utilizar luego para estimar el esfuerzo de prueba para un nuevo proyecto. Por ejemplo, si en un proyecto similar anterior la proporción de

El esfuerzo de implementación respecto al esfuerzo de prueba fue de 3:2, y en el proyecto actual se espera que el esfuerzo de desarrollo sea de 600 días-persona, el esfuerzo de prueba se puede estimar en 400 días-persona, ya que la misma o similar proporción de implementación a prueba Lo más probable es que la prueba se realice como en el proyecto anterior similar.

Extrapolación En

esta técnica basada en métricas, las mediciones se toman lo antes posible para recopilar datos históricos reales del proyecto actual. Con suficientes observaciones de este tipo (puntos de datos), el esfuerzo requerido para el trabajo restante se puede aproximar extrapolando estos datos. Este método es muy útil en técnicas de desarrollo de software iterativo. Por ejemplo, un equipo puede extraer el esfuerzo de prueba en la cuarta iteración como un promedio del esfuerzo en las últimas tres iteraciones. Si el esfuerzo en las últimas tres iteraciones fue de 30, 32 y 25 días-persona, respectivamente, la extrapolación del esfuerzo para la cuarta iteración sería $(30+32+25) / 3 = 29$ días-persona.

Tenga en cuenta que, procediendo de manera análoga, podríamos estimar el esfuerzo para iteraciones posteriores utilizando la extrapolación calculada para iteraciones anteriores. En nuestro ejemplo, la extrapolación del esfuerzo para la quinta iteración será el promedio de las iteraciones 2, 3 y 4, por lo que será $(32+25+29) / 3 = 14,66$. De forma similar, podemos extraer el esfuerzo para la sexta iteración: $(25 + 29 + 14,66) / 3$ y así sucesivamente. Sin embargo, tenga en cuenta que cuanto más "más" sea el punto de datos que extrapolamos, mayor será el riesgo de cometer un error de estimación creciente, ya que la primera estimación ya tiene algún error. Aplicar dicho resultado a la siguiente estimación puede hacer que el error aumente (reducir la precisión del valor estimado), porque los errores pueden acumularse.

Wideband Delphi En el

método Wideband Delphi basado en expertos, los expertos hacen estimaciones basadas en su propia experiencia. Cada experto, de forma aislada, estima la carga de trabajo. Se recopilan los resultados y los expertos discuten sus estimaciones actuales. Luego se pide a cada experto que haga nuevas predicciones basadas en esta información. La discusión permite a todos los expertos repensar su proceso de estimación. Puede ser, por ejemplo, que algunos expertos no hayan tenido en cuenta ciertos factores al realizar la estimación. El proceso se repite hasta que se alcanza un consenso o hay un rango suficientemente pequeño en las estimaciones obtenidas.

En esta situación, por ejemplo, la media o mediana de las estimaciones de los expertos puede considerarse el resultado final.

La figura 5.3 muestra la idea detrás del método Delphi. De iteración en iteración, el rango de valores estimados por los expertos se estrecha. Una vez que sea lo suficientemente pequeño, se puede extraer, por ejemplo, la media o mediana de todas las estimaciones y considerarla el resultado del proceso de estimación. Muy a menudo, este resultado no se desviará demasiado del valor real. Este fenómeno se conoce coloquialmente como "sabiduría de la multitud" y resulta del simple hecho de que los errores se anulan entre sí: un experto puede sobreestimar un poco, otro puede subestimar un poco, otro puede subestimar mucho y otro más puede subestimar un poco. Puede sobreestimar mucho. Por tanto, el error suele tener una distribución normal con la media igual a cero. La dispersión de los resultados puede ser muy grande, pero promediálos será a menudo una aproximación suficiente del valor real.

Una variación del método Delphi es el llamado póquer de planificación. Este es un enfoque comúnmente utilizado en metodologías ágiles. Al planificar el póquer, las estimaciones son

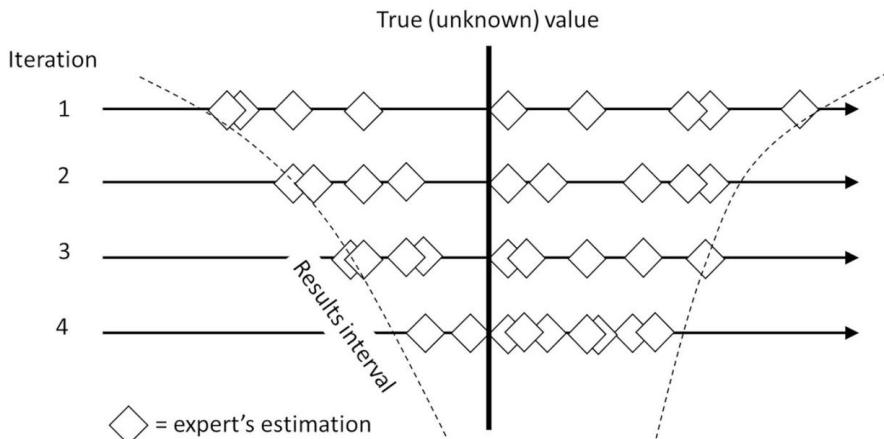


Fig. 5.3 El método Delphi de banda ancha

Realizado con tarjetas numeradas. Los valores de las tarjetas representan la cantidad de esfuerzo expresada en unidades específicas que están bien definidas y comprendidas por todos los expertos.

Las cartas de póquer de planificación suelen contener valores derivados de la llamada secuencia de Fibonacci, aunque para valores mayores puede haber algunas desviaciones. En la secuencia de Fibonacci, cada término sucesivo es la suma de los dos anteriores. Los siguientes valores se utilizan con mayor frecuencia:

1, 2, 3, 5, 8, 13, 20, 40, 100

Como puedes ver, los dos últimos valores ya no son la suma de los dos anteriores. Esto se debe a que son lo suficientemente grandes como para asumir aquí valores redondeados, simplemente representando algo lo suficientemente grande como para que ni siquiera tenga sentido estimar dicho valor con precisión. Si un equipo estima, por ejemplo, el esfuerzo necesario para implementar y probar alguna historia de usuario y la mayoría de los expertos tiran una carta de 40 o 100 sobre la mesa, esto significa que la historia es lo suficientemente grande como para que probablemente sea una épica y deba desglosarse. en varias historias más pequeñas, cada una de las cuales ya puede estimarse razonablemente.

En la figura 5.4 se muestra un ejemplo de una baraja de cartas de póquer de planificación . Si una tarjeta con "?" es arrojado, significa que el experto no tiene suficiente información para hacer la estimación. Un valor de "0", por otro lado, significa que el experto considera que la historia es trivial de implementar y que el tiempo dedicado a ella será insignificante. Una tarjeta con una imagen de café significa "Estoy cansado, ¡hagamos un descanso y tomemos un café!".

Otros conjuntos de valores utilizados con frecuencia en la planificación del póker son el conjunto de potencias sucesivas de dos:

1, 2, 4, 8, 16, 32, 64, 128

o las llamadas tallas de camiseta.

XS, S, M, L, XL.

En el último caso, la medición (estimación) no se realiza en una escala numérica de relación y, por lo tanto, no representa directamente el tamaño físico del objeto estimado.



Fig. 5.4 Planificación de cartas de póquer

esfuerzo. Las tallas de camisetas se definen únicamente en una escala ordinal, lo que permite únicamente comparar elementos entre sí. Por lo tanto, podemos decir que un piso estimado como "L" requiere más mano de obra que uno estimado como "S", pero no podemos decir cuántas veces más esfuerzo se necesitará para implementar un piso de tamaño "L" en relación con uno. de talla "S". En esta variante del método, sólo podemos priorizar historias por esfuerzo, agrupándolas en cinco grupos de tamaño de esfuerzo creciente. Sin embargo, esta no es una práctica recomendada, porque desde el punto de vista de la teoría de la medición, es importante que las diferencias entre los sucesivos grados de la escala sean constantes.

En el caso de que la escala exprese puntos de historia, no hay problema: la diferencia entre 4 y 5 puntos es la misma que entre 11 y 12 puntos y es de 1 punto.

Sin embargo, esto ya no es tan obvio en el caso de una escala como "tallas de camisa". Así que se debe suponer que, por ejemplo, la diferencia entre S y XS es la misma que entre L y M, y así sucesivamente.

También se aceptan otras escalas. Cuál usar específicamente es una decisión del equipo o de la gerencia. Sólo es importante entender cómo una unidad de esta escala se traduce en una unidad de esfuerzo. A menudo, los equipos toman como unidad el llamado punto de la historia del usuario.

Entonces, una unidad puede significar la cantidad de esfuerzo necesario para implementar/probar una historia de usuario promedio. Si el equipo sabe por experiencia, por ejemplo, que implementar una historia de usuario de 1 punto de historia generalmente les lleva 4 días-persona, entonces un componente estimado en 5 puntos de historia de usuario debería requerir aproximadamente $5 * 4 = 20$ días-persona de esfuerzo.

Estimación de tres puntos

En esta técnica basada en expertos, los expertos realizan tres tipos de estimación: la más optimista (a), la más probable (m) y la más pesimista (b). La estimación final (E) es su media aritmética ponderada calculada como

$$E = \frac{1}{3}a + \frac{4}{3}m + \frac{1}{3}b$$

La ventaja de esta técnica es que permite a los expertos calcular directamente también el error de medición:

$$DE = \sqrt{\frac{1}{3}(b-a)^2}$$

Por ejemplo, si las estimaciones (en horas-persona) son a = 6, m = 9 y b = 18, la estimación final es 10 ± 2 horas-persona (es decir, entre 8 y 12 horas-persona), porque

$$mi = \frac{1}{3}a + \frac{4}{3}m + \frac{1}{3}b = 10$$

y

$$DE = \sqrt{\frac{1}{3}(b-a)^2} = \sqrt{\frac{1}{3}(18-6)^2} = 2$$

Tenga en cuenta que el resultado de la estimación no siempre será igual al valor más probable m, porque dependerá de qué tan lejos esté este valor de los valores optimistas y pesimistas. Cuanto más cerca esté el valor de m del valor de estimación extremo (a o b), mayor será la distancia entre el valor de m y el resultado de la estimación E.

La fórmula que se muestra arriba es la más utilizada en la práctica. Se deriva del método de la técnica de evaluación y revisión de programas (PERT). Sin embargo, los equipos a veces usan otras variantes, ponderando el factor m de manera diferente, por ejemplo, con un peso de 3 o 5. Entonces la fórmula toma la forma $E = (a + 3m + b) / 5$ o $E = (a + 5m + b) / 7$.

Para obtener más información sobre estas y otras técnicas de estimación, consulte [45, 65, 66].

Ejemplo Un equipo utiliza un póquer de planificación para estimar la implementación y el esfuerzo de prueba de una determinada historia de usuario. El poker de planificación se juega en sesiones según el siguiente procedimiento.

1. Cada jugador tiene un juego completo de cartas.
2. El moderador (generalmente el propietario del producto) presenta la historia del usuario estimado.
3. A continuación se hace una breve discusión del alcance del trabajo, aclarando ambigüedades.
4. Cada experto elige una carta.
5. A la señal del facilitador, todos arrojan simultáneamente su tarjeta al centro de la mesa.
6. Si todos han elegido la misma carta, se llega a un consenso y la reunión finaliza; el resultado es el valor elegido por todos los miembros del equipo. Si después de un número fijo

Después de iteraciones, el equipo no ha llegado a un consenso, el resultado final se determina de acuerdo con la variante establecida y aceptada del procedimiento (ver más abajo).

7. Si las estimaciones difieren, la persona que eligió la estimación más baja y la persona que eligió la más alta presentan su punto de vista.
8. Si es necesario, sigue una breve discusión.
9. Volver al punto 4.

La planificación del poker, como cualquier técnica de estimación basada en el juicio de expertos, sólo será tan efectiva como lo sean los buenos expertos que participan en la sesión de estimación (ver Fig. 5.5).

Por lo tanto, las personas que realicen estimaciones deben ser expertos, preferiblemente con años de experiencia.

El equipo tiene las siguientes opciones posibles para determinar la estimación final cuando no se ha alcanzado un consenso después de un cierto número de iteraciones de póquer.

Supongamos que el equipo que realiza la estimación del esfuerzo para una determinada historia de usuario está formado por cinco personas y cada participante tiene una baraja completa de cartas con valores 0, 1, 2, 3, 5, 8, 13, 20, 40 y 100. Supongamos que en la última ronda, cinco personas presenten simultáneamente una carta de su elección. Los resultados son:

3, 8, 13, 13, 20.

Variante 1 Se promedia el resultado y la estimación final es el promedio (11,4) redondeado al valor más cercano del mazo, que es 13.

Variante 2 Se elige el valor más común que es 13.

Variante 3 El resultado es la mediana (valor medio) de los valores 3, 8, 13, 13 y 20, es decir, 13.

Variante 4 Además de proporcionar una estimación, cada persona también califica el grado de confianza en esa estimación, en una escala de 1 (alta incertidumbre) a 5 (alta confianza). El grado de certeza puede reflejar, por ejemplo, el nivel de experiencia y conocimientos en el área en la que se realiza la estimación. Supongamos que los estimadores 3, 8, 13, 13 y 20 calificaron la certeza de su elección en 4, 5, 2, 3 y 2, respectivamente.

Luego, la puntuación se calcula como un promedio ponderado:

$$\frac{3 \cdot 4 + 8 \cdot 5 + 13 \cdot 2 + 13 \cdot 3 + 20 \cdot 2}{4 + 5 + 2 + 3 + 2} = \frac{12}{12} = 1$$

Luego, el resultado se puede redondear al valor más cercano en la escala, que es 8. La ventaja de este enfoque es que los resultados de aquellos más convencidos de la exactitud de su estimación pesan más en la evaluación promediada. Por supuesto, la precisión del método dependerá en particular de la precisión de las puntuaciones de confianza de los evaluadores individuales.

Las opciones anteriores no agotan todas las formas posibles de estimación. Cada equipo puede adoptar sus propias reglas individuales para determinar el valor final del parámetro que se somete al procedimiento de estimación. También hay que tener en cuenta que todos estos

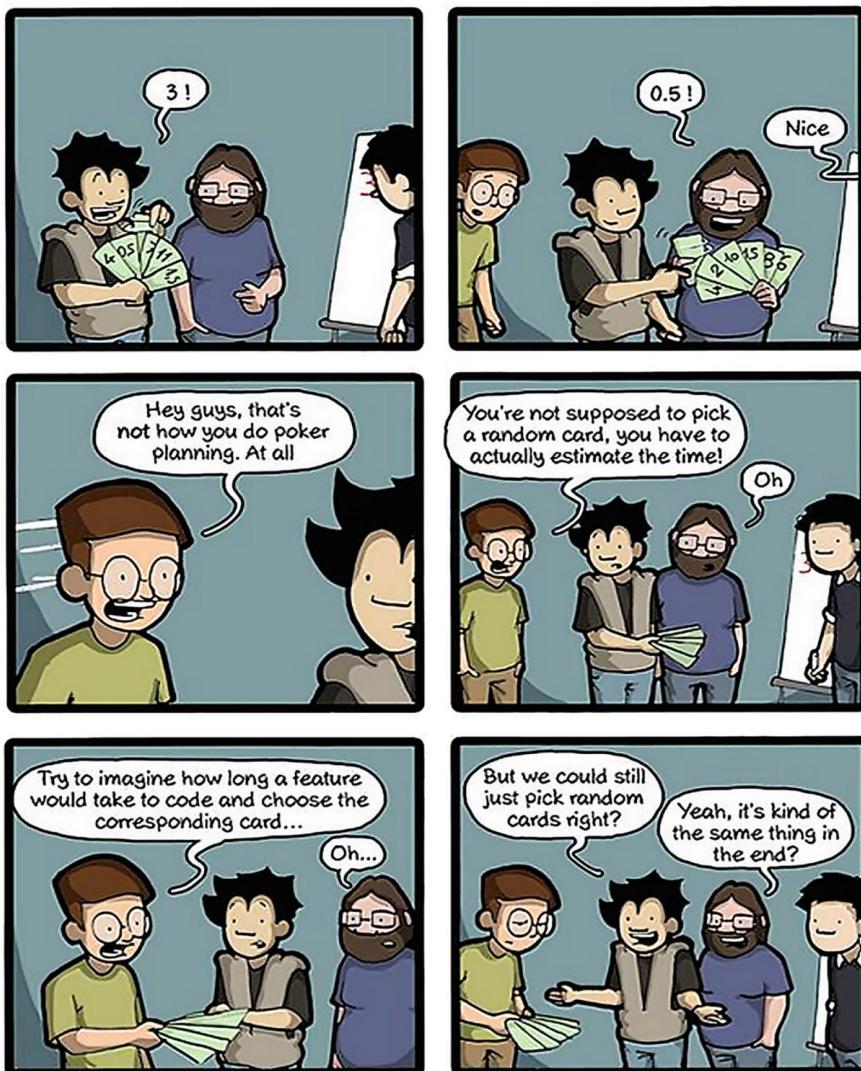


Fig. 5.5 Con humor sobre la planificación del poker (www.commitstrip.com/en/2015/01/22/poker-planning/)

Las variantes para seleccionar la respuesta final en ausencia de consenso siempre están sujetas a algún error. Por ejemplo, tomar la mediana (13) de los valores 8, 13, 13, 13, 13 y 13 estará sujeto a menos error que, por ejemplo, de los valores 1, 8, 13, 40 y 100, debido a la varianza mucho mayor de los resultados en el último caso.

Factores que afectan el esfuerzo de la prueba.

Los factores que afectan el esfuerzo de prueba incluyen:

- Características del producto (p. ej., riesgos del producto, calidad de las especificaciones (es decir, base de prueba), tamaño del producto, complejidad del dominio del producto, requisitos para las características de calidad (p. ej., seguridad y confiabilidad), nivel requerido de detalle en la documentación de prueba, requisitos regulatorios, requisitos de conformidad)
- Características del proceso de desarrollo de software (p. ej., estabilidad y madurez de la organización, modelo SDLC utilizado, enfoque de prueba, herramientas utilizadas, proceso de prueba, presión de tiempo)
- Factores humanos (p. ej., habilidades y experiencia de los evaluadores, cohesión del equipo, capacidad del gerente). habilidades)
- Resultados de las pruebas (p. ej., número, tipo e importancia de los defectos detectados, número de correcciones requeridas, número de pruebas fallidas)

La información sobre los tres primeros grupos de factores generalmente se conoce de antemano, a menudo antes de iniciar el proyecto. Los resultados de las pruebas, por otro lado, son un factor que funciona "desde adentro": no podemos incluir los resultados de las pruebas en la estimación antes de ejecutarlas. Por lo tanto, sus resultados pueden afectar significativamente la estimación más adelante en el proyecto.

5.1.5 Priorización de casos de prueba

Una vez que los casos de prueba y los procedimientos de prueba se crean y organizan en conjuntos de pruebas, estos conjuntos se pueden organizar en un programa de ejecución de pruebas, que define el orden en el que se ejecutarán. Se deben tener en cuenta varios factores al priorizar los casos de prueba (y, por lo tanto, el orden en que se ejecutan).

Un cronograma es una forma de planificar tareas a lo largo del tiempo. El cronograma debe tener en cuenta:

- Prioridades •
- Dependencias • La necesidad de pruebas de confirmación y regresión • El orden más efectivo para realizar las pruebas

Al crear un cronograma de ejecución de pruebas, es importante considerar:

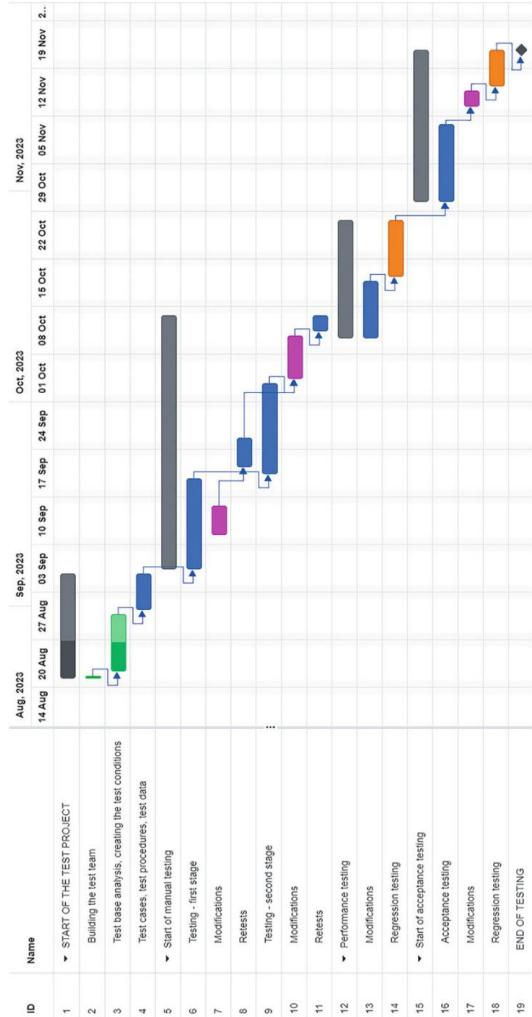
- Las fechas de las actividades centrales dentro del cronograma del proyecto • Que el cronograma esté dentro del cronograma del proyecto • Los hitos: inicio y fin de cada etapa

Una de las formas gráficas más típicas de presentar el cronograma es el llamado diagrama de Gantt (ver Fig. 5.6). En este gráfico, las tareas se representan como rectángulos que expresan su duración y las flechas definen diferentes tipos de relaciones entre tareas.

Por ejemplo, si una flecha va desde el final del rectángulo X hasta el comienzo del rectángulo Y, significa que la tarea Y solo puede comenzar cuando la tarea X haya finalizado.

Las pruebas bien planificadas deben tener lo que se conoce como una tendencia lineal: si conecta los hitos de "inicio del proyecto de prueba" y "fin de las pruebas" de la figura 5.6 con una línea, las tareas realizadas deben alinearse a lo largo de esta línea. Esto significa que las tareas relacionadas con la prueba

5.1 Planificación de pruebas



se realizan de tal manera que el equipo de pruebas trabaja a un ritmo constante, sin persecuciones de tiempo innecesarias.

La práctica muestra que el período de prueba final (>95% de los casos) requiere que se comprometa la cantidad máxima de recursos en la fase de prueba final (la fase final del proyecto); consulte la figura 5.7.

Las estrategias de priorización de casos de prueba más comunes son las siguientes.

- Priorización basada en riesgos, donde el orden de ejecución de las pruebas se basa en los resultados del análisis de riesgos (ver Sección 5.2.3). Primero se ejecutan los casos de prueba que cubren los riesgos más importantes.
 - Priorización basada en cobertura, donde el orden de ejecución de la prueba se basa en la cobertura (por ejemplo, cobertura de código, cobertura de requisitos). Los casos de prueba que alcanzan la mayor cobertura se ejecutan primero. En otro enfoque, llamado priorización basada en cobertura adicional, el caso de prueba que logra la cobertura más alta se ejecuta primero. Cada caso de prueba posterior es el que logra la mayor cobertura adicional.
 - Priorización basada en requisitos, donde el orden en que se ejecutan las pruebas se basa en prioridades de requisitos que están vinculadas a los casos de prueba correspondientes.
- Las prioridades de los requisitos las determinan las partes interesadas. Primero se ejecutan los casos de prueba relacionados con los requisitos más importantes.
- Idealmente, los casos de prueba deberían ordenarse para ejecutarse en función de sus niveles de prioridad, utilizando, por ejemplo, una de las estrategias de priorización mencionadas anteriormente. Sin embargo, es posible que esta práctica no funcione si los casos de prueba o las funciones que se prueban son dependientes. Aquí se aplican las siguientes reglas:

- Si un caso de prueba de mayor prioridad depende de un caso de prueba de menor prioridad, el caso de prueba de menor prioridad El caso de prueba prioritario debe ejecutarse primero.
- Si existen interdependencias entre varios casos de prueba, el orden de sus La ejecución debe determinarse independientemente de las prioridades relativas. •

Es posible que sea necesario priorizar las ejecuciones de pruebas de confirmación y regresión. •

Se debe equilibrar adecuadamente la eficiencia de la ejecución de las pruebas versus el cumplimiento de las prioridades establecidas.

El orden en que se ejecutan las pruebas también debe tener en cuenta la disponibilidad de recursos.

Por ejemplo, es posible que las herramientas, los entornos o las personas necesarios solo estén disponibles dentro de un período de tiempo determinado.

Ejemplo Estamos probando un sistema cuyo CFG se muestra en la figura 5.8. Estamos adoptando un método de priorización de pruebas basado en la cobertura. El criterio que utilizaremos será la cobertura de declaraciones. Supongamos que tenemos cuatro casos de prueba TC1-TC4 definidos en la Tabla 5.1.

Esta tabla también proporciona la cobertura que logra cada caso de prueba. La cobertura más alta (70%) la logra el TC4, seguida del TC1 y TC2 (60% cada uno), y la más baja la del TC3 (40%). En consecuencia, la prioridad de ejecución de la prueba será la siguiente: TC4, TC2, TC1 y TC3, donde TC1 y TC2 se pueden ejecutar en cualquier orden ya que logran la misma cobertura.

5.1 Planificación de pruebas

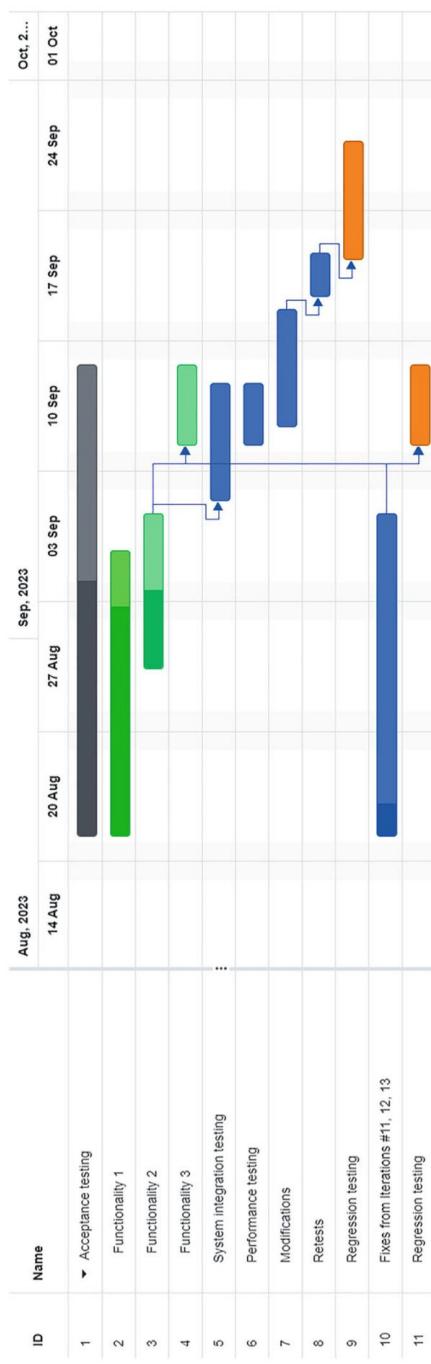


Fig. 5.8 CFG del sistema bajo prueba

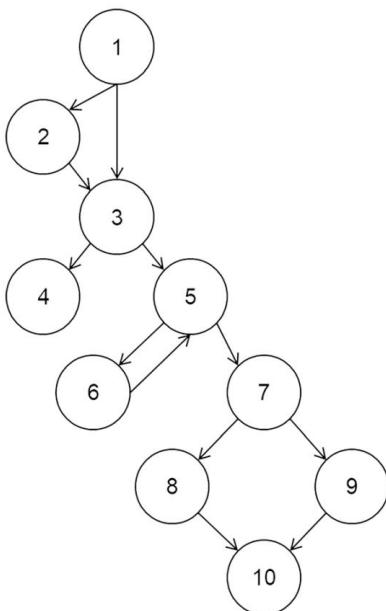


Tabla 5.1 Casos de prueba existentes y cobertura que logran

Caso de prueba	Camino ejercido	Cobertura	Prioridad
TC1	1→3→5→7→8→10	60%	2
TC2	1→3→5→7→9→10	60%	2
TC3	1→2→3→4	40%	3
TC4	1→3→5→6→5→7→8→10	70%	1

Sin embargo, tenga en cuenta que la ejecución de TC1, después de la ejecución de TC4 y TC2, no cubrirá ninguna declaración nueva. Entonces apliquemos una variante de priorización basada en cobertura adicional. Se dará máxima prioridad, como en la variante anterior, a el caso de prueba que logra la mayor cobertura, a saber, TC4. Ahora veamos cuantos declaraciones adicionales no cubiertas por TC4 están cubiertas por los otros casos de prueba.

El cálculo se muestra en la Tabla 5.2 en la penúltima columna. TC1 no cubrir cualquier declaración adicional, porque las declaraciones que cubre (es decir, 1, 3, 5, 7, 8, 10) ya están cubiertos por el TC4. TC2, respecto a TC4, cubre adicionalmente estado de cuenta 9, por lo que logra un 10% de cobertura adicional (cubrió un estado de cuenta nuevo de diez en total). TC3, en relación con TC4, cubre adicionalmente dos afirmaciones, 2 y 4, lo que resultó en una cobertura adicional del 20% (cubrió dos declaraciones nuevas de cada diez total). Por lo tanto, el siguiente caso de prueba después del TC4 que cubre la mayoría de las declaraciones nuevas aún no cubierto es TC3. Repetimos ahora el análisis anterior contando la cobertura adicional en relación con la cobertura lograda por TC4 y TC3. El cálculo se muestra en la última columna de la Tabla 5.2. TC1 no cubre nada nuevo. TC2, por su parte, cubre una nueva declaración que TC4 y TC3 no cubrieron, a saber, 9. Por lo tanto, logra una

Tabla 5.2 Cálculo de la cobertura adicional lograda mediante pruebas

Prueba caso Camino ejecutado	Cobertura	Adicional cobertura después de TC4	Cobertura adicional después de TC4 y TC3
TC1 1→3→5→7→8→10	60%	0%	0%
CT2 1→3→5→7→9→10	60%	10% (9)	10% (9)
CT3 1→2→3→4	40%	20% (2, 4)	-
TC4 1→3→5→6→5→7→8→10	70%	-	-

cobertura adicional del 10%. Así, obtuvimos la siguiente priorización según cobertura adicional:

TC4 → TC3 → TC2 → TC1.

Tenga en cuenta que este orden difiere del orden resultante de la variante anterior de el método. Su ventaja es que obtenemos cobertura de todas las declaraciones muy rápidamente (después de sólo las tres primeras pruebas). En la variante anterior (con el orden TC4→TC2→TC1→TC3), las afirmaciones 2 y 4 están cubiertas solo en la última, cuarta prueba.

Otro ejemplo muestra que a veces las dependencias técnicas o lógicas entre pruebas puede alterar el orden deseado de ejecución de las pruebas y obligarnos, por Por ejemplo, ejecutar primero una prueba de menor prioridad para "desbloquear" la posibilidad. de ejecutar una prueba de mayor prioridad.

Ejemplo Supongamos que estamos probando la funcionalidad de un sistema para realizar exámenes de conducción de forma electrónica. Supongamos también que al comienzo de cada ejecución de prueba, el La base de datos del sistema está vacía. Se han identificado los siguientes casos de prueba, junto con sus prioridades:

TC1: Agregar el examinado a la base de datos. Prioridad: baja

TC2: Realización del examen para el examinado. Prioridad: alta

TC3: Realización de un examen de revisión. Prioridad: media

CT4: Tomar una decisión y enviar los resultados del examen al examinado. Prioridad: medio

El caso de prueba con mayor prioridad aquí es TC2 (realizar el examen), pero No podemos ejecutarlo antes de agregar al examinado a la base de datos. Entonces, aunque TC2 tiene un mayor prioridad que TC1, la dependencia lógica que se produce requiere que ejecutemos Primero TC1, que "desbloqueará" la ejecución de la prueba para los otros casos de prueba. la próxima prueba El caso a ejecutar sería TC2, como tarea de mayor prioridad. Al final, podemos ejecutar TC3 y TC4. Tenga en cuenta que para aprobar un examen de revisión, el examinado debe saber que reprobaron el examen original. Por tanto, desde la perspectiva de un usuario particular, tiene sentido ejecutar TC3 sólo después de completar TC4. De este modo, La ejecución de la prueba final es la siguiente:

TC1 → TC2 → TC4 → TC3.

Tabla 5.3 Tareas con sus prioridades y dependencias

Tarea	Prioridad (1 = alta, 5 = baja)	Depende de
12		5
2	4	
3	1	2, 4
4	1	2, 1
53		6
63		7
7	5	

En muchas situaciones, puede suceder que el orden en el que se realizan las acciones Depende de varios factores, como la prioridad y las dependencias lógicas y técnicas. Existe un método sencillo para abordar este tipo de tareas, incluso si las dependencias parecen complicado. Consiste en apuntar a la ejecución lo antes posible de tareas de mayor prioridad, pero si están bloqueadas por otras tareas, debemos ejecutar esas tareas. primero. Dentro de estas tareas, también establecemos su orden según prioridades y dependencias. Consideremos este método con un ejemplo concreto.

Ejemplo En la Tabla 5.3 se proporciona una lista de tareas con sus prioridades y dependencias .

Empezamos por establecer un orden que dependa únicamente de prioridades, sin mirando las dependencias, con tareas de igual prioridad agrupadas:

3, 4 → 1 → 5, 6 → 2 → 7.

Dentro del primer grupo, la tarea 3 depende de la tarea 4. Por lo tanto, aclaramos el orden. de las tareas 3 y 4:

4 → 3 → 1 → 5, 6 → 2 → 7.

Comprobamos ahora si podemos realizar las tareas en este orden. Resulta que no podemos: La tarea 4 depende de las tareas 2 y 1, y la tarea 1 tiene mayor prioridad que la tarea 2. Entonces movemos ambas tareas antes de la tarea 4 con sus prioridades preservadas, y después de la tarea 4, mantenemos el orden original de las otras tareas:

1 → 2 → 4 → 3 → 5, 6 → 7.

¿Podemos realizar las tareas en este orden? Todavía no: la tarea 1 depende de la tarea 5, esto en la tarea 6, y ésta a su vez en la tarea 7. Por lo tanto, debemos mover las tareas 5, 6 y 7 delante de tarea 1 con sus dependencias (tenga en cuenta que aquí las prioridades de las tareas 5, 6 y 7 no importa: el orden es forzado por la dependencia):

7 → 6 → 5 → 1 → 2 → 4 → 3.

Tenga en cuenta que en este punto ya podemos realizar la secuencia completa de tareas, 7, 6, 5, 1, 2, 4 y 3, ya que cada tarea en la secuencia anterior no depende de ninguna otra. tarea o depende sólo de las tareas que la preceden.

5.1.6 Pirámide de prueba

La pirámide de pruebas es un modelo que muestra que diferentes pruebas pueden tener diferente "granularidad". El modelo de pirámide de pruebas apoya al equipo en la automatización de pruebas y la asignación del esfuerzo de prueba. Las capas de la pirámide representan grupos de pruebas. Cuanto mayor sea la capa, menor será la granularidad de la prueba, el aislamiento de la prueba, la velocidad de ejecución de la prueba y el costo de calidad.

Las pruebas en la capa inferior de la pirámide de pruebas son pequeñas, aisladas y rápidas y verifican una pequeña parte de la funcionalidad, por lo que generalmente se necesitan muchas para lograr una cobertura razonable. La capa superior representa pruebas de extremo a extremo de gran tamaño y de alto nivel.

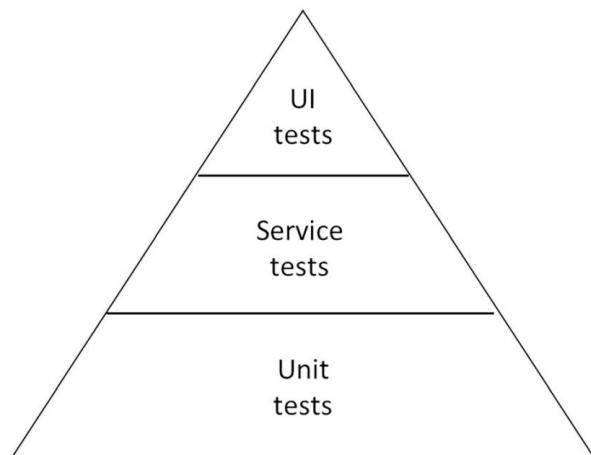
Estas pruebas de alto nivel son más lentas que las pruebas de las capas inferiores y normalmente comprueban una gran parte de la funcionalidad, por lo que normalmente sólo se necesitan algunas para lograr una cobertura razonable.

El número y el nombre de las capas en una pirámide de prueba pueden variar. Por ejemplo, el modelo de pirámide de pruebas original [67] define tres capas: "pruebas unitarias" (llamadas pruebas de componentes en el programa de estudios ISTQB®), "pruebas de servicio" y "pruebas de UI" (ver Fig. 5.9). Otro modelo popular define pruebas unitarias (de componentes), pruebas de integración y pruebas de un extremo a otro.

El costo de la ejecución de pruebas desde una capa inferior suele ser mucho menor en relación con el costo de la ejecución de pruebas desde una capa superior. Sin embargo, como se mencionó anteriormente, una sola prueba de una capa inferior generalmente logra una cobertura mucho menor que una sola prueba de una capa superior. Por lo tanto, en la práctica, en proyectos de desarrollo típicos, el esfuerzo de prueba se refleja bien en el modelo de pirámide de pruebas. El equipo normalmente escribe muchas pruebas de bajo nivel y pocas pruebas de alto nivel. Sin embargo, tenga en cuenta que la cantidad de pruebas de diferentes niveles siempre estará determinada por el contexto amplio del proyecto y producto.

Puede suceder, por ejemplo, que la mayoría de las pruebas que realizará el equipo sean pruebas de integración y aceptación (por ejemplo, en la situación de integrar dos productos comprados a una casa de software; en este caso, no tiene sentido realizar pruebas de componentes en todo).

Fig. 5.9 Pirámide de prueba



Aunque la pirámide de pruebas es ampliamente conocida, todavía hay bastantes organizaciones que le dan la vuelta a esta pirámide, centrándose en las pruebas más complejas (UI). Una variante es la pirámide inclinada, donde pruebas específicas y series de pruebas cortan la pirámide, siguiendo un enfoque funcional.

5.1.7 Prueba de cuadrantes

El modelo de cuadrantes de prueba [68], definido por Brian Marick [68], [21], alinea los niveles de prueba con tipos de prueba, actividades, técnicas y productos de trabajo relevantes en enfoques de desarrollo ágil. El modelo respalda la gestión de pruebas al garantizar que todos los tipos y niveles de prueba importantes se incluyan en el proceso de desarrollo de software y al comprender que algunos tipos de pruebas están más relacionados con ciertos niveles de prueba que otros. El modelo también proporciona una manera de distinguir y describir los tipos de pruebas para todas las partes interesadas, incluidos desarrolladores, evaluadores y representantes comerciales. El modelo del cuadrante de prueba se muestra en la figura 5.10.

En los cuadrantes de pruebas, las pruebas pueden dirigirse a las necesidades del negocio (usuario, cliente) o de la tecnología (desarrollador, equipo de fabricación). Algunas pruebas validan el comportamiento del software y, por tanto, respaldan el trabajo realizado por el equipo ágil; otros verifican (critican) el producto. Las pruebas pueden ser completamente manuales, completamente automatizadas, una combinación de

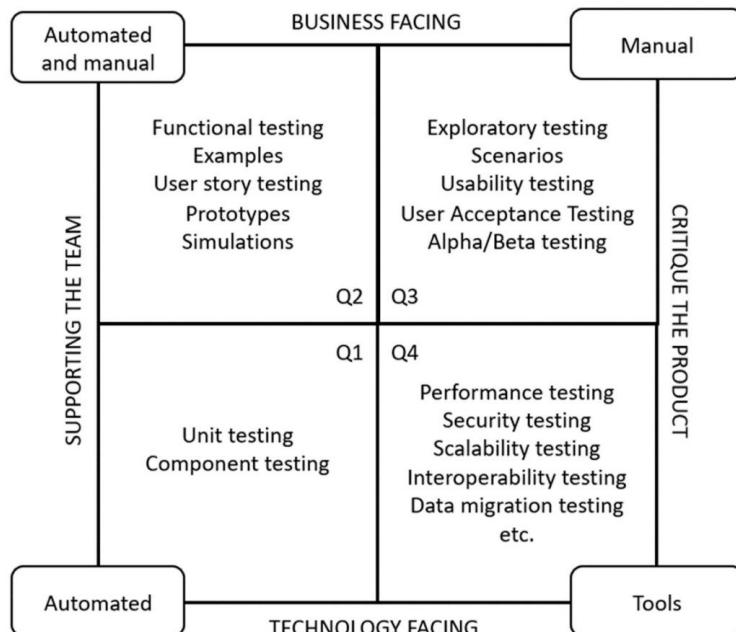


Fig. 5.10 Cuadrantes de prueba

manual y automatizado, o manual pero apoyado en herramientas. Los cuatro cuadrantes son los siguientes:

Cuadrante Q1 EI

cuadrante Q1 describe el nivel de prueba de los componentes, orientado a tecnología y soporte al equipo. Este cuadrante incluye pruebas de componentes. Estas pruebas deben automatizarse tanto como sea posible e integrarse en el proceso de integración continua.

Cuadrante Q2 EI

cuadrante Q2 describe el nivel de prueba del sistema, que se considera orientado al negocio y de apoyo al equipo. Este cuadrante incluye pruebas funcionales, ejemplos (ver Sección 4.5), pruebas de historias de usuario, prototipos y simulaciones. Estas pruebas verifican los criterios de aceptación y pueden ser manuales o automatizadas. A menudo se crean durante el desarrollo de historias de usuario, mejorando así su calidad. Son útiles al crear conjuntos de pruebas de regresión automatizadas.

Cuadrante Q3 EI

cuadrante Q3 describe un nivel de prueba de aceptación orientado al negocio que incluye pruebas de crítica de productos utilizando escenarios y datos realistas. Este cuadrante incluye pruebas exploratorias, pruebas basadas en escenarios (por ejemplo, pruebas basadas en casos de uso) o pruebas de flujo de procesos, pruebas de usabilidad y pruebas de aceptación del usuario, incluidas las pruebas alfa y beta. Estas pruebas suelen ser manuales y están orientadas al usuario.

Cuadrante Q4 EI

cuadrante Q4 describe un nivel de prueba de aceptación orientado a la tecnología que incluye pruebas de "crítica de producto". Este cuadrante contiene la mayoría de las pruebas no funcionales (excepto las pruebas de usabilidad). Estas pruebas suelen estar automatizadas.

Durante cada iteración, es posible que se requieran pruebas de cualquiera o todos los cuadrantes. Los cuadrantes de prueba se refieren a pruebas dinámicas en lugar de pruebas estáticas.

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 exhaustividad y alcance de las pruebas

FL-5.2.4 (K2) Explicar

qué medidas se pueden tomar en respuesta a los riesgos del producto analizados

Las organizaciones enfrentan muchos factores internos y externos que hacen que sea incierto si lograrán sus objetivos y cuándo [7]. La gestión de riesgos permite a las organizaciones aumentar la probabilidad de lograr sus objetivos, mejorar la calidad del producto y aumentar la confianza de las partes interesadas.

El nivel de riesgo generalmente se define por la probabilidad del riesgo y el impacto del riesgo (ver Sección).

5.2.1). Los riesgos, desde el punto de vista del evaluador, se pueden dividir en dos grupos principales:

riesgos del proyecto y riesgos del producto (ver Sección 5.2.2). Las principales actividades de gestión de riesgos incluyen:

- Análisis de riesgos (que consiste en la identificación y evaluación de riesgos; consulte la Sección).

5.2.3)

- Control de riesgos (que consiste en mitigación de riesgos y seguimiento de riesgos; consulte la Sección 5.2.4)

Un enfoque de las pruebas en el que las actividades de prueba se gestionan, seleccionan y priorizado en función del análisis de riesgos y el control de riesgos se denomina prueba basada en riesgos .

Uno de los muchos desafíos en las pruebas es la selección y priorización adecuadas de las condiciones de prueba. El riesgo se utiliza para enfocar y asignar adecuadamente el esfuerzo requerido durante la prueba de estas condiciones de prueba. Se utiliza para decidir dónde y cuándo comenzar las pruebas e identificar áreas que necesitan más atención. Las pruebas se utilizan para reducir el riesgo, es decir, para reducir la probabilidad de un evento adverso o para reducir el impacto de un evento adverso.

Varias formas de pruebas son una de las actividades típicas de mitigación de riesgos en proyectos de desarrollo de software (ver Sección 5.2.4). Proporcionan retroalimentación sobre los riesgos identificados y restantes (no resueltos).

El análisis temprano de riesgos del producto (ver Sección 5.2.3) contribuye al éxito del proyecto.

Las pruebas basadas en riesgos contribuyen a reducir los niveles de riesgo del producto y proporcionan información completa para ayudar a decidir si un producto está listo para su lanzamiento (este es uno de los principales objetivos de las pruebas; consulte la Sección 1.1.1). Normalmente, la relación entre el riesgo del producto y las pruebas se controla mediante un mecanismo de trazabilidad bidireccional.

Los principales beneficios de las pruebas basadas en riesgos son:

- Aumentar la probabilidad de descubrir defectos en orden de prioridad realizando pruebas en orden relacionado con la priorización de riesgos. • Minimizar el riesgo residual del producto después del lanzamiento asignando el esfuerzo de prueba de acuerdo. arriesgar
- Informar el riesgo residual midiendo los resultados de las pruebas en términos de niveles de riesgos relacionados.
- Contrarrestar los efectos de la presión del tiempo en las pruebas y permitir que el período de prueba se acorte con el menor aumento posible en el riesgo del producto asociado con una reducción en el tiempo asignado para las pruebas.

Para garantizar que se minimice la probabilidad de falla del producto, las actividades de prueba basadas en riesgos brindan un enfoque disciplinado para:

- Analizar (y reevaluar periódicamente) qué puede salir mal (riesgo) • Determinar qué riesgos son lo suficientemente importantes como para abordarlos • Implementar medidas para mitigar estos riesgos • Diseñar planes de contingencia para hacer frente a los riesgos cuando se produzcan

Además, las pruebas pueden identificar nuevos riesgos, ayudar a determinar qué riesgos deben mitigarse y reducir la incertidumbre relacionada con los riesgos.

5.2 Gestión de riesgos

5.2.1 Definición de riesgo y atributos de riesgo

Según el Glosario de términos de prueba de ISTQB® [42], el riesgo es un factor que ~~puede~~ tener consecuencias negativas en el futuro; generalmente se describe por impacto y probabilidad (probabilidad). Por lo tanto, el nivel de riesgo está determinado por la probabilidad de que ocurra ~~un~~ evento adverso y el impacto o consecuencias (el daño resultante del evento). Esto suele definirse mediante la ecuación:

$$\text{Nivel de riesgo} = \text{Probabilidad de riesgo} \times \text{Impacto del riesgo}$$

Esta ecuación puede entenderse simbólicamente (como se describió anteriormente) o de manera bastante literal. Nos ocupamos del último caso en el llamado enfoque cuantitativo del riesgo, donde la probabilidad y el impacto se expresan numéricamente: la probabilidad como un número en el intervalo (0, 1), mientras que el impacto, en dinero. El impacto representa la cantidad de pérdida que sufrirímos cuando el riesgo se materialice.

Ejemplo Una organización define un nivel de riesgo como el producto de su probabilidad e impacto. Se ha identificado el riesgo de que un informe clave se genere demasiado lentamente cuando hay una gran cantidad de usuarios conectados simultáneamente.

Dado que en este caso el requisito de un alto rendimiento es fundamental, se estimó que el impacto de este riesgo sería muy alto. Teniendo en cuenta la cantidad de usuarios que podrían experimentar retrasos en la generación de informes y las consecuencias de estos retrasos, el impacto del riesgo se estimó en 800.000 dólares.

Por otro lado, se estimó que la probabilidad de un rendimiento deficiente del programa era muy baja, debido a un proceso de prueba bastante bien definido, revisiones de arquitectura programadas y la amplia experiencia del equipo en pruebas de rendimiento. La probabilidad de riesgo se estimó en un 5%.

Finalmente, el nivel de riesgo según la ecuación anterior se fijó en:

$$5\% \times \$800\,000 = 0,05 \times \$800\,000 = \$40\,000$$

5.2.2 Riesgos del proyecto y riesgos del producto

Hay dos tipos de riesgos en las pruebas de software: riesgos del proyecto y riesgos del producto.

Riesgos del

proyecto Los riesgos ~~del~~ del proyecto están relacionados con la gestión y el control del proyecto. Los riesgos del proyecto son riesgos que afectan el éxito del proyecto (la capacidad del proyecto para lograr sus objetivos). Los riesgos del proyecto incluyen:

- Problemas organizacionales (p. ej., retrasos en la entrega de productos de trabajo, estimaciones inexactas, reducción de costos, procesos mal implementados, administrados y mantenidos, incluido el proceso de ingeniería de requisitos o control de calidad) • Problemas humanos (p. ej., habilidades insuficientes, conflictos, comunicación problemas, escasez de personal, falta de expertos en la materia disponibles) • Problemas técnicos (p. ej., variación del alcance; soporte deficiente de herramientas; estimación inexacta; cambios de último momento; clarificación insuficiente de los requisitos; incapacidad para cumplir con los requisitos debido a limitaciones de tiempo; falta de hacer que el entorno de prueba esté disponible a tiempo; programación tardía de la conversión de datos; migración o provisión de herramientas necesarias para esto; defectos en el proceso de fabricación, incluidas especificaciones de requisitos o casos de prueba u otras deudas técnicas;)
- Problemas relacionados con proveedores (por ejemplo, fallas en la entrega de terceros, quiebra del empresa de soporte, retraso en la entrega, problemas contractuales)

Los riesgos del proyecto, cuando ocurren, pueden afectar el cronograma, el presupuesto o el alcance del proyecto, lo que a su vez afecta la capacidad del proyecto para lograr sus objetivos. La consecuencia directa más común de los riesgos del proyecto es que el proyecto se retrasa, lo que trae consigo problemas adicionales, como mayores costos debido a la necesidad de asignar más tiempo a ciertas actividades o la necesidad de pagar sanciones contractuales por la entrega tardía de un producto.

Riesgos del

producto Los riesgos del producto están relacionados con las características de calidad de un producto, como funcionalidad, confiabilidad, rendimiento, usabilidad y otras características descritas, por ejemplo, por el modelo de calidad ISO/IEC 25010 [5]. Los riesgos del producto ocurren dondequiera que un producto de trabajo (por ejemplo, una especificación, componente, sistema o prueba) pueda no satisfacer las necesidades de los usuarios y/o partes interesadas. Los riesgos del producto se definen por áreas de posible falla en el producto bajo prueba, ya que amenazan la calidad del producto de varias maneras. Ejemplos de riesgos del producto incluyen:

- Funcionalidad faltante o inadecuada • Cálculos incorrectos • Fallos durante el funcionamiento del software (p. ej., la aplicación se cuelga o se cierra inactivo)
- Arquitectura deficiente
- Algoritmos inefficientes • Tiempo de respuesta inadecuado
- Mala experiencia de usuario • Vulnerabilidades de seguridad

El riesgo del producto, cuando ocurre, puede tener varias consecuencias negativas, que incluyen:

- Insatisfacción del usuario final
- Pérdida de ingresos • Daños causados a terceros • Altos costos de mantenimiento

5.2 Gestión de riesgos

- Sobre carga del servicio de asistencia técnica
- Pérdida de imagen
- Pérdida de confianza en el producto
- Sanciones penales

En casos extremos, la aparición de riesgos en el producto puede provocar daños físicos, lesiones e incluso la muerte.

5.2.3 Análisis de riesgos del producto

El propósito del análisis de riesgos es brindar conciencia sobre los riesgos para enfocar el esfuerzo de prueba de tal manera que se minimice el nivel de riesgo residual del producto. Idealmente, el análisis de riesgos del producto comienza temprano en el ciclo de desarrollo. El análisis de riesgos consta de dos fases principales: identificación de riesgos y evaluación de riesgos. La información de riesgo del producto obtenida en la fase de análisis de riesgo se puede utilizar para:

- Planificación de pruebas
- Especificación, preparación y ejecución de casos de prueba
- Monitoreo y control de pruebas

El análisis temprano de riesgos del producto contribuye al éxito de todo el proyecto, porque el análisis de riesgos del producto permite:

- Identificar niveles de prueba específicos y tipos de pruebas a realizar • Definir el alcance de las pruebas a realizar • Priorizar las pruebas (para detectar defectos críticos lo antes posible) • Seleccionar técnicas de prueba apropiadas que lograrán de manera más efectiva la cobertura establecida y detectarán defectos asociados con los riesgos identificados • Estimar para cada tarea la cantidad de esfuerzo puesto en las pruebas • Determinar si se pueden utilizar otras medidas además de las pruebas para reducir (mitigar) riesgos •

Establecer otras actividades no relacionadas con las pruebas

Identificación de riesgos

La identificación de riesgos implica generar una lista completa de riesgos. Las partes interesadas pueden identificar riesgos utilizando diversas técnicas y herramientas, tales como:

- Lluvia de ideas (para aumentar la creatividad en la búsqueda de riesgos) •
- Talleres de riesgos (para realizar conjuntamente la identificación de riesgos por parte de varias partes interesadas) • Método Delphi o evaluación de expertos (para obtener el acuerdo o desacuerdo de los expertos en riesgo)
- Entrevistas (para preguntar a las partes interesadas sobre los riesgos y recopilar sus opiniones) •
- Listas de verificación (para abordar los riesgos típicos, conocidos y que ocurren comúnmente)
- Bases de datos de proyectos pasados, retrospectivas y lecciones aprendidas (para abordar los riesgos pasados) • experiencia)

- Diagramas de causa y efecto (para descubrir riesgos mediante la realización de análisis de causa raíz) •

Plantillas de riesgo (para determinar quién puede verse afectado por los riesgos y cómo)

Evaluación de riesgos

La evaluación de riesgos implica categorizar los riesgos identificados; determinar su probabilidad, impacto y nivel de riesgo; priorizarlos; y proponer formas de afrontarlos. La categorización ayuda a asignar acciones de mitigación, ya que normalmente los riesgos de la misma categoría se pueden mitigar utilizando un enfoque similar.

La evaluación de riesgos puede utilizar un enfoque cuantitativo o cualitativo o una combinación de ambos. En un enfoque cuantitativo, el nivel de riesgo se calcula como el producto de la probabilidad y el impacto. En un enfoque cualitativo, el nivel de riesgo se puede calcular utilizando una matriz de riesgo.

En la figura 5.11 se muestra un ejemplo de matriz de riesgos. La matriz de riesgo es una especie de "tabla de multiplicar", en la que el nivel de riesgo se define como el producto de ciertas categorías de probabilidad e impacto.

En la matriz de riesgo de la figura 5.11, hemos definido cuatro categorías de probabilidad: baja, media, alta y muy alta, y tres categorías de impacto: bajo, medio y alto.

En la intersección de las categorías definidas de probabilidad e impacto, se define un nivel de riesgo. En esta matriz de riesgo se definen seis categorías de niveles de riesgo: muy bajo, bajo, medio, alto, muy alto y extremo. Por ejemplo, un riesgo con probabilidad media y alto impacto tiene un nivel de riesgo alto, porque

probabilidad media impacto alto = nivel de riesgo alto:

Hay otros métodos de riesgo en los que, como en el método de análisis modal de fallas y efectos (FMEA), por ejemplo, la probabilidad y el impacto se definen en una escala numérica (por ejemplo, del 1 al 5, donde 1 es el más bajo y 5 es el más alto). categoría más alta) y el nivel de riesgo se calcula multiplicando los dos números. Por ejemplo, para una probabilidad de 2 y un impacto de 4, el nivel de riesgo se define como $2 * 4 = 8$.

En tales casos, sin embargo, se debe tener cuidado. La multiplicación es conmutativa, por lo que el nivel de riesgo de una probabilidad de 1 y un impacto de 5 es 5 y es el mismo que para un riesgo con una probabilidad de 5 y un impacto de 1, porque $1 * 5 = 5 * 1$. Sin embargo, en la práctica, el impacto es

RISK LEVEL		RISK LIKELIHOOD			
		low	medium	high	very high
RISK IMPACT	low	very low	low	medium	medium
	medium	low	medium	high	high
	high	medium	high	very high	extreme

Fig. 5.11 Matriz de riesgos

un factor mucho más importante para nosotros. Un riesgo con una probabilidad de 5 y un impacto de 1 quizás ocurra muy a menudo, pero prácticamente no causará problemas debido a su impacto insignificante. Por el contrario, un riesgo con una probabilidad de 1 y un impacto de 5 ciertamente tiene una probabilidad muy pequeña de ocurrir, pero cuando ocurre, sus consecuencias pueden ser catastróficas.

Además de los enfoques cuantitativos y cualitativos para la evaluación de riesgos, también existe un enfoque mixto. En este enfoque, la probabilidad y el impacto no se definen mediante valores numéricos específicos, sino que se representan mediante rangos de valores que expresan cierta incertidumbre sobre la estimación de estos parámetros. Por ejemplo, la probabilidad se puede definir como el intervalo [0,2, 0,4] y el impacto, como el intervalo [\$1000, \$3000]. Esto significa que esperamos que la probabilidad de riesgo esté entre el 20% y el 40%, mientras que se espera que su impacto esté entre \$1000 y \$3000. El nivel de riesgo en este enfoque también se puede especificar como un rango, según la siguiente fórmula de multiplicación:

$$\frac{1}{2} a, b \frac{1}{2} c, d = \frac{1}{2} ac, bd$$

es decir, en nuestro caso, el nivel de riesgo será

$$\frac{1}{2} 0:2, 0:4 \quad \frac{1}{2} \$1000, \$3000 = \frac{1}{2} \$200, \$1200 :$$

Por lo tanto, el nivel de riesgo está entre \$200 y \$1200.

El enfoque mixto es una buena solución si nos resulta difícil estimar los valores exactos de las probabilidades y los impactos de los riesgos individuales (enfoque cuantitativo), pero al mismo tiempo queremos resultados de evaluación de riesgos más precisos que los expresados en una escala ordinal. (enfoque cualitativo).

Ejemplo Una organización define un nivel de riesgo como el producto de su probabilidad e impacto. Se definen los siguientes riesgos, junto con estimaciones de la probabilidad y el impacto de cada riesgo:

Riesgo 1: probabilidad del 20%, impacto de 40.000 dólares

Riesgo 2: probabilidad del 10%, impacto de 100.000 dólares

Riesgo 3: probabilidad del 5%, impacto de 20.000 dólares

Para calcular el nivel de riesgo total sumamos los respectivos productos: Nivel de riesgo total = $0,2 * \$40.000 + 0,1 * \$100.000 + 0,05 * \$20.000 = \$8000 + \$10.000 + \$1000 = \$19.000$.

Esto significa que si no tomamos ninguna medida para minimizar (mitigar) estos riesgos, la pérdida promedio (esperada) en la que incurriremos como resultado de su posible ocurrencia será de aproximadamente \$19,000.

Tenga en cuenta que el análisis de riesgo cuantitativo para un solo riesgo no tiene mucho sentido, porque los riesgos (desde el punto de vista del evaluador o de un observador externo) son fenómenos aleatorios. No sabemos cuándo ocurrirán o si ocurrirán y, de ser así, qué tipo de daño causarán realmente. Un análisis de un único fenómeno aleatorio con probabilidad P e impacto X no tiene sentido, porque ocurrirá o no, por lo que la pérdida será 0 o X (asumiendo que nuestra estimación del impacto fue correcta) y

nunca seas P*X. Sin embargo, un análisis realizado para múltiples riesgos, como en el ejemplo anterior, tiene mucho más sentido. Esto se debe a que la suma de los niveles de riesgo para todos los riesgos identificados puede tratarse como el valor esperado de la pérdida en la que incurriremos debido a la ocurrencia de algunos de estos riesgos.

5.2.4 Control de riesgos del producto

El control de riesgos del producto  juega un papel clave en la gestión de riesgos, ya que incluye todas las medidas que se toman en respuesta a los riesgos del producto identificados y evaluados. El control de riesgos se define como las medidas tomadas para mitigar y monitorear los riesgos a lo largo del ciclo de desarrollo. El control de riesgos es un proceso que incluye dos tareas principales: mitigación de riesgos y seguimiento de riesgos.

Mitigación de riesgos

Una vez analizados los riesgos, existen varias opciones para mitigarlos o posibles respuestas al riesgo [71,  72]:

- Aceptación de riesgos • Transferencia de riesgos • Plan de contingencia • Mitigación de riesgos mediante pruebas

Aceptar (ignorar) riesgos puede ser una buena idea cuando se trata de riesgos de bajo nivel.

Estos riesgos suelen tener la prioridad más baja y abordarse al final. A menudo, es posible que simplemente se nos acabe el tiempo para ocuparnos de ellos, pero esto no nos molesta especialmente. Esto se debe a que, incluso cuando se producen tales riesgos, el daño que provocan es menor o incluso insignificante. Por lo tanto, no tiene sentido perder el tiempo ocupándose de ellos cuando tenemos otras cuestiones mucho más importantes y serias que considerar (por ejemplo, riesgos de un nivel mucho más alto).

Un ejemplo de transferencia de riesgo podría ser, por ejemplo, la contratación de una póliza de seguro.

A cambio de pagar la prima del seguro, se transfiere el riesgo de sufrir en el costo del riesgo a un tercero, en este caso, al asegurador.

Se desarrollan planes de contingencia para ciertos tipos de riesgos. Estos planes se desarrollan con el fin de evitar el caos y el retraso en la respuesta a los riesgos cuando ocurre una situación indeseable. Gracias a la existencia de este tipo de planes, cuando se produce un riesgo, todo el mundo sabe exactamente qué hacer. Por lo tanto, la respuesta al riesgo es rápida y bien pensada de antemano y, por tanto, eficaz.

En el programa de estudios de nivel básico, de los métodos de respuesta al riesgo anteriores, solo se presenta en detalle la mitigación de riesgos mediante pruebas, ya que el programa de estudios trata sobre pruebas. Se pueden diseñar pruebas para comprobar si realmente existe un riesgo. Si la prueba falla (provoca una falla, por lo que muestra un defecto), el equipo es consciente del problema. En consecuencia, esto mitiga el riesgo, que ya está identificado y no es desconocido. Si la prueba pasa, el equipo gana más confianza en la calidad del sistema. La mitigación de riesgos puede reducir la probabilidad o el impacto de un riesgo. En general, las pruebas contribuyen a reducir el riesgo general.

nivel de riesgo en el sistema. Cada prueba superada se puede interpretar como una situación en la que hemos demostrado que un riesgo específico (al que está asociada la prueba) no existe porque se supera la prueba. Esto reduce el riesgo residual total por el valor del nivel de riesgo de ese riesgo particular.

Las acciones que pueden tomar los evaluadores para mitigar el riesgo del producto son las siguientes:

- Seleccionar evaluadores con el nivel correcto de experiencia, apropiado para el tipo de riesgo
- Aplicar un nivel apropiado de independencia de las pruebas
- Realizar revisiones
- Realizar análisis estáticos
- Seleccionar técnicas de diseño de pruebas y niveles de cobertura apropiados
- Priorizar las pruebas según el nivel de riesgo
- Determinar el rango apropiado de pruebas de regresión

Algunas de las actividades de mitigación abordan las pruebas preventivas tempranas aplicándolas antes de que comiencen las pruebas dinámicas (por ejemplo, revisiones). En las pruebas basadas en riesgos, las actividades de mitigación de riesgos deben realizarse en todo el SDLC.

El paradigma de desarrollo ágil de software requiere la autoorganización del equipo (consulte la Sección 1.4.2 sobre roles de prueba y la Sección 1.5.2 sobre el enfoque de “todo el equipo”) al tiempo que proporciona prácticas de mitigación de riesgos que pueden verse como un sistema holístico de mitigación de riesgos. Una de sus fortalezas es su capacidad para identificar riesgos y proporcionar buenas prácticas de mitigación. Ejemplos de riesgos reducidos por estas prácticas son [69]:

- El riesgo de insatisfacción del cliente. En un enfoque ágil, el cliente o el representante del cliente ve el producto de forma continua, lo que, con una buena ejecución del proyecto, retroalimentación frecuente y comunicación intensiva, mitiga este riesgo.
- El riesgo de no completar todas las funciones. El lanzamiento frecuente y la planificación y priorización de productos reducen este riesgo al garantizar que los incrementos relacionados con altas prioridades comerciales se entreguen primero.
- El riesgo de una estimación y planificación inadecuadas. Las actualizaciones de los productos de trabajo se rastrean diariamente para garantizar el control de la gestión y oportunidades frecuentes de corrección.
- El riesgo de no resolver los problemas rápidamente. Un equipo autoorganizado, una gestión adecuada y los informes diarios de trabajo brindan la oportunidad de informar y resolver problemas diariamente.
- El riesgo de no completar el ciclo de desarrollo. En un ciclo determinado (cuanto más corto, mejor), el enfoque ágil entrega software funcional (o, más precisamente, una parte específica del mismo) de modo que no haya problemas importantes de desarrollo.
- El riesgo de asumir demasiado trabajo y cambiar las expectativas. La gestión del trabajo pendiente del producto, la planificación de iteraciones y los lanzamientos de trabajo evitan el riesgo de cambios inadvertidos que impactan negativamente en la calidad del producto al obligar a los equipos a enfrentar y resolver los problemas tempranamente.

Monitoreo de riesgos

El monitoreo de riesgos  es una tarea de gestión de riesgos que se ocupa de las actividades relacionadas con la verificación del estado de los riesgos del producto. El monitoreo de riesgos permite a los evaluadores medir las actividades de mitigación de riesgos implementadas (acciones de mitigación) para garantizar que estén logrando los efectos previstos e identificar eventos o circunstancias que crean riesgos nuevos o mayores. El seguimiento del riesgo se suele realizar a través de informes que comparan el estado real con lo esperado. Idealmente, el monitoreo de riesgos se lleva a cabo en todo el SDLC.

Las actividades típicas de monitoreo de riesgos implican revisar e informar sobre los riesgos del producto. Hay varios beneficios del monitoreo de riesgos, por ejemplo:

- Conocer el estado exacto y actual de cada riesgo del producto
- La capacidad de informar el progreso en la reducción del riesgo residual del producto
- Centrarse en los resultados positivos de la mitigación del riesgo
- Descubrir riesgos que no han sido identificados previamente
- Captar nuevos riesgos a medida que surgen
- Factores de seguimiento que afectan los costos de gestión de riesgos

Algunos enfoques de gestión de riesgos tienen incorporados métodos de seguimiento de riesgos. Por ejemplo, en el método AMEF antes mencionado, la estimación de la probabilidad y el impacto de un riesgo se realiza dos veces: primero en la fase de evaluación del riesgo y luego, nuevamente, después de implementar acciones para mitigar ese riesgo.

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

FL-5.3.1 (K1) Recordar las 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

El objetivo principal del monitoreo de pruebas es recopilar  y compartir información para obtener información sobre las actividades de prueba y visualizar el proceso de prueba. La información monitoreada se puede recopilar de forma manual o automática, utilizando herramientas de gestión de pruebas. Más bien sugerimos un enfoque automático, utilizando el registro de pruebas y la información de la herramienta de informe de defectos, respaldado por un enfoque manual (control directo del trabajo del evaluador; esto permite una evaluación más objetiva del estado actual de las pruebas). Sin embargo, no olvide recopilar información manualmente, por ejemplo, durante las reuniones diarias de Scrum (ver Sección 2.1.1).

Los resultados obtenidos se aplican a:

- Medición del cumplimiento de los criterios de salida, por ejemplo, logro de la cobertura de riesgo asumido del producto, requisitos y criterios de aceptación.
- Evaluación del progreso del trabajo en comparación con el cronograma y el presupuesto.

Actividades realizadas como parte del control de pruebas.  son principalmente los siguientes:

- Tomar decisiones basadas en la información obtenida del monitoreo de pruebas • Volver a priorizar las pruebas cuando se materialicen los riesgos identificados (por ejemplo, no entregar el software a tiempo) • Realizar cambios en el cronograma de ejecución de pruebas • Evaluar la disponibilidad o indisponibilidad del entorno de prueba u otros recursos

5.3.1 Métricas utilizadas en las pruebas

Durante y después de un nivel de prueba determinado, se pueden (y deben) recopilar métricas que permitan estimar:

- El progreso de la implementación del cronograma y presupuesto • El nivel de calidad actual del objeto de prueba • La idoneidad del enfoque de prueba elegido • La efectividad de las actividades de prueba desde el punto de vista de lograr el objetivos

El monitoreo de pruebas recopila varias métricas para respaldar las actividades de control de pruebas. Típico Las métricas de prueba incluyen:

- Métricas del proyecto (por ejemplo, finalización de tareas, utilización de recursos, esfuerzo de prueba, porcentaje de finalización del trabajo de preparación del entorno de prueba planificado, fechas de hitos)
- Métricas de casos de prueba (p. ej., progreso de implementación de casos de prueba, progreso de preparación del entorno de prueba, número de casos de prueba ejecutados/no ejecutados, aprobados/fallidos, tiempo de ejecución de la prueba)
- Métricas de calidad del producto (por ejemplo, disponibilidad, tiempo de respuesta, tiempo medio hasta la falla) • Métricas de defectos (por ejemplo, número de defectos encontrados/reparados, prioridades de defectos, densidad de defectos (número de defectos por unidad de volumen, por ejemplo, 1000 líneas de código), frecuencia de defectos (número de defectos por unidad de tiempo), porcentaje de defectos detectados, porcentaje de pruebas de confirmación exitosas) • Métricas de riesgo (p. ej., nivel de riesgo residual, prioridad de riesgo) • Métricas de cobertura (p. ej., cobertura de requisitos, cobertura de historias de usuario, aceptación cobertura de criterios de prueba, cobertura de condiciones de prueba, cobertura de código, cobertura de riesgos) • Métricas de costos (por ejemplo, costo de pruebas, costo organizacional de calidad, costo promedio de ejecución de pruebas, costo promedio de reparación de defectos)

Tenga en cuenta que diferentes métricas tienen diferentes propósitos. Por ejemplo, desde un punto de vista puramente de gestión, un gerente estará interesado en saber cuántas ejecuciones de pruebas planificadas se realizaron o si no se superó el presupuesto planificado.

Sin embargo, desde el punto de vista de la calidad del software, serán más importantes cuestiones como, por ejemplo, el número de defectos encontrados según su grado de criticidad, el tiempo medio entre fallos, etc. Ser capaz de elegir el conjunto adecuado de métricas para medir Es un arte, ya que normalmente es imposible medirlo todo. Además, cuantos más datos haya en el

informes, menos legibles se vuelven. Debes elegir un pequeño conjunto de métricas a medir, que al mismo tiempo te permitirán responder todas las preguntas sobre los aspectos del proyecto que te interesan. La técnica Goal-Question-Metric (GQM), por ejemplo, puede ayudar a desarrollar dicho plan de medición, pero no lo presentamos aquí, ya que está más allá del alcance del programa de estudios.

5.3.2 Propósito, contenido y destinatario de los informes de prueba

Los informes de prueba se utilizan para resumir y proporcionar información sobre las actividades de prueba (por ejemplo, pruebas en un nivel determinado) tanto durante como después de su ejecución. Un informe de prueba elaborado durante la ejecución de una actividad de prueba se denomina informe de progreso de la prueba, y el informe elaborado al finalizar dicha actividad, informe de finalización de la prueba.



Ejemplos de estos informes se muestran en las Figs. 5.12, 5.13 y 5.14.

Según la norma ISO/IEC/IEEE 29119-3 [3], un informe de prueba típico debe incluir:

- Resumen de las pruebas realizadas • Descripción de lo que sucedió durante el período de prueba • Información sobre desviaciones del plan • Información sobre el estado de las pruebas y la calidad del producto, incluida información sobre cumplir con los criterios de salida (o Definición de Hecho)
- Información sobre los factores que bloquean las pruebas.

Summary Status Test Report for: New subscription system (NSS) **Vers.:** Iteration 3

Covers: Complete NSS iteration 3 results.

Progress against Test Plan: Test has been done in the iteration on the 5 user stories for this iteration. For the one high risk story 92 % statement coverage was achieved, and for the others 68 % statement coverage was achieved on average.

There are no outstanding defects of severity 1 and 2, but the showcase showed that the product has 16 defects of severity 3.

Factors blocking progress: None

Test measures: 6 new test procedures have been developed, and 2 of the other test procedures have been changed.

The testing in the iteration has taken up approx. 30 % of the time. The test took about 2½ hours.

New and changed risks: The risks for the stories have been mitigated satisfactorily. New risks are not identified yet.

Planned testing: As per test plan.

Backlog added: 16 defects (severity 3)

Fig. 5.12 Informe de progreso de pruebas en una organización ágil (según ISO/IEC/IEEE 29119-3)

- Medidas relacionadas con defectos, casos de prueba, cobertura de prueba, progreso del trabajo y recursos utilización
- Información sobre riesgos residuales •

Información sobre productos de trabajo para su reutilización

Un informe de progreso de prueba típico incluye además información sobre:

- Estado de las actividades de prueba y progreso del plan de prueba. •

Pruebas programadas para el próximo período de informe.

Los informes de prueba deben adaptarse tanto al contexto del proyecto como a las necesidades del público objetivo. Deben incluir:

- Información detallada sobre tipos de defectos y tendencias relacionadas, para audiencia técnica. •

Resumen del estado de los defectos por prioridad, presupuesto y cronograma, así como aprobados.

falló y bloqueó casos de prueba, para las partes interesadas del negocio

Los principales destinatarios del informe de progreso de las pruebas son aquellos que pueden realizar cambios en la forma en que se realizan las pruebas. Estos cambios pueden incluir agregar más

Test report for: New subscription system (NSS) Vers.: Iteration 3361

Covers: NSS final iteration result, including result of previous iterations, in preparation for a major customer delivery (for use).

Risks: The live data risk was retired by creation of a simulated database using historic live data "cleaned" by the test team and customer.

Test Results: Customer accepted this release of the product based on:

16 user stories were successful, including one added after the last status report.

100% statement coverage was achieved in technology-facing testing with the one high risk story, and for the others 72% statement coverage was achieved on average.

Team accepted the backlog on 4 defects of severity 3.

Showcase was accepted by the customer with no added findings. Showcase demo iteration features interfaced with "live" data.

Performance of the iteration features was found to be acceptable by team and customer.

New, changed, and residual risks: Security of the system could become an issue in future releases, assuming a follow work activity is received from the customer.

Notes for future work from retrospective:

Iteration team feels a new member could be needed given possible new risk since no one has knowledge in this area.

Severity 3 defects that move on to backlog should be addressed in next release to reduce technical debt.

The modified live data worked well and should be maintained.

Test automation and exploratory testing is working, but additional test design techniques should be considered, e.g. security and combinatorial testing.

Fig. 5.13 Informe resumido de pruebas en una organización ágil (según ISO/IEC/IEEE 29119-3)

Project PC-part of the UV/TRT-14 33a product**System Test Completion Report, V 1.0, 29/11/2019****Written by:** Carlo Titlesen (Test Manager)**Approved by:** Benedicte Rytter (Project Manager)**Summary of testing performed:**

- The test specification was produced; it included 600 test procedures.
- The test environment was established according to the plan.
- Test execution and recording were performed according to the plan.
- Based on the outcomes of testing, it is recommended that UAT commence.

Deviations from planned testing: The Requirement Specification was updated during the test to v5.6. This entailed rewriting of a few test cases, but this did not have an impact on the schedule.

Test completion evaluation: All test procedures were executed and passed testing, except 3. There are no remaining severity 1 or 2 defects, and all open severity 3 and 4 defects have been signed off by the business. The remaining 3 failed test cases will be rerun during UAT, once remaining severity 3 and 4 defects have been fixed. The requirements for these 3 failed tests are low risk.

Factors that blocked progress: Deployment was delayed on 5 occasions, preventing testing from starting on time. This equated to an increase in testing duration by 25 person days.

Test measures:

Three test procedures out of 605 were not passed due to defects. All test procedures that were run had passed at the end of the planned 3 weeks of testing.

During test execution, 83 defects were detected and 80 were closed.

Duration of test design and execution:

- 12,080 working hours were spent on the production of test cases and test procedures
- 10 working hours were spent on the establishment of the test environment
- 15,040 working hours were spent on test execution and defect reporting
- One hour was spent on the production of this report.

Residual risks: All the risks listed in the test plan have been eliminated, except the one with the lowest exposure, Risk # 19. This risk has been deemed acceptable by the Business Representative.

Test deliverables: All deliverables specified in the plan have been delivered to the common CM-system according to the procedure.

Reusable test assets: The test specification and the related test data and test environment requirements can be reused for maintenance testing, if and when this is needed.

Lessons learned: The test executors should have been given a general introduction to the system; it sometimes delayed them finding out how to perform a test case, but fortunately the test analyst was available throughout the test period.

Fig. 5.14 Informe resumido de prueba en una organización tradicional (según ISO/IEC/IEEE 29119)

probar recursos o incluso realizar cambios en el plan de prueba. Por lo tanto, los miembros típicos de este grupo son directores de proyectos y propietarios de productos. También puede ser útil incluir en este grupo a personas responsables de los factores que inhiben el progreso de la prueba, para que quede claro que son conscientes del problema. Finalmente, el equipo de prueba también debería

ser incluidos en este grupo, ya que esto les permitirá ver que su trabajo es apreciado y les ayudará a comprender cómo su trabajo contribuye al progreso general.

La audiencia del informe de finalización de la prueba se puede dividir en dos grupos principales: los responsables de tomar decisiones (qué hacer a continuación) y los que realizarán pruebas en el futuro utilizando la información del informe. Quienes toman las decisiones variarán según las pruebas que se informen (por ejemplo, nivel de prueba, tipo de prueba o proyecto) y pueden decidir qué pruebas deben incluirse en la siguiente iteración o si implementar un objeto de prueba dado el riesgo residual informado. El segundo grupo son aquellos que son responsables de las pruebas futuras del objeto de prueba (por ejemplo, como parte de las pruebas de mantenimiento) o aquellos que de otro modo pueden decidir sobre la reutilización de los productos a partir de las pruebas informadas (por ejemplo, datos de prueba limpiados del cliente en un proyecto separado). . Además, todas las personas que aparecen en la lista de distribución original del plan de prueba también deben recibir una copia del informe de finalización de la prueba.

5.3.3 Comunicación del estado de las pruebas

Los medios para comunicar el estado de las pruebas varían, dependiendo de las preocupaciones, expectativas y visión de la gestión de pruebas, las estrategias de prueba de la organización, los estándares regulatorios o, en el caso de equipos autoorganizados (ver Sección 1.5.2), el equipo mismo. Las opciones incluyen, en particular:

- Comunicación verbal con los miembros del equipo y otras partes interesadas.
- Paneles de control, como paneles de CI/CD, tableros de tareas y gráficos de evolución •
- Canales de comunicación electrónicos (p. ej., correos electrónicos, chats)
- Documentación en línea •

Informes de prueba formales (consulte la Sección 5.3.2)

Se pueden utilizar una o más de estas opciones. Una comunicación más formal puede ser más apropiada para equipos distribuidos, donde la comunicación directa cara a cara no siempre es posible debido a diferencias geográficas o horarias.

La Figura 5.15 muestra un ejemplo de un tablero típico para comunicar información sobre el proceso de integración y entrega continua. Desde este panel podrás obtener muy rápidamente la información básica más importante sobre el estado del proceso, por ejemplo:

- El número de lanzamientos por estado de lanzamiento (exitoso, no creado, abortado, etc.) • Las fechas y el estado de los últimos diez lanzamientos •

Información sobre las últimas diez confirmaciones •

Frecuencia de los lanzamientos

- Tiempo promedio para crear un lanzamiento como función de tiempo



Fig. 5.15 Ejemplo de un panel de CI/CD (fuente: <https://devops.com/electric-cloud-extends-continuity-delivery-platform/>)

5.4 Gestión de la configuración

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

El objetivo principal de la gestión de la configuración es garantizar y mantener la integridad del componente/sistema y el software de prueba y las interrelaciones entre ellos durante todo el ciclo de vida del proyecto y del producto. La gestión de la configuración tiene como objetivo garantizar que:

- Todos los elementos de configuración, incluidos los objetos de prueba, los elementos de prueba (partes individuales de los objetos de prueba) y otro software de prueba, han sido identificados, controlados en versión, rastreados para detectar cambios y vinculados entre sí de una manera que mantiene la trazabilidad en todas las etapas del proceso de prueba • Toda la documentación y los elementos de software identificados se mencionan explícitamente en la documentación de prueba.

Hoy en día, se utilizan herramientas apropiadas para la gestión de la configuración, pero es importante tener en cuenta que los procedimientos de gestión de la configuración junto con la infraestructura (herramientas) necesaria deben identificarse e implementarse en la etapa de planificación, ya que el proceso cubre todos los productos de trabajo que ocurren dentro de el proceso de desarrollo.

La Figura 5.16 muestra una representación gráfica de un determinado repositorio almacenado en el sistema de versiones de código git. Los vértices C0, C1, ..., C6 indican las llamadas instantáneas, es decir, versiones sucesivas del código fuente. Las flechas indican en función de qué versión se creó la siguiente versión.

Por ejemplo, la confirmación C1 se creó sobre la base de la confirmación C0 (por ejemplo, un desarrollador descargó el código C0 a su computadora local, le hizo algunos cambios y

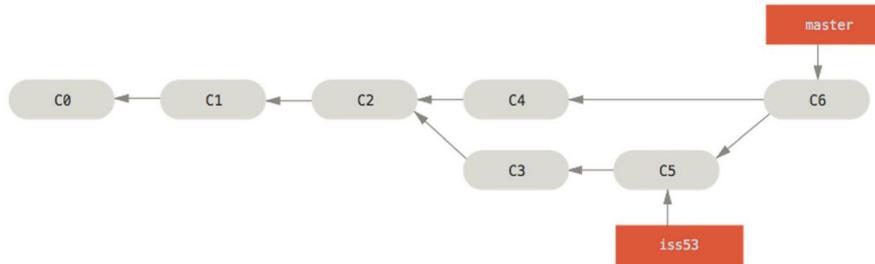


Fig. 5.16 Representación gráfica de un repositorio de código en el sistema Git (fuente: <https://git-scm.com/docs/gittutorial>)

lo comprometí en el repositorio). El repositorio distingue entre estas dos versiones del código. Es posible que otros dos desarrolladores hayan creado las confirmaciones C3 y C4 de forma independiente, basándose en la misma confirmación C2. El rectángulo "maestro" representa la rama maestra. La confirmación C5, por otro lado, representa la rama "iss53", donde se creó el código como resultado de corregir un error detectado en la versión C3 del código. La confirmación C6 se creó como resultado de una combinación de cambios realizados de forma independiente en las confirmaciones C5 (reparación de defectos) y C4 (por ejemplo, agregar una nueva característica basada en la versión C2 del código).

Al utilizar un sistema de control de versiones como git, los desarrolladores tienen control total sobre su código. Pueden rastrear, guardar y, si es necesario, recrear cualquier versión histórica del código. Si se comete un error o falla en alguna compilación, siempre es posible deshacer los cambios y volver a la versión anterior que funciona.

Hoy en día, el uso de herramientas de control de versiones de código es el estándar de facto. Sin estas herramientas, el proyecto se convertiría en un caos muy rápidamente.

Ejemplo Consideremos un ejemplo muy simplificado de la aplicación de la gestión de configuración en la práctica. Supongamos que el sistema que estamos produciendo consta de tres componentes: A, B y C. El sistema es utilizado por dos de nuestros clientes: C1 y C2.

La siguiente secuencia de eventos muestra el historial de cambios en el repositorio de código y el historial de lanzamientos de software para los clientes. Suponemos que la creación de una determinada versión de la aplicación siempre tiene en cuenta las últimas versiones de los componentes incluidos en la misma.

1. Cargando la versión 1.01 del componente A al repositorio 2. Cargando la versión 1.01 del componente B al repositorio 3. Cargando la versión 1.02 del componente A (después de la eliminación del defecto) al repositorio 4. Cargando la versión 1.01 del componente C al repositorio 5 Crear la versión 1.0 de la aplicación para su lanzamiento y enviarla a C1 6. Cargar la versión 1.02 del componente C al repositorio (agregar nuevo.

funcionalidad)

7. Crear la versión 1.01 de la aplicación para su lanzamiento y enviarla a C1 8. Cargar la versión 1.03 del componente A al repositorio (agregar nuevos
- funcionalidad)

9. Crear la versión 1.02 de la aplicación para su lanzamiento y enviarla al C2.

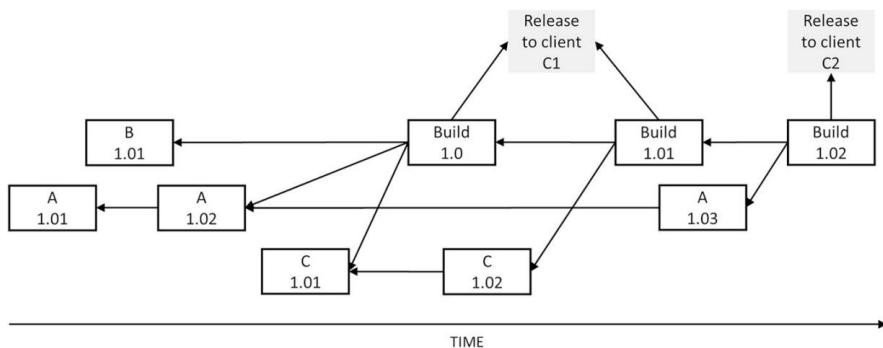


Fig. 5.17 Historia de la creación de versiones sucesivas de componentes y lanzamientos de software.

Este proceso se representa gráficamente en la figura 5.17.

Ahora supongamos que en este punto (después del paso 9), el cliente C1 informó una falla. Si no tuviéramos establecido un proceso de gestión de configuración, no sabríamos en qué versiones de los componentes A, B y C buscar defectos potenciales. Supongamos que el cliente nos envió información de que el problema se observó en la versión 1.01 del software. La herramienta de gestión de configuración, basándose en esta información y en los datos anteriores, puede reconstruir los componentes de software individuales que se incluyeron en la versión de software 1.01 entregada al cliente C1 en el paso 7. Además, la herramienta también puede reproducir versiones adecuadas de los casos de prueba utilizados para el software en la versión 1.01, las versiones adecuadas de los componentes del entorno, bases de datos, etc. que se utilizaron para crear la versión 1.01 del software. En particular, analizando la secuencia de eventos anterior, vemos que la versión de software 1.01 consta de:

- Componente A versión 1.02 •
- Componente B versión 1.01 •
- Componente C versión 1.02

Esto significa que el defecto potencial está en uno (o más) de estos tres componentes.

Tenga en cuenta que si se encuentra que el defecto está en el componente B, después de solucionarlo y crear la versión 1.02, se debe enviar una nueva versión del software (1.03) no solo al cliente C1 sino también al cliente C2, ya que la versión actual del software utilizado por C2 utiliza el componente B defectuoso en la versión 1.01.

5.5 Gestión de defectos

FL-5.5.1 (K3) Preparar un informe de defectos

Uno de los objetivos de las pruebas es encontrar defectos y, por lo tanto, todos los defectos encontrados deben registrarse. La forma en que se informa un defecto depende del contexto de prueba del componente o sistema, el nivel de prueba y el modelo SDLC elegido. Cada defecto identificado debe

ser investigado y rastreado desde el momento en que se detecta y clasifica hasta que se resuelve el problema.

La organización debe implementar un proceso de gestión de defectos, cuya formalización puede variar desde un enfoque informal hasta uno muy formal. Es posible que algunos informes describan situaciones de falsos positivos en lugar de fallas reales causadas por defectos. Los evaluadores deben intentar minimizar la cantidad de resultados falsos positivos que se informan como defectos; sin embargo, en proyectos reales, un cierto número de defectos reportados son falsos positivos.

Los objetivos principales del informe de defectos.  son:

- Comunicación:

- Proporcionar a los fabricantes información sobre la incidencia para que puedan identificar, aislar y reparar el defecto si fuera necesario.
- Proporcionar a la gestión de pruebas información actualizada sobre la calidad de las sistema bajo prueba y el progreso de las pruebas.

- Permitir la recopilación de información sobre el estado del producto bajo prueba:

- Como base para la toma de decisiones.
 - Para identificar los riesgos tempranamente
 - Como base para el análisis post-proyecto y la mejora del desarrollo.
- procedimientos

En el estándar ISO/IEC/IEEE 29119-3 [3], los informes de defectos se denominan informes de incidentes. Se trata de una nomenclatura un poco más precisa, porque no toda anomalía observada tiene por qué significar inmediatamente un problema a resolver: un evaluador puede cometer un error y considerar algo que es perfectamente correcto como un defecto que debe corregirse.

Estas situaciones generalmente se analizan durante el ciclo de vida del defecto y, por lo general, dichos informes se rechazan, con un estado de "no es un defecto" o similar.

En la figura 5.18 se muestra un ejemplo de ciclo de vida de un defecto. Desde el punto de vista del evaluador, sólo los estados Rechazado, Modificado, Aplazado y Cerrado significan que se ha completado el análisis de defectos; el estado Corregido significa que el desarrollador ha solucionado el defecto, pero solo se puede cerrar después de volver a realizar la prueba.

El informe de defectos para pruebas dinámicas debe incluir:

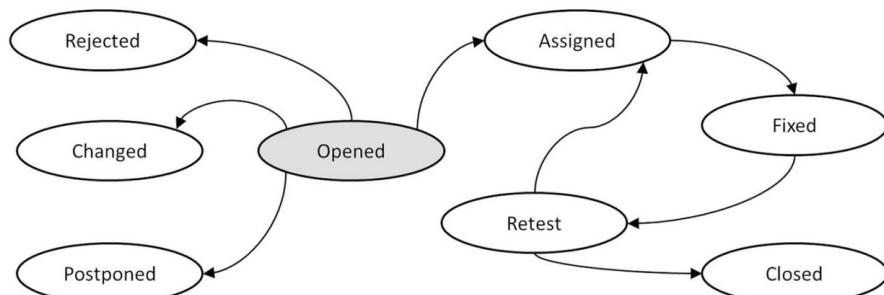


Fig. 5.18 Un ejemplo de ciclo de vida de defectos y estados de defectos

- Identificador único • Título
y un breve resumen de la anomalía reportada • Fecha del informe (la fecha en que se descubrió la anomalía) • Información sobre el autor del informe de defecto
- Identificación del elemento bajo prueba • Fase del ciclo de desarrollo de software en la que se observó anomalía • Descripción de la anomalía para permitir su reproducción y eliminación, incluyendo cualquier tipo de registros, volcados de bases de datos, capturas de pantalla o grabaciones • Resultados reales y esperados • Prioridad de eliminación de defectos (si se conoce) • Estado (p. ej., abierto, aplazado, duplicado, reabierto, etc.) • Descripción de la no conformidad para ayudar a determinar la causa del problema • Urgencia de la solución (si se conoce) • Identificación de un elemento de configuración del sistema o software • Conclusiones y recomendaciones • Historial de cambios • Referencias a otros elementos, incluido el caso de prueba a través del cual se resolvió el problema fue revelado

En la figura 5.19 se muestra un ejemplo de informe de defecto .

Las partes interesadas en los informes de defectos incluyen evaluadores, desarrolladores, gerentes de pruebas y gerentes de proyectos. En muchas organizaciones, el cliente también obtiene acceso al sistema de gestión de defectos, especialmente en el período inmediatamente posterior a la liberación. Sin embargo, se debe tener en cuenta que el cliente típico no está familiarizado con las reglas de notificación de defectos. Además, es típico que desde el punto de vista del cliente, casi todos los defectos sean críticos, porque impiden el trabajo del cliente.

Preguntas de muestra

Pregunta 5.1

(FL-5.1.1, K2)

¿Cuál de los siguientes NO es parte del plan de prueba?

- A. Estrategia de prueba.
- B. Restricciones presupuestarias.
- C. Alcance de las pruebas.
- D. Registro de riesgos.

Elija una respuesta.

Pregunta 5.2

(FL-5.1.2, K2)

¿Cuál de las siguientes acciones realiza el evaluador durante la planificación del lanzamiento?

- A. Realizar análisis de riesgos detallados para historias de usuarios.
- B. Identificar aspectos no funcionales del sistema a probar.

Incident Registration Form			
Number	278		
Short Title	Information truncated		
Software product	Project PC-part of the UV/TIT-14 33a product		
Version (n.m)	5.2		
Status = Created			
Registration created by	Heather Small	Date & time	14 th May
Anomaly observed by	Heather Small	Date & time	14 th May
Comprehensive description	<i>The text in field "Summary" is truncated after 54 characters; it should be able to show 75 characters.</i>		
Observed during	Walk-through / Review / Inspection / Code & Build / Test / Use		
Observed in	Requirement / Design / Implementation / Test / Operation		
Symptom	Oper. system crash / Program hang up / Program crash / Input / Output / Total product failure / System error / Other:		
User impact	High / Medium / Low		
User urgency	Urgent / High / Medium / Low / None		

Fig. 5.19 Ejemplo de informe de defectos según ISO/IEC/IEEE 29119-3

C. Estimar el esfuerzo de prueba para nuevas funciones planificadas en una iteración determinada.

D. Definición de historias de usuario y sus criterios de aceptación.

Elija una respuesta.

Pregunta 5.3

(FL-5.1.3, K2)

Considere los siguientes criterios de entrada y salida:

i. Disponibilidad de probadores. ii.

Sin defectos críticos abiertos. III.

Cobertura del 70% del estado de cuenta lograda en las pruebas de componentes.

IV. Todas las pruebas de humo realizadas antes de la prueba del sistema han pasado.

¿Cuáles de los anteriores son criterios de entrada y cuáles son criterios de salida para las pruebas?

A. (i), (iii) y (iv) son los criterios de entrada; (ii) es el criterio de salida.

B. (ii) y (iii) son los criterios de entrada; (i) y (iv) son los criterios de salida.

- C. (i) y (ii) son los criterios de entrada; (iii) y (iv) son los criterios de salida.
 D. (i) y (iv) son los criterios de entrada; (ii) y (iii) son los criterios de salida.

Elija una respuesta.

Pregunta 5.4

(FL-5.1.4, K3)

El equipo utiliza el siguiente modelo de extrapolación para la estimación del esfuerzo de prueba:

$$E_{n\ddot{o} p} = \frac{E_{n\ddot{o} p - 1} + E_{n\ddot{o} p - 2} + E_{n\ddot{o} p - 3}}{3},$$

donde $E(n)$ es el esfuerzo en la enésima iteración. El esfuerzo en la enésima iteración es, por tanto, el promedio del esfuerzo de las últimas tres iteraciones.

En las primeras tres iteraciones, el esfuerzo real fue de 12, 15 y 18 días-persona, respectivamente. El equipo acaba de completar la tercera iteración y quiere utilizar el modelo anterior para estimar el esfuerzo de prueba necesario en la QUINTA iteración. ¿Cuál debería ser la estimación hecha por el equipo?

- A. 24 días-persona.
 B. 16 días-persona.
 C. 15 días-persona.
 D. 21 días-persona.

Elija una respuesta.

Pregunta 5.5

(FL-5.1.5, K3)

Quiere priorizar los casos de prueba para lograr un orden de ejecución de prueba óptimo. Utiliza un método de priorización basado en cobertura adicional. La cobertura de funciones se utiliza como métrica de cobertura. Hay siete funciones en el sistema, denominadas A, B, C, D, E, F y G. La tabla 5.4 muestra qué funciones cubre cada caso de prueba: ¿Qué caso de prueba se ejecutará TERCERO en orden?

- A.TC5.
 B. TC2.
 C.TC4.
 D.TC3.

Elija una respuesta.

Tabla 5.4 Cobertura de funciones por casos de prueba

Caso de prueba	Funciones cubiertas
TC1	A, B, C, F
TC2	D
TC3	A, F, G
TC4	mi
TC5	re, g

Pregunta 5.6

(FL-5.1.6, K1)

¿Qué describe el modelo piramidal de prueba?

- A. Esfuerzo del equipo dedicado a las pruebas, que aumenta de iteración en iteración.
- B. Una lista de tareas del proyecto ordenadas por número descendente de actividades de prueba requeridas para cada tarea.
- C. La granularidad de las pruebas en cada nivel de prueba.
- D. Esfuerzo de prueba en cada nivel de prueba.

Elija una respuesta.

Pregunta 5.7

(FL-5.1.7, K2)

¿En qué cuadrante de prueba se encuentran las pruebas de componentes?

- R. En el cuadrante orientado a la tecnología y de apoyo al equipo que incluye pruebas automatizadas y ser parte del proceso de integración continua.
- B. En el cuadrante orientado a los negocios y de apoyo al equipo que incluye la aceptación. prueba de criterios.
- C. En el sector empresarial, de crítica del producto, que incluye pruebas centradas en los usuarios. necesidades.
- D. En el cuadrante orientado a la tecnología y que critica el producto, que incluye la automatización. pruebas no funcionales.

Elija una respuesta.

Pregunta 5.8

(FL-5.2.1, K1)

La probabilidad de un riesgo de rendimiento del sistema se estimó como "muy alta".

¿Qué se puede decir sobre el impacto de este riesgo?

- R. No sabemos nada sobre el impacto; el impacto y la probabilidad son independientes.
- B. El impacto también es muy alto; Los riesgos de alta probabilidad también tienen un alto impacto.
- C. El impacto es bajo porque el impacto es inversamente proporcional a la probabilidad.
- D. Hasta que este riesgo ocurra, no podemos evaluar su impacto.

Elija una respuesta.

Pregunta 5.9

(FL-5.2.2, K2)

¿Cuál de los siguientes es un ejemplo de una consecuencia de un riesgo de proyecto?

- A. Muerte del usuario debido a una falla del software.
- B. No completar todas las tareas previstas para completarse en una iteración determinada.
- C. Costos de mantenimiento de software muy elevados.
- D. Insatisfacción del cliente debido a una interfaz de usuario inconveniente.

Elija una respuesta.

Pregunta 5.10

(FL-5.2.3, K2)

El evaluador está trabajando en un proyecto que prepara una nueva versión de una aplicación de banca móvil desde casa. Durante el análisis de riesgos, el equipo identificó los dos riesgos siguientes:

- Interfaz demasiado complicada para definir las transferencias, especialmente para las personas mayores.
- Mal funcionamiento del mecanismo de transferencia: las transferencias se ejecutan tarde cuando el pago cae en sábado o domingo.

¿Cuáles son las acciones de mitigación de riesgos más razonables que un evaluador debería proponer para estos dos riesgos?

- A. Revisión técnica de la interfaz de transferencia y cobertura de sucursales para la transferencia mecanismo.
- B. Pruebas de componentes para interfaz de transferencia y pruebas de aceptación para transferencia mecanismo.
- C. Pruebas beta para la interfaz de transferencia y pruebas de usabilidad para el mecanismo de transferencia
- D. Pruebas de caja blanca para la interfaz de transferencia y pruebas no funcionales para la transferencia mecanismo.

Elija una respuesta.

Pregunta 5.11

(FL-5.2.4, K2)

Después de realizar pruebas del sistema y entregar el software al cliente, el productor de software contrató un seguro con una compañía de seguros. El productor hizo esto en caso de que un mal funcionamiento del software provocara que sus usuarios perdieran la salud.

¿A qué tipo de acción de mitigación nos enfrentamos aquí?

- A. Planes de contingencia.
- B. Mitigación de riesgos mediante pruebas.
- C. Transferencia de riesgos.
- D. Aceptación del riesgo.

Pregunta 5.12

(FL-5.3.1, K1)

¿Cuál de las siguientes NO es una métrica utilizada para realizar pruebas?

- A. Nivel de riesgo residual.
- B. Cobertura de requisitos por código fuente.
- C. Número de defectos críticos encontrados.
- D. Progreso de la implementación del entorno de prueba.

Elija una respuesta.

Pregunta 5.13

(FL-5.3.2, K2)

¿Cuál de los siguientes NO se incluirá normalmente en un informe de finalización de prueba?

- R. Los riesgos restantes (no mitigados) son los riesgos R-001-12 y R-002-03.
- B. Desviaciones del plan de pruebas: las pruebas de integración se retrasaron 5 días.
- C. Número de defectos críticos abiertos: 0.
- D. Pruebas programadas para el próximo período de informe: pruebas de componentes del M3 componente.

Elija una respuesta.

Pregunta 5.14

(FL-5.3.3, K2)

¿Cuál de las siguientes es la MEJOR forma de comunicar el estado de las pruebas?

- A. Informes de progreso de pruebas e informes de finalización de pruebas, porque son los más forma formal de comunicación.
- B. La mejor forma de comunicación dependerá de varios factores.
- C. Correos electrónicos, porque permiten un rápido intercambio de información.
- D. Comunicación verbal, cara a cara, porque es la forma más eficaz de comunicación entre personas.

Elija una respuesta.

Pregunta 5.15

(FL-5.4.1, K2)

El equipo recibió información del cliente sobre una falla del software. Según el número de versión del software, el equipo pudo reconstruir todas las versiones de componentes y software de prueba que se utilizaron para generar la versión del software para este cliente.

Esto hizo posible localizar y corregir el defecto más rápidamente, así como analizar qué otras versiones de la versión de software deberían parchearse en relación con el defecto.

¿Qué proceso permitió al equipo ejecutar el escenario anterior?

- A. Gestión de la configuración.
- B. Análisis de impacto.
- C. Entrega continua.
- D. Retrospectiva.

Elija una respuesta.

Pregunta 5.16

(FL-5.5.1, K3)

Mientras prueba una nueva aplicación, un evaluador encuentra un mal funcionamiento del componente de inicio de sesión. Según la documentación, se supone que la contraseña debe tener al menos diez caracteres, incluido un mínimo de una letra mayúscula, una minúscula y un dígito.

El probador prepara un informe de defectos que contiene la siguiente información:

- Título: Inicio de sesión incorrecto.
- Breve resumen: En ocasiones, el sistema permite contraseñas de 6, 7 y 9 personajes.
- Versión del producto: compilación 11.02.2020.
- Nivel de riesgo: Alto.
- Prioridad: Normal.

¿Qué información VALIOSA falta en el informe de defectos anterior?

- A. Pasos para reproducir el defecto.
- B. Datos que identifican el producto que se está ensayando.
- C. Estado del defecto.
- D. Ideas para mejorar el caso de prueba.

Elija una respuesta.

Ejercicios para el Capítulo 5

Ejercicio 5.1

(FL-5.1.4, K3)

Un grupo de tres expertos estima el esfuerzo de prueba para la tarea de “realizar prueba del sistema” utilizando un método que es una combinación de póquer de planificación y estimación de tres puntos. El procedimiento es el siguiente:

- Los expertos determinan los valores pesimista, más probable y optimista para la estimación de tres puntos. Cada uno de ellos se determina realizando un póquer de planificación. La sesión de poker se lleva a cabo hasta que al menos dos expertos den el mismo valor, en el cuyo caso el póker termina y el resultado es el valor determinado por la mayoría de expertos.
- Los expertos utilizan la estimación de tres puntos con los parámetros pesimista, más probable y optimista determinados en el paso anterior; esta estimación es el resultado final. La desviación estándar también se calcula para describir el error de estimación.

Los resultados de la sesión de póquer se muestran en la Tabla 5.5. Todos los valores se expresan en días-persona.

Tabla 5.5 Resultados de la planificación de una sesión de póquer

Valor estimado	Iteraciones de planificación del póquer
Optimista (un)	Resultados de la iteración 1: 3, 5, 8 Resultados de la iteración 2: 3, 3, 5
Lo más probable (m)	Resultados de la iteración 1: 5, 5, 3
pesimista (b)	Resultados de la iteración 1: 13, 8, 21 Resultados de la iteración 2: 8, 13, 40 Resultados de la iteración 3: 13, 13, 13

Tabla 5.6 Datos históricos de un proyecto

Fase	Número de personas involucradas	Duración (Días)
Diseño	4	5
Implementación	10	18
Pruebas	4	10

¿Cuáles son la estimación del esfuerzo final y el error de estimación para la tarea bajo análisis?

Ejercicio 5.2

(FL-5.1.4, K3)

Nota: esta tarea es bastante difícil y requiere algunas habilidades analíticas y una buena comprensión de las métricas del producto, como el esfuerzo. Sin embargo, damos este ejercicio en El propósito es mostrar qué tipo de problemas pueden enfrentar los gerentes en la práctica. A menudo, Estos son problemas no triviales, como el siguiente.

La Tabla 5.6 muestra datos históricos de un proyecto completado en el que el diseño, Las fases de implementación y prueba se realizaron secuencialmente, una tras otra:

Desea utilizar la técnica de estimación del esfuerzo basada en proporciones para estimar el esfuerzo. necesario para ejecutar un proyecto nuevo y similar. Se sabe que el nuevo proyecto tendrá cuatro diseñadores, seis desarrolladores y dos evaluadores. El contrato exige que el proyecto sea completado en 66 días.

¿Cuántos días debemos planificar para el diseño, cuántos para la implementación y ¿Cuántos para probar en un nuevo proyecto?

Pista. Calcular el esfuerzo de cada una de las tres fases del proyecto anterior. (en días-persona). Luego calcule el esfuerzo total requerido para el nuevo proyecto. Finalmente, utilizar la técnica basada en ratios para distribuir el esfuerzo en el nuevo proyecto entre los etapas.

Ejercicio 5.3

(FL-5.1.5, K3)

Estás probando una aplicación que soporta el funcionamiento de la línea de ayuda en una compañía de seguros. compañía y permite encontrar la póliza de un cliente según su número de cédula.

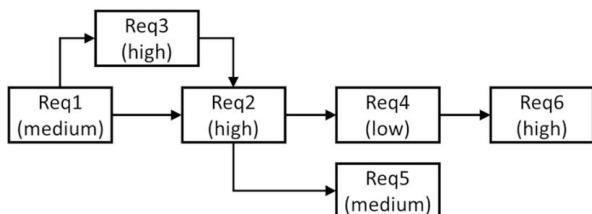
Los casos de prueba se describen en la Tabla 5.7. Prioridad 1 significa la prioridad más alta y 4: la prioridad más baja.

Defina el orden correcto en el que se deben ejecutar los casos de prueba.

Tabla 5.7 Casos de prueba para las pruebas del sistema de línea de ayuda con prioridades y dependencias

Caso de prueba	Condición de prueba cubierta	Prioridad	Dependencia lógica
TC001	Buscar por número de identificación	1	002, 003
TC002	Introduciendo datos personales	3	
TC003	modificación del número de identificación	2	002
TC004	Eliminación de datos personales	4	002

Fig. 5.20 Prioridades y relaciones entre requisitos



Ejercicio 5.4

(FL-5.1.5, K3)

El equipo quiere priorizar las pruebas de acuerdo con la priorización de los requisitos presentados al equipo por el cliente. El cliente especifica la prioridad de cada uno de los seis requisitos Req1–Req6 como baja, media o alta. Además de las prioridades, existe un cierto orden lógico entre los requisitos. Algunos de ellos pueden implementarse (y probarse) sólo después de que otros se hayan implementado.

Las prioridades y dependencias entre requisitos se muestran en la Fig. 5.20. Una flecha que va del requisito A al requisito B significa que el requisito B puede implementarse y probarse sólo después de que se completen la implementación y las pruebas del requisito A.

Determine el orden final en el que se deben probar los requisitos.

Ejercicio 5.5

(FL-5.5.1, K3)

Estás probando una aplicación para una tienda de comercio electrónico. Los requisitos para considerar el descuento para un cliente se detallan a continuación:

- Si un cliente ha realizado una compra inferior a \$50 y no tiene una tarjeta de fidelidad, no recibe descuento. • Si un cliente ha realizado una compra de menos de \$50 y tiene una tarjeta de fidelidad, recibirá un 5% de descuento.
- Si un cliente ha realizado una compra de al menos \$50 y menos de \$500 y no tiene tarjeta de fidelización, recibirá un 5% de descuento. • Si un cliente ha realizado una compra de al menos \$50 y menos de \$500 y tiene una tarjeta de fidelidad, recibirán un 10% de descuento.
- Si un cliente ha realizado una compra de al menos \$500 y no tiene una tarjeta de fidelización, recibirá un 10% de descuento. • Si un cliente ha realizado una compra de al menos \$500 y tiene una tarjeta de fidelización, recibirá un 15% de descuento.

Tabla 5.8 Tabla de resultados de casos de prueba (Ejercicio 5.5)

Prueba caso	Monto de la compra [\$]	¿Tiene una tarjeta?	Valor de descuento [\$]	A pagar [\$]	Prueba resultado
TC 001	25	Sí	1	24	Fallar
TC 002	50	Sí	0	50	Fallar
TC 003	50	No	2.50	48	Fallar
TC 004 500		Sí	50	450	Aprobar
TC 005 600		Sí	50	550	Fallar

Ha ejecutado varios casos de prueba. Sus resultados se muestran en la Tabla 5.8.

Elaborar un informe de defectos para TC 003.

Capítulo 6 Herramientas de prueba



Palabra clave

Automatización de pruebas El uso de software para realizar o soportar actividades de prueba.

6.1 Soporte de herramientas para pruebas

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

Un dicho muy conocido dice que la automatización reemplaza lo que funciona con algo que casi funciona, pero que es más rápido y más barato [70]. Este dicho capta con bastante precisión la realidad de la automatización de pruebas.

Ya sabemos que las tareas básicas del testing son:

- Análisis y evaluación del producto •

Determinación de qué medidas se utilizarán en las pruebas •

Análisis de pruebas

- Diseño de

pruebas • Implementación

de pruebas • Ejecución

de pruebas • Análisis del resultado

de las pruebas • Gestión del entorno de pruebas

No todas estas actividades pueden automatizarse completamente. De hecho, las pruebas automatizadas significan pruebas asistidas por computadora y, en la práctica, cuando se habla de automatización de pruebas uno generalmente significa automatizar su ejecución, es decir, implementar y ejecutar scripts de prueba automatizados.

Sin embargo, es importante señalar que la automatización también puede incluir otras áreas de pruebas. Por ejemplo, en un enfoque de pruebas basadas en modelos (MBT), la automatización puede incluir el diseño de pruebas y la determinación del resultado esperado basándose en el análisis del modelo proporcionado por el evaluador.

Las herramientas actuales que respaldan las pruebas respaldan varias actividades de prueba, como por ejemplo:

- Diseño, implementación y ejecución de pruebas • Preparación de datos de prueba • Gestión de pruebas, gestión de defectos y gestión de requisitos • Monitoreo e informes de ejecución de pruebas

El uso de herramientas de prueba puede tener varios propósitos:

- Automatizar tareas repetitivas o tareas que requieren muchos recursos o un esfuerzo significativo para realizarlas manualmente (por ejemplo, pruebas de regresión); esto nos permite aumentar la eficiencia de las pruebas. • Respaldar las actividades manuales y aumentar la eficiencia de las actividades de prueba, lo que aumenta la confiabilidad de las pruebas. • Aumentar la consistencia de las pruebas y la reproducibilidad de los defectos, así aumentando la calidad de las pruebas.
- Automatizar actividades que no se pueden realizar o que son muy difíciles de realizar manualmente (como las pruebas de rendimiento).

Efecto sonda

Algunos tipos de herramientas de prueba pueden ser invasivas; su uso puede afectar el resultado real de la prueba. Este fenómeno se llama efecto sonda. Por ejemplo, cuando utilizamos una herramienta de prueba de rendimiento, el rendimiento del software bajo prueba puede ser ligeramente peor, debido a la introducción de instrucciones de código adicionales en el software por parte de la herramienta de prueba de rendimiento.

Las herramientas de prueba apoyan y facilitan muchas actividades de prueba. Los ejemplos incluyen, entre otros, los siguientes grupos de herramientas:

- Herramientas de gestión: aumentan la eficiencia del proceso de prueba al facilitar la gestión del ciclo de vida de las aplicaciones, la base de pruebas para la trazabilidad de los requisitos, el seguimiento de las pruebas, la gestión de defectos y la gestión de la configuración; Ofrecen herramientas de trabajo en equipo (p. ej., tablero scrum) y proporcionan informes automatizados.
 - Herramientas de pruebas estáticas: ayudan al evaluador a realizar revisiones (principalmente en la planificación de revisiones, respaldar la trazabilidad, facilitar la comunicación, colaborar en las revisiones y mantener un repositorio para recopilar e informar métricas) y análisis estático.
 - Herramientas de prueba dinámicas: apoyan al evaluador en la realización de análisis dinámicos, creación de perfiles de código, monitoreo del uso de la memoria durante la ejecución del programa, etc.
 - Herramientas de diseño e implementación de pruebas: facilitan la generación de casos de prueba, datos de prueba y procedimientos de prueba, y soportan modelos. enfoque de prueba basado en, y proporcionar marcos para el desarrollo impulsado por el comportamiento (BDD) o el desarrollo impulsado por pruebas de aceptación (ATDD). •
- Herramientas de ejecución de pruebas y medición de cobertura: facilitan la ejecución automatizada de pruebas (scripts de prueba) y la medición automatizada de la cobertura lograda por estas pruebas, así como también permiten la medición de los resultados de prueba esperados y reales; Estas herramientas incluyen herramientas para marcos y pruebas automatizadas de GUI/API.

para pruebas de componentes y para ejecutar pruebas en enfoques como BDD o ATDD.

- Herramientas de prueba no funcionales: permiten al evaluador realizar pruebas no funcionales que son difíciles o imposibles de realizar manualmente (por ejemplo, generar carga para pruebas de rendimiento, escaneo en busca de vulnerabilidades de seguridad, pruebas de usabilidad de interfaces y páginas web, etc.).
- Herramientas DevOps: respaldan la canalización de implementación, el seguimiento del flujo de trabajo, la creación de procesos de automatización, la implementación automatizada de software, la integración continua y la entrega continua.
- Herramientas de colaboración: facilitan la comunicación. • Herramientas para soportar la escalabilidad y estandarización de implementaciones (por ejemplo, máquinas virtuales, herramientas de contenedорización, etc.).

6.2 Beneficios y riesgos de la automatización de pruebas

FL-6.2.1 (K2) Recordar los beneficios y riesgos de la automatización de pruebas

El simple hecho de poseer y utilizar una herramienta todavía no garantiza el éxito. Un conocido dicho de Grady Booch dice: "un tonto con una herramienta sigue siendo un tonto". Lograr beneficios reales y duraderos al implementar una nueva herramienta en una organización siempre requiere un esfuerzo adicional. Una herramienta es sólo una herramienta y no hará todo el trabajo para el evaluador. Es como esperar que el martillo que compramos pueda clavar clavos por sí solo.

Los beneficios potenciales del uso de las herramientas incluyen:

- Ahorrar tiempo al reducir el trabajo manual repetitivo (por ejemplo, ejecutar pruebas de regresión, volver a ingresar los mismos datos de prueba, comparar los resultados de las pruebas reales y esperados, comparar el código con los estándares de codificación).
- Una mayor coherencia y repetibilidad evita errores humanos simples (por ejemplo, las pruebas se derivan consistentemente de los requisitos, los datos de las pruebas se crean de manera sistemática y las pruebas se ejecutan mediante la herramienta en el mismo orden, de la misma manera y con la misma frecuencia). • Evaluación más objetiva (por ejemplo, medición de cobertura consistente) y la capacidad de calcular métricas que son demasiado complejas para que las calculen los humanos. • Acceso más fácil a la información de las pruebas (p. ej., estadísticas, gráficos y datos agregados sobre el progreso de las pruebas, los defectos y el tiempo de ejecución) para respaldar la gestión y los informes de las pruebas.
- Reducción del tiempo de ejecución de las pruebas, lo que proporciona una detección más temprana de defectos y una retroalimentación más rápida. y un tiempo de comercialización más rápido.
- Más tiempo para que los evaluadores diseñen pruebas nuevas, más sólidas y más efectivas.

También existen ciertos riesgos asociados con el uso de herramientas de prueba:

- Expectativas poco realistas sobre los beneficios de la herramienta (incluida la funcionalidad y facilidad de uso).

- Estimación inexacta o errónea del tiempo, el costo y el esfuerzo necesarios para implementar la herramienta, mantener los scripts de prueba y cambiar las pruebas manuales existentes.
- proceso.
- Usar una herramienta de prueba cuando las pruebas manuales sean más apropiadas (por ejemplo, pruebas de usabilidad de la interfaz sujetas a evaluación humana).
- Confiar en la herramienta cuando se necesita pensamiento crítico humano.
- Dependencia del proveedor de la herramienta, que puede cerrar, retirar la herramienta, venderla a otro proveedor o brindar soporte deficiente (p. ej., respuestas a consultas, actualizaciones y corrección de errores).
- Al utilizar la herramienta de código abierto, este proyecto de herramienta puede abandonarse, lo que significa que no habrá más actualizaciones disponibles o que sus componentes internos necesitarán actualizarse con bastante frecuencia como parte del desarrollo posterior de la herramienta.
- La plataforma y la herramienta no son compatibles entre sí.
- El incumplimiento de requisitos reglamentarios y/o normas de seguridad.

Preguntas de muestra

Pregunta 6.1

(FL-6.1.1, K2)

¿Cuál de las siguientes actividades debería estar respaldada por una herramienta de gestión de pruebas?

- A. Diseño de prueba.
- B. Gestión de requisitos.
- C. Ejecución de la prueba.
- D. Informe de defectos.

Elija una respuesta.

Pregunta 6.2

(FL-6.1.2, K1)

¿Cuáles DOS son los beneficios asociados con el uso de herramientas de prueba?

- A. Dependencia de la herramienta.
- B. Dependencia del proveedor de herramientas.
- C. Incrementar la repetibilidad de las pruebas.
- D. Evaluación de cobertura mecánica.
- E. Costo de mantenimiento del software de prueba superior al estimado.

Elija dos respuestas.

Parte III

Respuestas a preguntas y ejercicios

Respuestas a preguntas de muestra



Respuestas a las preguntas del cap. 1

Pregunta 1.1

(FL-1.1.1, K1)

Respuesta correcta: B

La respuesta A es incorrecta. Este es uno de los objetivos de la prueba según el plan de estudios.

La respuesta B es correcta. El objetivo de las pruebas de aceptación es confirmar (validación) que el sistema funcione como se esperaba, en lugar de buscar fallas (verificación).

La respuesta C es incorrecta. Este es uno de los objetivos de la prueba según el plan de estudios.

La respuesta D es incorrecta. Este es uno de los objetivos de la prueba según el plan de estudios.

Pregunta 1.2

(FL-1.1.2, K2)

Respuesta correcta: A

Las actividades de prueba son provocar fallas (ii) y realizar nuevas pruebas, o pruebas de confirmación (iv). La depuración es el proceso de encontrar, analizar y corregir las causas de fallas en un componente o sistema, por lo que encontrar defectos en el código (i) y analizar los defectos encontrados (iii) son las actividades de depuración. Por tanto, la respuesta A es correcta.

Pregunta 1.3

(FL-1.2.1, K2)

Respuesta correcta: B

La respuesta A es incorrecta. Tal comentario puede crear conflicto en el equipo, y esto no debe permitirse, porque el conflicto amenaza el logro de los objetivos del proyecto.

La respuesta B es correcta. Informar de esta deficiencia o incluir la cuestión del momento de convertir un objeto en oro como uno de los criterios de aceptación de la historia ilustra bien la

Contribución del testing en el proceso de desarrollo. Esto se debe a que minimiza el riesgo de que los desarrolladores no tengan en cuenta este requisito.

La respuesta C es incorrecta. No hay justificación para la reacción inmediata del propietario del producto. Además, las pruebas no pueden obligar a nadie a hacer nada. Sólo puede informar sobre problemas.

La respuesta D es incorrecta. La eficiencia del rendimiento es una característica de calidad del software no funcional. El problema encontrado es un defecto funcional, no no funcional.

Además, no mejoramos el rendimiento del juego eliminando este defecto.

Pregunta 1.4

(FL-1.2.2, K1)

Respuesta correcta: A

El aseguramiento de la calidad se enfoca en prevenir la introducción de defectos mediante el establecimiento, implementación y control de procesos apropiados (i), mientras que las pruebas se enfocan en evaluar el software y los productos relacionados para determinar si cumplen con los requisitos específicos (iii). El aseguramiento de la calidad no controla la calidad del producto que se está desarrollando (ii). Las pruebas no se centran en eliminar defectos del software (iv). Entonces las oraciones correctas son (i) y (iii). Por tanto, la respuesta correcta es A.

Pregunta 1.5

(FL-1.2.3, K1)

Respuesta correcta: B

La respuesta A es incorrecta. Esta es la definición de error según el glosario ISTQB®.

La respuesta B es correcta. Según el glosario ISTQB®, un defecto es una imperfección o deficiencia en un producto de trabajo donde no cumple con sus requisitos o especificaciones.

La respuesta C es incorrecta. Esta es la definición de falla según el glosario ISTQB®.

La respuesta D es incorrecta. Un caso de prueba es un producto de trabajo, no un defecto en un producto de trabajo.

Pregunta 1.6

(FL-1.3.1, K2)

Respuesta correcta:D

La respuesta A es incorrecta. Este principio establece que las pruebas tempranas y la detección de defectos nos permiten corregir defectos en las primeras etapas del SDLC, reduciendo o eliminando cambios costosos que tendrían que realizarse si los defectos se descubrieran más tarde, como después del lanzamiento.

La respuesta B es incorrecta. Este principio dice que las pruebas se realizan de manera diferente en diferentes contextos empresariales.

La respuesta C es incorrecta. Este principio dice que es necesario modificar las pruebas y los datos de prueba existentes y escribir nuevas pruebas para que el conjunto de pruebas esté constantemente listo para detectar nuevos defectos.

La respuesta D es correcta. Este principio dice exactamente que "una pequeña cantidad de componentes del sistema generalmente contienen la mayoría de los defectos descubiertos o son responsables de la mayoría de las fallas operativas. Este fenómeno es una ilustración del principio de Pareto".

Pregunta 1.7

(FL-1.4.1, K2)

Respuesta correcta: C.

Según el plan de estudios, la comprobabilidad de la base del examen se verifica durante el examen. análisis. Por tanto, la respuesta correcta es C.

Pregunta 1.8

(FL-1.4.2, K2)

Respuesta correcta: C.

La respuesta A es incorrecta. El presupuesto tiene un impacto significativo en el proceso de prueba.

La respuesta B es incorrecta. Los estándares y normas tienen un impacto significativo en la prueba. proceso, especialmente en proyectos auditados o proyectos de sistemas críticos.

La respuesta C es correcta. La cantidad de probadores certificados empleados en una organización no tiene un impacto significativo en el proceso de prueba.

La respuesta D es incorrecta. El conocimiento de los evaluadores sobre el ámbito empresarial tiene un impacto significativo en el proceso de prueba, ya que permite una comunicación más efectiva con el cliente y contribuye a la eficiencia de las pruebas.

Pregunta 1.9

(FL-1.4.3, K2)

Respuesta correcta:D

La respuesta A es incorrecta. Un informe de progreso de la prueba es un producto de trabajo típico del monitoreo y control de la prueba.

La respuesta B es incorrecta. La información sobre el nivel de riesgo actual de un producto es la información típica reportada como parte del monitoreo de pruebas.

La respuesta C es incorrecta. Si las decisiones tomadas dentro del control de pruebas están documentadas, es en esta fase.

La respuesta D es correcta. El informe de finalización de la prueba es un producto del trabajo elaborado durante la fase de finalización de la prueba.

Pregunta 1.10

(FL-1.4.4, K2)

Respuesta correcta: A

La respuesta A es correcta. Si se cuantifican los riesgos, los resultados de las pruebas se pueden rastrear hasta los casos de prueba y estos hasta los riesgos que cubren. Si todos los casos de prueba relacionados con un riesgo determinado pasan, se puede considerar que el nivel de riesgo restante en el producto ha disminuido en el valor de ese riesgo.

La respuesta B es incorrecta. Definir un nivel aceptable de cobertura de código es un ejemplo de cómo establecer un criterio de salida. El mecanismo de trazabilidad no tiene nada que ver con este proceso.

La respuesta C es incorrecta. La trazabilidad no ayudará a determinar el resultado esperado de un caso de prueba, porque la trazabilidad no tiene la propiedad de un oráculo de prueba.

La respuesta D es incorrecta. Es posible obtener este tipo de datos de prueba utilizando una técnica de prueba adecuada en lugar de un mecanismo de trazabilidad.

Pregunta 1.11 (FL

1.4.5, K2)

Respuesta correcta:D

Las actividades de gestión de pruebas incluyen principalmente tareas realizadas en la planificación, seguimiento, control y finalización de las pruebas. En particular, esto incluye coordinar la implementación de la estrategia de prueba y el plan de prueba (i), crear un informe de finalización de la prueba (iii) y decidir sobre la implementación del entorno de prueba (v). Las actividades de prueba incluyen tareas que ocurren principalmente durante las fases de análisis de prueba, diseño de prueba, implementación de prueba y ejecución de prueba. Por lo tanto, el evaluador es responsable de definir las condiciones de prueba (ii), la automatización de pruebas (iv) y verificar los entornos de prueba (vi). Por tanto, la respuesta correcta es D.

Pregunta 1.12

(FL-1.5.1, K2)

Respuesta correcta: C.

Según el programa de estudios, las habilidades genéricas típicas de un buen evaluador incluyen, en particular, pensamiento analítico (A), conocimiento del dominio (B) y habilidades de comunicación (D). La habilidad de programación (C) no es un atributo crítico de un evaluador. No es necesariamente necesario para una buena ejecución de la prueba (por ejemplo, para pruebas manuales o exploratorias).

Pregunta 1.13

(FL-1.5.2, K1)

Respuestas correctas: B, E

La respuesta A es incorrecta. Normalmente, los desarrolladores, no los evaluadores, implementan y ejecutan pruebas de componentes.

La respuesta B es correcta. Esta es una característica del enfoque de "todo el equipo", que se basa en la cooperación de todas las partes interesadas.

La respuesta C es incorrecta. El representante empresarial no es competente para elegir las herramientas para el equipo de desarrollo: es el equipo el que elige las herramientas que quiere utilizar y la decisión formal la toma la dirección o, en metodologías ágiles, el propio equipo.

La respuesta D es incorrecta. El cliente no es competente en el diseño de pruebas no funcionales; esta tarea recae en los probadores y desarrolladores.

La respuesta E es correcta. La responsabilidad compartida y la atención a la calidad son dos de los principios básicos del enfoque de "todo el equipo".

Pregunta 1.14 (FL
1.5.3, K2)

Respuesta correcta: A

La respuesta A es correcta. Los evaluadores tienen una perspectiva diferente sobre el sistema bajo prueba que los desarrolladores y evitan muchos de los errores cognitivos de los autores de los productos de trabajo.

La respuesta B es incorrecta. Los desarrolladores pueden (y de hecho deberían) probar el código que crean.

La respuesta C es incorrecta. Así es como deberían trabajar los evaluadores, pero otras partes interesadas también pueden informar de los fallos de forma constructiva. Éste no es el motivo para realizar pruebas independientes.

La respuesta D es incorrecta. Encontrar defectos no debe considerarse una crítica a los desarrolladores, pero esto no tiene nada que ver con la independencia de las pruebas, sino sólo con el deseo de una cooperación armoniosa entre desarrolladores y evaluadores.

Respuestas a las preguntas del cap. 2

Pregunta 2.1
(FL-2.1.1, K2)

Respuesta correcta:D

La respuesta A es incorrecta. En un modelo secuencial, como el modelo V, y para software crítico para la vida como el piloto automático, las técnicas de prueba basadas en la experiencia deberían ser una técnica complementaria, no la principal.

La respuesta B es incorrecta. La elección del modelo SDLC no afecta directamente si habrá pruebas estáticas en el proyecto. Además, se considera una buena práctica realizar pruebas estáticas con antelación, especialmente para sistemas críticos para la vida, como el piloto automático.

La respuesta C es incorrecta. El modelo V es un modelo SDLC secuencial, por lo que no hay iteraciones. Además, debido a su naturaleza secuencial, el software en ejecución, incluso en forma de prototipo, normalmente sólo puede estar disponible en fases posteriores del ciclo de desarrollo.

La respuesta D es correcta. En las primeras fases de los modelos SDLC como el modelo V, los evaluadores suelen participar en revisiones de requisitos, análisis de pruebas y diseño de pruebas. El código ejecutable normalmente se desarrolla en fases posteriores, por lo que las pruebas dinámicas normalmente no se pueden realizar en las primeras fases del SDLC.

Pregunta 2.2

(FL-2.1.2, K1)

Respuesta correcta:D

En el modelo V, cada fase de desarrollo (el brazo izquierdo del modelo) corresponde a la fase de prueba asociada (el brazo derecho del modelo). En el modelo, esto está representado por las flechas horizontales entre las fases (por ejemplo, desde las pruebas de aceptación hasta la fase de requisitos; desde las pruebas del sistema hasta la fase de diseño, etc.).

Por tanto, la respuesta correcta es D.

Pregunta 2.3

(FL-2.1.3, K1)

Respuesta correcta: B

La respuesta A es incorrecta. TDD (desarrollo basado en pruebas) consiste en escribir pruebas de componentes de bajo nivel que no utilizan historias de usuarios.

La respuesta B es correcta. ATDD (desarrollo basado en pruebas de aceptación) utiliza criterios de aceptación de las historias de usuario como base para el diseño de casos de prueba.

La respuesta C es incorrecta. En el enfoque FDD (desarrollo basado en funciones), la base para el desarrollo de software son las funciones definidas, no las pruebas; este enfoque no tiene nada que ver con ATDD (ver respuesta correcta).

La respuesta D es incorrecta. BDD (desarrollo basado en el comportamiento) utiliza como base de prueba una descripción del comportamiento deseado del sistema, generalmente en formato Dado/Cuándo/Entonces.

Pregunta 2.4

(FL-2.1.4, K2)

Respuesta correcta: B

La respuesta A es incorrecta. DevOps no tiene nada en común con la generación automatizada de datos de prueba.

La respuesta B es correcta. Las actividades realizadas automáticamente después de que el desarrollador envía el código al repositorio, como el análisis estático, las pruebas de componentes o las pruebas de integración, permiten una retroalimentación muy rápida al desarrollador sobre el nivel de calidad del código confirmado por el desarrollador.

La respuesta C es incorrecta. Este tipo de actividad es posible en pruebas basadas en modelos, por ejemplo. El enfoque DevOps no se trata de la generación automatizada de casos de prueba.

La respuesta D es incorrecta. El enfoque DevOps no afecta el momento de la planificación del lanzamiento y la planificación de la iteración; Además, incluso si esto fuera cierto, no es un beneficio relacionado con las pruebas, sino más bien un beneficio de gestión de proyectos.

Pregunta 2.5

(FL-2.1.5, K2)

Respuesta correcta: A

La respuesta A es correcta. Un ejemplo del enfoque de desplazamiento a la izquierda es el uso de la prueba-primer enfoque ejemplificado por el desarrollo basado en pruebas de aceptación (ATDD).

La respuesta B es incorrecta. El enfoque de desplazamiento a la izquierda no distingue las pruebas exploratorias de ninguna manera. Más bien, el énfasis en el uso de tipos de pruebas específicos depende del análisis de riesgos.

La respuesta C es incorrecta. La creación de prototipos de GUI no es un ejemplo de uso de la Enfoque de desplazamiento a la izquierda, porque la creación en sí no está relacionada con las pruebas.

La respuesta D es incorrecta. Este es un ejemplo de un enfoque de cambio a la derecha, es decir, pruebas tardías, después de que el software se haya entregado al cliente, para monitorear el nivel de calidad del producto en el entorno operativo de manera continua.

Pregunta 2.6

(FL-2.1.6, K2)

Respuesta correcta: C.

La respuesta A es incorrecta. Los evaluadores deben participar en reuniones retrospectivas abordando todas las cuestiones planteadas en estas reuniones.

La respuesta B es incorrecta. Los evaluadores deben participar en todos los aspectos de la retrospectiva. reunión. El papel descrito se parece más al de un facilitador.

La respuesta C es correcta. Esta es la actividad típica realizada por un evaluador en un Reunión retrospectiva: para discutir lo que sucedió durante la iteración completa.

La respuesta D es incorrecta. Este no es el propósito de una reunión retrospectiva. El evaluador debe discutir lo que sucedió durante la última iteración.

Pregunta 2.7

(FL-2.2.1, K2)

Respuesta correcta: B

La respuesta A es incorrecta. Las pruebas de integración de componentes que estamos tratando aquí se centran en la interacción y comunicación entre componentes, no en los componentes en sí.

La respuesta B es correcta. Queremos realizar pruebas de integración. El diseño de la arquitectura es la base de prueba típica para este tipo de pruebas, porque generalmente describe cómo los distintos componentes del sistema se comunican entre sí.

La respuesta C es incorrecta. Los informes de análisis de riesgos son más útiles para las pruebas del sistema que las pruebas de integración.

La respuesta D es incorrecta. Las regulaciones suelen ser útiles para pruebas de alto nivel y validación, como en las pruebas de aceptación.

Pregunta 2.8

(FL-2.2.2, K2)

Respuesta correcta:D

Las respuestas A y C son incorrectas. Estas pruebas verifican "qué" hace el sistema y son por lo tanto, ejemplos de pruebas funcionales.

La respuesta B es incorrecta. Este es un ejemplo de prueba de caja blanca, no una prueba no funcional.

La respuesta D es correcta. Este es un ejemplo de prueba no funcional o, más precisamente, de prueba de rendimiento. Este tipo de prueba comprueba "cómo" funciona el sistema, no "qué" hace.

Pregunta 2.9

(FL-2.2.3, K2)

Respuesta correcta:D

Las pruebas de confirmación se realizan después de que se ha encontrado un defecto y se ha informado que se ha solucionado. Dado que no sabemos cuándo la prueba provocará una falla, ni generalmente sabemos cuánto tiempo llevará la reparación, no podemos predecir el momento de la ejecución de la prueba de confirmación. Por lo tanto, es imposible programar con precisión estas pruebas con antelación.

Todos los demás tipos de pruebas se pueden planificar con antelación y se puede incluir un cronograma para su ejecución en el plan de pruebas.

Pregunta 2.10

(FL-2.3.1, K2)

Respuesta correcta: C.

La respuesta A es incorrecta. No actualizamos el software, pero lo arreglamos.

La respuesta B es incorrecta. No se desprende del escenario que estamos realizando. cualquier actividad de migración de software.

La respuesta C es correcta. La modificación del software es uno de los eventos que desencadenan el mantenimiento. Reparar un defecto es una modificación del software.

La respuesta D es incorrecta. Si bien esto es un factor desencadenante de la mantenibilidad de los sistemas de IoT, en este escenario, queremos corregir un defecto, no introducir una nueva funcionalidad del sistema.

Respuestas a las preguntas del cap. 3

Pregunta 3.1

(FL-3.1.1, K1)

Respuesta correcta: A

La respuesta A es correcta. También se puede revisar el documento que define las reglas para las revisiones. Al hacerlo, no es necesario revisarlo según lo establecido en este documento; la revisión se puede realizar basándose en criterios de sentido común.

La respuesta B es incorrecta. El documento habla de las reglas para realizar revisiones, pero podemos revisarlo sin seguir las reglas que analiza, simplemente usando sentido común.

La respuesta C es incorrecta. El hecho de que un documento no sea producto del trabajo de algún El proceso específico no excluye que se incluya en una revisión.

La respuesta D es incorrecta. Las revisiones se pueden aplicar a cualquier documento comprensible para los humanos.

Pregunta 3.2

(FL-3.1.2, K2)

Respuesta correcta: B

La respuesta A es incorrecta porque no ejecutamos el código sino que solo analizamos sus propiedades.

La respuesta B es correcta, este es un ejemplo clásico de la ganancia que se obtiene al utilizar un sistema estático. técnica, en este caso, análisis estático.

La respuesta C es incorrecta porque la medición de la complejidad ciclomática, el análisis de esta medición y la refactorización del código no son actividades de gestión sino técnicas.

La respuesta D es incorrecta porque el análisis estático no es un ejemplo de técnica de prueba formal; tales técnicas incluyen partición de equivalencia, análisis de valores límite u otras técnicas de prueba de caja blanca o negra. El resultado del análisis estático no es un diseño de caso de prueba.

Pregunta 3.3

(FL-3.1.3, K2)

Respuesta correcta: C.

La respuesta A es incorrecta. Las pruebas estáticas detectan directamente defectos, no fallas. No puede detectar fallas, ya que no ejecutamos el producto de trabajo bajo prueba. Las pruebas dinámicas detectan directamente fallas, no defectos.

La respuesta B es incorrecta. Por ejemplo, las revisiones pueden realizarse en etapas muy tardías (por ejemplo, pueden verificar la documentación del usuario) y las pruebas dinámicas pueden comenzar temprano en la fase de implementación.

La respuesta C es correcta. El análisis estático y el análisis dinámico tienen los mismos objetivos (ver Apartado 1.1.1), como la identificación de defectos (directa o indirectamente, a través de fallos).

lo antes posible en el ciclo de desarrollo. Entonces, en cuanto al propósito, no existe diferencia entre estas técnicas.

La respuesta D es incorrecta. En primer lugar, las revisiones no suelen requerir conocimientos de programación; en segundo lugar, no responde a la pregunta, que se refería al criterio del propósito, no a las habilidades requeridas.

Pregunta 3.4

(FL-3.2.1, K1)

Respuesta correcta: C.

La oración (i) es falsa; los desarrolladores implementan aquellas características que requiere el negocio y son parte de la iteración. Cuando completan sus tareas, apoyan otras tareas relacionadas con la iteración.

La oración (ii) es verdadera; La retroalimentación frecuente ayuda a centrar la atención en las características de mayor valor.

La oración (iii) es falsa; La retroalimentación temprana puede incluso resultar en la necesidad de realizar más pruebas, debido a cambios frecuentes o significativos.

La oración (iv) es verdadera; Los usuarios indican qué requisitos se omiten o malinterpretados, lo que da como resultado un producto final que satisface mejor sus necesidades.

Por tanto, la respuesta correcta es C.

Pregunta 3.5

(FL-3.2.2, K2)

Respuesta correcta: C.

Las respuestas A y D son incorrectas. Estas actividades forman parte de la fase de planificación.

La respuesta B es incorrecta. La recopilación de métricas es parte de la actividad de "corrección e informes".

La respuesta C es correcta. Según el plan de estudios, esta es una actividad que se realiza durante la fase de inicio de la revisión.

Pregunta 3.6

(FL-3.2.3, K1)

Respuesta correcta: B

La respuesta A es incorrecta. El papel del líder de la revisión es la responsabilidad general de la revisión, así como también decidir quién participará en la revisión y dónde y cuándo se realizará la revisión. Así, el líder tiene un rol organizativo, mientras que el facilitador (ver la respuesta correcta) tiene un rol técnico directamente relacionado con la realización de reuniones de revisión.

La respuesta B es correcta. El moderador (también conocido como facilitador) es responsable para garantizar que las reuniones de revisión se desarrollos con eficacia.

La respuesta C es incorrecta. Puede ser responsabilidad del autor hacer una presentación o comentar sobre el producto de su trabajo, pero el autor nunca asume el rol de moderador.

La respuesta D es incorrecta. El trabajo de los revisores es revisar sustancialmente el producto del trabajo, no moderar las reuniones. El moderador es un rol que precisamente libera a los revisores de la necesidad de anotar las observaciones realizadas por ellos durante la reunión de revisión.

Pregunta 3.7

(FL-3.2.4, K2)

Respuesta correcta:D

Comprender el producto del trabajo (o aprender algo) es uno de los objetivos de un recorrido. Los recorridos pueden tomar la forma de los llamados ensayos, por lo que, como resultado de este tipo de revisión, el equipo puede comprender de manera más efectiva cómo funciona el software y, por lo tanto, descubrir más fácilmente la causa de una falla extraña.

Pregunta 3.8

(FL-3.2.5, K1)

Respuesta correcta: C.

La respuesta A es incorrecta. El objetivo principal de la inspección es encontrar defectos.

Evaluar alternativas es un objetivo más apropiado para una revisión técnica.

La respuesta B es incorrecta. Las inspecciones suelen ser realizadas por pares del autor.

La presencia de la dirección en una reunión de revisión conlleva el riesgo de malinterpretar el propósito de la revisión y, por ejemplo, la evaluación de riesgos realizada por esa dirección de los miembros individuales de la reunión.

La respuesta C es correcta. La formación en técnicas de revisión es uno de los factores de éxito de las revisiones.

La respuesta D es incorrecta. Medir métricas ayuda a mejorar el proceso de revisión.

Además, en una revisión formal como la inspección, la recopilación de métricas es una actividad obligatoria.

Respuestas a las preguntas del cap. 4

Pregunta 4.1

(FL-4.1.1, K2)

Respuesta correcta: B

El análisis del dominio de entrada del software se realiza cuando se utiliza la prueba de caja negra. técnicas como la partición de equivalencia y el análisis de valores límite.

Pregunta 4.2

(FL-4.1.1, K2)

Respuesta correcta:D

La respuesta A es incorrecta. Describe una característica común de las técnicas de prueba de caja blanca.

La respuesta B es incorrecta. Si ya queremos diseñar datos de prueba basados en código análisis, necesitamos tener acceso a él, para poder hacerlo mediante pruebas de caja blanca.

La respuesta C es incorrecta. Describe cómo se mide la cobertura en técnicas de prueba de caja blanca.

La respuesta D es correcta. Las técnicas enumeradas en la pregunta son técnicas de prueba de caja negra. Según el programa de estudios, en las técnicas de prueba de caja negra, las condiciones de prueba, los datos de prueba y los casos de prueba se derivan de una base de prueba externa al objeto bajo prueba (por ejemplo, requisitos, especificaciones, casos de uso, etc.). Por tanto, los casos de prueba podrán detectar discrepancias entre estos requisitos y su implementación real.

Pregunta 4.3

(FL-4.2.1, K3)

Respuesta correcta: A

El dominio es el importe de la compra. Existen tres particiones de equivalencia para este dominio según la especificación:

- Partición "Sin descuento" {0.01, 0.02, ..., 99.98, 99.99} • Partición "5% de descuento" {100.00, 100.01, ..., 299.98, 299.99} • Partición "10% de descuento" {300.00, 300.01, 300.02, ...}

Tenga en cuenta que la cantidad más pequeña posible es 0,01 como el número positivo más pequeño posible.

La respuesta A es correcta. Los tres valores 0,01, 100,99 y 500, cada uno de los cuales pertenece a una partición diferente, cubra las tres particiones de equivalencia.

La respuesta B es incorrecta. Sólo tenemos tres particiones, por lo que el conjunto mínimo de Los valores tomados para las pruebas deben tener tres elementos.

La respuesta C es incorrecta. Los valores 1 y 99 pertenecen a la misma partición. Este conjunto no contiene un valor por el cual el sistema debería asignar un descuento del 10%, por lo que la última partición no está cubierta.

La respuesta D es incorrecta. Todos los valores pertenecen a la partición "sin descuento". Son valores que representan posibles porcentajes de descuento (0, 5, 10), pero no estamos analizando el dominio del tipo de descuento sino el dominio de entrada (cantidad de compras).

Pregunta 4.4

(FL-4.2.1, K3)

Respuesta correcta: C.

Se supone que debemos comprobar dos situaciones: una en la que la máquina no da cambio y uno en el que da cambio.

La respuesta A es incorrecta. Tanto el escenario 1 como el escenario 2 son escenarios en los que la máquina no devuelve ningún cambio. Entonces nos perdemos un caso en el que se da un cambio.

La respuesta B es incorrecta. Ambos escenarios sólo cubren el caso en el que la máquina da cambio. Nos perdemos una prueba en la que la máquina no da cambio.

La respuesta C es correcta. En el escenario 1, la máquina no da cambio (la cantidad insertada es exactamente igual a 75c), y en el escenario 3, la máquina da cambio de 25c.

La respuesta D es incorrecta, porque dos escenarios son suficientes para cubrir todas (dos) las particiones de equivalencia; consulte la justificación de la respuesta correcta.

Pregunta 4.5

(FL-4.2.1, K3)

Respuesta correcta: A

Para lograr un 100% de equivalencia dividiendo la cobertura de "cada elección", debemos cubrir todos los tipos de tarjetas y todos los descuentos. Los casos de prueba existentes no cubren únicamente la tarjeta diamante y el descuento del 5%. Estos dos elementos se pueden cubrir con uno.

caso de prueba adicional:

PT05: tarjeta diamante, 5%.

Por tanto, la respuesta correcta es A.

Pregunta 4.6

(FL-4.2.2, K3)

Respuesta correcta:D

El dominio considerado es la longitud de la contraseña y las particiones equivalentes tienen el siguiente aspecto:

- Contraseña demasiado corta: {0, 1, 2, 3, 4, 5} •

Contraseña con la longitud correcta: {6, 7, 8, 9, 10, 11} • Contraseña

demasiado larga: {12, 13, 14, ...}

Los casos de prueba existentes logran una cobertura BVA del 100 % de 2 valores, por lo que deben cubrir todos los valores límite, es decir, 0, 5, 6, 11 y 12. Para lograr la cobertura BVA del 100 % de 3 valores, debemos cubrir los siguientes valores:

- 0, 1 (para el valor límite 0) • 4, 5, 6 (para el valor límite 5) • 5, 6, 7 (para el valor límite 6)
- 10, 11, 12 (para el valor límite 11) • 11, 12, 13 (para el valor límite 12)

Entonces, en total, se deben probar los valores 0, 1, 4, 5, 6, 7, 10, 11, 12 y 13.

Sin embargo, como sabemos que los valores 0, 5, 6, 11 y 12 ya están en nuestro conjunto de prueba (porque se logra el 100% de BVA de 2 valores), los valores que faltan son 1, 4, 7, 10 y 13. Por tanto, la respuesta D es correcta.

Pregunta 4.7

(FL-4.2.2, K3)

Respuesta correcta:D

Lograr una cobertura total de BVA no es factible. Los números de lavados gratuitos consecutivos son todos múltiplos de 10: {10, 20, 30, 40, 50,...}. Pero para aplicar el método BVA, las particiones deben ser consistentes, es decir, no deben tener "huecos". Por lo tanto, si quisieramos aplicar el BVA a este problema, tendríamos que derivar infinitas particiones de equivalencia:

{1, ..., 9}, {10}, {11, ..., 19}, {20}, {21, ..., 29}, {30}, {31, ..., 39 }, {40}, etc...,

y esto significaría que tenemos que ejecutar infinitos casos de prueba (ya que tenemos infinitos valores límite: 1, 9, 10, 11, 19, 20, 21, 29, 30, 31, etc.).

Tenga en cuenta que si utilizáramos el método de partición de equivalencia, el problema podría resolverse, porque tendríamos solo dos particiones: números divisibles por 10 y otros números. Por lo tanto, sólo dos pruebas, como 9 y 10, serían suficientes para lograr la cobertura de las particiones de equivalencia.

Pregunta 4.8

(FL-4.2.3, K3)

Respuesta correcta: A

Los requisitos son contradictorios si, para una combinación dada de condiciones, podemos indicar dos conjuntos diferentes de acciones correspondientes. En nuestro caso, las dos acciones diferentes son "viaje gratis"=Sí y "viaje gratis"=NO. Para forzar el valor de NO se debe cumplir la condición "estudiante"=Sí. Para forzar el valor de Sí, debe aparecer "miembro del parlamento"=Sí o "persona discapacitada"=Sí.

La respuesta A es correcta. Esta combinación coincide con las reglas R1 y R3, que dan acciones contradictorias.

La respuesta B es incorrecta. Esta combinación sólo coincide con las reglas R1 y R2, que resulta en la misma acción.

La respuesta C es incorrecta. Esta combinación sólo coincide con la columna R3, por lo que puede haber No habrá contradicción dentro de una sola regla.

La respuesta D es incorrecta. No se ajusta ni a la regla R1 ni a R2 ni a R3. Entonces este es un ejemplo de un requisito faltante, pero no de requisitos contradictorios.

Tabla 1 Combinaciones de condiciones

Edad combinada		Ganancias de residencia	
1	Hasta 18	Ciudad	Hasta 4000/mes
2	Hasta 18	Ciudad	Desde 4001/mes
3	Hasta 18	Aldea	Hasta 4000/mes
4	Hasta 18	Aldea	Desde 4001/mes
5	19–40	Ciudad	Hasta 4000/mes
6	19–40	Ciudad	Desde 4001/mes
7	19–40	Aldea	Hasta 4000/mes
8	19–40	Aldea	Desde 4001/mes
9	Desde 41	Ciudad	Hasta 4000/mes
10	Desde 41	Ciudad	Desde 4001/mes
11	Desde 41	Aldea	Hasta 4000/mes
12	Desde 41	Aldea	Desde 4001/mes

Pregunta 4.9

(FL-4.2.3, K3)

Respuesta correcta: C.

El número de columnas en la tabla de decisiones completa es igual al número de todas posibles combinaciones de condiciones. Como tenemos tres condiciones, con 3, 2 y 2 opciones posibles, respectivamente, el número de todas las combinaciones posibles de estas condiciones es $3 \times 2 \times 2 = 12$. Estas se enumeran en la Tabla 1.

Pregunta 4.10

(FL-4.2.4, K3)

Respuesta correcta: A

Como tenemos tres estados y cuatro eventos, hay $3 \times 4 = 12$ combinaciones posibles (estado, evento). La tabla de la Pregunta 4.10 contiene sólo cuatro de ellos (es decir, sólo hay cuatro transiciones válidas en la máquina). Por lo tanto, las transiciones inválidas son $12 - 4 = 8$. Aquí están (el estado y el evento se colocan entre paréntesis secuencialmente):

1. (Inicial, Iniciar sesiónOK)
2. (Inicial, Error de inicio de sesión)
3. (Inicial, Cerrar sesión)
4. (Iniciar sesión, iniciar sesión)
5. (Iniciar sesión, cerrar sesión)
6. (Registrado, Iniciar sesión)
7. (Iniciado sesión, iniciar sesión Aceptar)
8. (Registrado, Error de inicio de sesión)

Ninguna de estas combinaciones aparece en la lista de transiciones válidas proporcionada en la tarea.

Otra forma de demostrar que hay ocho transiciones no válidas es contar celdas vacías en la tabla de estados, que se ve como se muestra en la Tabla 2.

Tabla 2 Tabla de estados para la Pregunta 4.10

Transición de estado	Acceso	Iniciar sesiónOK	Error de inicio de sesión	Cerrar sesión
Inicial	Inicio sesión			
Inicio sesión		registrado	Inicial	
registrado				Inicial

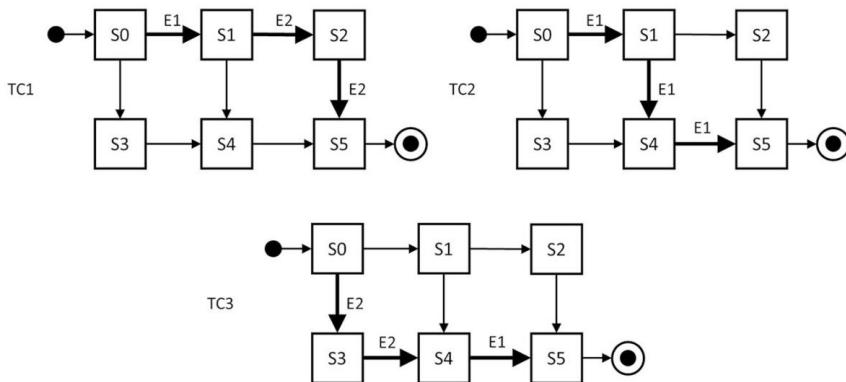


Fig. 1 Rutas determinadas mediante pruebas que cubren todas las transiciones válidas (Pregunta 4.11)

Pregunta 4.11

(FL-4.2.4, K3)

Respuesta correcta: B

Tenga en cuenta que no pueden ocurrir dos de las siguientes tres transiciones dentro de un solo caso de prueba:

- S0 > (E2) > S3.
- S1 > (E1) > S4.
- S2 > (E2) > S5.

Esto significa que necesitamos al menos tres casos de prueba para cubrir todas las transiciones válidas. En De hecho, tres casos de prueba son suficientes, y estos son:

TC1. S0 > (E1) > S1 > (E2) > S2 > (E2) > S5.

TC2. S0 > (E1) > S1 > (E1) > S4 > (E1) > S5.

TC3. S0 > (E2) > S3 > (E2) > S4 > (E1) > S5.

Las rutas definidas por estos casos de prueba se muestran en la Fig. 1. Tenga en cuenta que cada transición (flecha) está cubierto por al menos un caso de prueba.

Pregunta 4.12

(FL-4.3.1, K2)

Respuesta correcta:D

La respuesta A es incorrecta. El criterio de cobertura de sucursales incluye la declaración criterio de cobertura, pero no al revés. Por ejemplo, para el código:

```
1. SI (x==0) ENTONCES 2. x :=  
x+1  
3. REGRESAR x
```

un caso de prueba (para $x = 0$) dará como resultado el paso de las rutas 1, 2 y 3 y, por lo tanto, logrará una cobertura de declaración del 100%, pero esta prueba cubre solo dos de las tres ramas: (1, 2) y (2, 3). Se descubre la rama (1, 3), que se ejecutará cuando la entrada x sea distinta de cero.

La respuesta B es incorrecta. El ejemplo de código anterior muestra esto. El caso de prueba ($x = 0$) logra una cobertura de código del 100% pero solo provoca el resultado verdadero de la decisión en la declaración 1. No tenemos una prueba que fuerce un resultado falso para esta decisión.

La respuesta C es incorrecta. El programa anterior puede devolver cualquier número distinto de cero. La prueba ($x = 0$) logra una cobertura de declaración del 100% pero solo fuerza el retorno del valor $x = 1$.

La respuesta D es correcta. La cobertura de declaraciones fuerza la ejecución de cada declaración en el código, por lo que, en particular, significa ejecutar cada declaración que contenga un defecto. Por supuesto, esto no significa que se desencadenen todos los fallos causados por estos defectos, porque la ejecución de una declaración no válida puede no tener ningún efecto negativo. Por ejemplo, la ejecución de la sentencia $x := a / b$ será completamente correcta, siempre y cuando el denominador (b) no sea igual a 0.

Pregunta 4.13

(FL-4.3.2, K2)

Respuesta correcta: C.

La cobertura de ramas requiere que las pruebas cubran todos los flujos de control posibles entre declaraciones ejecutables en el código, es decir, todas las ramas posibles en el código, tanto incondicionales como condicionales. El código bajo prueba tiene una estructura lineal, y cada ejecución de este código ejercitara declaraciones en el orden 1, 2, 3. Esto significa que cada vez que se ejecuta este programa, se cubrirán las dos ramas incondicionales que ocurren en él: (1 , 2) y (2, 3). Por lo tanto, la respuesta correcta es C: un caso de prueba con cualquier dato de entrada x , y es suficiente.

Pregunta 4.14

(FL-4.3.3 (K2))

Respuesta correcta: A

La respuesta A es correcta. Esta es una propiedad fundamental y una ventaja de las técnicas de prueba de caja blanca. En este enfoque, las pruebas se diseñan directamente sobre la base de la estructura de lo que se va a probar (por ejemplo, el código fuente), por lo que no es necesaria una especificación completa y precisa para el diseño de la prueba en sí (excepto para derivar el resultado esperado). .

La respuesta B es incorrecta. Las técnicas de caja blanca no siempre requieren habilidades de programación. Se pueden utilizar, por ejemplo, en el nivel de prueba del sistema, donde la estructura cubierta es, por ejemplo, el menú del programa.

La respuesta C es incorrecta. No existe relación entre las métricas de cobertura de

Técnicas de caja negra y caja blanca.

La respuesta D es incorrecta. No existe una relación directa entre la cobertura de la caja blanca y el riesgo, porque el nivel de riesgo depende específicamente del impacto del riesgo, no sólo de cuántas líneas de código tiene la función asociada con un riesgo particular.

Pregunta 4.15

(FL-4.4.1, K2)

Respuesta correcta:D

La respuesta A es incorrecta. El documento no menciona valores límite.

La respuesta B es incorrecta. En las pruebas basadas en listas de verificación, la lista de verificación define las características "positivas" del software, mientras que el documento analizado habla de posibles fallas.

La respuesta C es incorrecta. Estos no son casos de uso.

La respuesta D es correcta. El documento que aparece en la pregunta es una lista de posibles defectos o fallos. Estas listas se utilizan en la técnica de ataques de fallos, un enfoque formalizado para adivinar errores.

Pregunta 4.16

(FL-4.4.2, K2)

Respuesta correcta: A

Las pruebas exploratorias utilizan el conocimiento, las habilidades, la intuición y la experiencia del evaluador, pero le brindan total margen de maniobra en lo que respecta al repertorio de técnicas que pueden utilizar en una prueba exploratoria basada en sesiones. Por tanto, la respuesta A es correcta.

Las respuestas B y C son incorrectas; consulte A.

La respuesta D es incorrecta. Aunque se permiten técnicas de prueba formales (ver A), la explicación en esta respuesta es incorrecta. En un enfoque exploratorio, no es necesario tener la base de prueba necesaria para derivar casos de prueba, ya que se trata de una técnica de prueba basada en la experiencia.

Pregunta 4.17

(FL-4.4.3, K2)

Respuesta correcta:D

La respuesta A es incorrecta. Aunque las listas de verificación se pueden organizar en torno a pruebas no funcionales, esta no es la principal ventaja de utilizar listas de verificación.

La respuesta B es incorrecta. En un enfoque basado en la experiencia, como las pruebas basadas en listas de verificación, es imposible definir con precisión medidas significativas de cobertura, especialmente medidas de cobertura de código.

La respuesta C es incorrecta. El uso de listas de verificación no requiere necesariamente experiencia (especialmente en el caso de listas de verificación detalladas y de bajo nivel); este enfoque se adapta más a las pruebas exploratorias.

La respuesta D es correcta. En ausencia de casos de prueba detallados, las pruebas basadas en listas de verificación pueden proporcionar cierto grado de coherencia a las pruebas.

Pregunta 4.18

(FL-4.5.1, K2)

Respuesta correcta:D

La respuesta A es incorrecta. El evaluador no puede decidir por sí solo los criterios de aceptación. Las historias de usuarios, incluidos los criterios de aceptación, se escriben en base a la cooperación del propietario del producto, el desarrollador y el evaluador.

La respuesta B es incorrecta. Esta solución no tiene sentido. En primer lugar, la planificación de la historia no es el momento de escribir exámenes. En segundo lugar, las pruebas de aceptación deben crearse sobre la base de criterios de aceptación establecidos y precisos y, por el momento, el equipo está en el proceso de negociar estos criterios y aún no está claro qué forma adoptarán.

La respuesta C es incorrecta. El escenario no dice nada sobre el rendimiento, pero incluso si este tema surgiera en la discusión, no se puede excluir al propietario del producto.

La respuesta D es correcta. Este es un ejemplo modelo de negociación de una historia de usuario por parte de todos los miembros del equipo como parte de un enfoque de prueba basado en la colaboración.

Pregunta 4.19

(FL-4.5.2, K2)

Respuestas correctas: B, C

La respuesta A es incorrecta. Los criterios de aceptación pueden abordar un aspecto no funcional como el rendimiento, pero el término "suficientemente rápido" utilizado en este criterio es impreciso y, por lo tanto, no se puede comprobar.

La respuesta B es correcta. Este es el mecanismo deseado para ofrecer funcionalidad en este software: el usuario sólo puede realizar compras cuando está registrado. Se trata de un criterio preciso y comprobable, directamente relacionado con el contenido de la historia.

La respuesta C es correcta. Los criterios de aceptación pueden tener en cuenta eventos "negativos", como que un usuario cometiera un error durante el proceso de registro, lo que puede resultar en una denegación de procesamiento posterior. Se trata de un criterio preciso y comprobable, directamente relacionado con el contenido de la historia.

La respuesta D es incorrecta. Si bien es un criterio de aceptación razonable, preciso y comprobable, no aborda directamente la historia desde el escenario. Esto se debe a que está escrito desde el punto de vista del operador del sistema, no desde el punto de vista del cliente de la tienda electrónica.

La respuesta E es incorrecta. Este es un ejemplo de una regla para escribir criterios de aceptación, No es un criterio de aceptación específico para una historia de usuario.

Pregunta 4.20

(FL-4.5.3, K3)

Respuesta correcta: B

La prueba 1 es inconsistente con la regla comercial. Se cumple la primera condición (el tiempo de conservación del libro más largo no supera los 30 días), pero después de tomar prestados dos libros nuevos, habiendo ya tomado prestados otros tres, el estudiante tendrá un total de cinco libros prestados. Esto no excederá el límite de la segunda condición de la regla comercial. Entonces el sistema debería permitir préstamos, pero en la Prueba 1, la decisión es "no permite".

La prueba 2 sigue la regla de negocio: el estudiante no conserva ninguno de los cuatro libros prestados por más de 30 días y quiere pedir prestado uno nuevo, por lo que tendrá un total de cinco libros prestados. No excederán el límite, por lo que el sistema les permite pedir prestado.

La prueba 3 sigue la regla de negocio: un profesor tiene al menos un libro en poder ($Días > 30$), por lo que el sistema no puede permitir el préstamo de libros.

La prueba 4 sigue la regla empresarial: un profesor tiene una cuenta limpia y quiere pedir prestados seis libros, lo que está dentro del límite de diez libros. El sistema debería permitir el préstamo.

Por lo tanto, sólo uno es incorrecto en las pruebas, por lo que la respuesta correcta es B.

Respuestas a las preguntas del cap. 5

Pregunta 5.1

(FL-5.1.1, K2)

Respuesta correcta: A

El plan de prueba debe estar en consonancia con la estrategia de prueba, y no al revés. Por tanto, la estrategia de prueba no forma parte del plan de prueba. Todos los demás elementos, restricciones presupuestarias, alcance de las pruebas y registro de riesgos, son partes del plan de pruebas. Por tanto, la respuesta correcta es A.

Pregunta 5.2

(FL-5.1.2, K2)

Respuesta correcta:D

La respuesta A es incorrecta. El análisis de riesgo detallado para historias de usuarios se realiza durante planificación de iteraciones, no durante la planificación de lanzamientos.

La respuesta B es incorrecta. Identificación de aspectos no funcionales del sistema a ser probado se realiza durante la planificación de la iteración, no durante la planificación del lanzamiento.

La respuesta C es incorrecta. Estimar el esfuerzo de prueba para las nuevas funciones planificadas para una iteración se realiza durante la planificación de la iteración, no durante la planificación del lanzamiento.

La respuesta D es correcta. Durante la planificación del lanzamiento, los evaluadores participan en la creación de Historias de usuario comprobables y sus criterios de aceptación.

Pregunta 5.3

(FL-5.1.3, K2)

Respuesta correcta:D

Los criterios de entrada incluyen, en particular, los criterios de disponibilidad y el nivel de calidad inicial. del objeto de prueba. Por lo tanto, los criterios de entrada son la disponibilidad de probadores (i) y pasar todas las pruebas de humo (iv). Por el contrario, los criterios de salida típicos son medidas de minuciosidad. Por tanto, la ausencia de defectos críticos (ii) y la consecución de un determinado umbral de cobertura de la prueba (iii) son criterios de salida. Por tanto, la respuesta correcta es D.

Pregunta 5.4

(FL-5.1.4, K3)

Respuesta correcta: B

El equipo quiere calcular $E(5)$, para lo cual necesitará los valores de $E(4)$, $E(3)$, y $E(2)$. Como se desconoce el valor de $E(4)$, el equipo debe primero estimar $E(4)$ y entonces $E(5)$. Según el modelo de extrapolación del esfuerzo, tenemos:

$$E(4) = \frac{1}{2} E(1) + \frac{1}{2} E(3) + E(5) = 15$$

Entonces $E(4) = 15$. Ahora podemos extraer el esfuerzo en la quinta iteración:

$$E(5) = \frac{1}{2} E(1) + \frac{1}{2} E(3) + E(5) = 16$$

Entonces $E(5) = 16$. Esto significa que la respuesta correcta es B.

Pregunta 5.5

(FL-5.1.5, K3)

Respuesta correcta: C.

El primer caso de prueba ejecutado será el que alcance la mayor cobertura de características, es decir, TC1, que cubre cuatro de las siete funciones: A, B, C y F. El segundo en orden será el que cubra la mayor parte de las características previamente descubiertas (es decir, D, E, G). Cada uno de los TC2, TC3 y TC4 cubre solo una de estas características adicionales, mientras que TC5 cubre dos características adicionales no cubiertas: D y G. Por lo tanto, TC5 se ejecutará en segundo lugar. Estos dos primeros casos de prueba, PT1 y PT5, cubren un total de seis de las siete funciones: A, B, C, D, F y G. El tercer caso de prueba será el que cubra la mayoría de las características no cubiertas. hasta ahora, es la característica E. Esta característica está cubierta solo por TC4, por lo que este caso de prueba se ejecutará en tercer lugar. Por tanto, la respuesta correcta es C.

Pregunta 5.6

(FL-5.1.6, K1)

Respuesta correcta: C.

La respuesta A es incorrecta. El esfuerzo del equipo no tiene nada que ver con el concepto de pirámide de pruebas.

La respuesta B es incorrecta. La pirámide de pruebas no se refiere a tareas de diseño sino directamente a pruebas en diferentes niveles de prueba.

La respuesta C es correcta. La pirámide de pruebas ilustra el hecho de que tenemos pruebas más granulares (detalladas) en niveles de prueba más bajos y, por lo tanto, normalmente se necesitan más pruebas de este tipo, porque cada prueba logra una cobertura relativamente baja. Cuanto mayor es el nivel, menor es la granularidad de las pruebas y normalmente se necesita un número decreciente de ellas, ya que normalmente una sola prueba de un nivel superior logra más cobertura que una sola prueba de un nivel inferior.

La respuesta D es incorrecta. La pirámide de pruebas no modela el esfuerzo de prueba; modela la granularidad y el número de pruebas en cada nivel de prueba.

Pregunta 5.7

(FL-5.1.7, K2)

Respuesta correcta: A

La respuesta A es correcta. Según el modelo de cuadrantes de pruebas, las pruebas de componentes (unitarias) se colocan en el cuadrante orientado a la tecnología y de apoyo al equipo, ya que este cuadrante contiene pruebas automatizadas y pruebas que forman parte del proceso de integración continua. Las pruebas de componentes suelen ser este tipo de pruebas.

Las respuestas B, C y D son incorrectas; consulte el fundamento de la respuesta correcta.

Pregunta 5.8

(FL-5.2.1, K1)

Respuesta correcta: A

La respuesta A es correcta: la probabilidad del riesgo y el impacto del riesgo son factores independientes.

La respuesta B es incorrecta: consulte el fundamento de la respuesta A.

La respuesta C es incorrecta: consulte el fundamento de la respuesta A.

La respuesta D es incorrecta: el impacto del riesgo debe evaluarse antes de que ocurra.

Pregunta 5.9

(FL-5.2.2, K2)

Respuesta correcta: B

La respuesta A es incorrecta. Este es un ejemplo extremo de la materialización del riesgo del producto.

La respuesta B es correcta. La aparición de riesgos en el proyecto a menudo resulta en problemas relacionados con retrasos en las tareas del proyecto.

La respuesta C es incorrecta. Los altos costos de mantenimiento del software se deben a defectos de mantenibilidad del producto, es decir, son consecuencia de riesgos del producto, no de riesgos del proyecto.

La respuesta D es incorrecta. La insatisfacción del cliente se debe a un defecto del producto, por lo que es consecuencia de los riesgos del producto, no de los riesgos del proyecto.

Pregunta 5.10

(FL-5.2.3, K2)

Respuesta correcta: A

La respuesta A es correcta. Una revisión técnica para observar posibles problemas de usabilidad asociados con la interfaz parece ser una idea razonable, al igual que verificar el flujo de control en el código del mecanismo de transferencia aplicando cobertura de rama.

La respuesta B es incorrecta. Es posible que las pruebas de componentes no muestren problemas relacionados con la interfaz de transferencia; no sabemos cómo es la arquitectura de la aplicación.

Las pruebas de componentes tienen un nivel de prueba demasiado bajo para esto.

La respuesta C es incorrecta. La usabilidad no está relacionada con la implementación de la lógica empresarial de la aplicación.

La respuesta D es incorrecta. Las pruebas de caja blanca no abordan los riesgos relacionados con usabilidad de la interfaz.

Pregunta 5.11

(FL-5.2.4, K2)

Respuesta correcta: C.

La compra de un seguro transfiere la carga del riesgo a un tercero, en este caso el asegurador. Este es un ejemplo de transferencia de riesgo, por lo que la respuesta correcta es la C.

Pregunta 5.12

(FL-5.3.1, K1)

Respuesta correcta: B

La respuesta A es incorrecta. El nivel de riesgo residual es una métrica típica utilizada en las pruebas, como expresa el nivel actual de riesgo residual en un producto después del ciclo de prueba.

La respuesta B es correcta. La cobertura de requisitos por código fuente no tiene nada que ver con las pruebas. Esta métrica puede representar el progreso del trabajo de desarrollo, no las pruebas.

La respuesta C es incorrecta. El número de defectos críticos encontrados está directamente relacionado con las pruebas.

La respuesta D es incorrecta. El progreso de la implementación del entorno de prueba se refiere a una actividad importante realizada como parte del proceso de prueba, por lo que es una métrica utilizada en las pruebas.

Pregunta 5.13

(FL-5.3.2, K2)

Respuesta correcta:D

La respuesta A es incorrecta. La información sobre riesgos no mitigados es una información típica contenida en un informe de finalización de prueba.

La respuesta B es incorrecta. Las desviaciones del plan de prueba son una información típica contenida en un informe de finalización de la prueba.

La respuesta C es incorrecta. La información sobre defectos es una información típica contenida en un informe de finalización de la prueba.

La respuesta D es correcta. La información típica incluida en un informe de progreso de prueba son las pruebas programadas para el siguiente período de informe. Esta no es una información típica incluida en un informe de finalización de prueba, ya que este tipo de informe describe un alcance de trabajo cerrado y completado para el cual no habrá más períodos de informe.

Pregunta 5.14

(FL-5.3.3, K2)

Respuesta correcta: B

No existe un único y mejor método de comunicación. Por ejemplo, los informes formales o los correos electrónicos no serán útiles cuando el equipo necesite comunicarse de forma rápida, frecuente y en tiempo real. Por otro lado, la comunicación verbal "cara a cara" será imposible cuando el equipo esté disperso y trabaje en muchas zonas horarias diferentes. La forma de comunicación siempre debe elegirse individualmente, según las circunstancias, teniendo en cuenta diversos factores contextuales. Por tanto, la respuesta correcta es B.

Pregunta 5.15

(FL-5.4.1, K2)

Respuesta correcta: A

La respuesta A es correcta. El propósito de la gestión de la configuración es garantizar y mantener la integridad del componente/sistema, el software de prueba relacionado y las interrelaciones entre ellos durante todo el ciclo de vida del proyecto y del software para que se puedan realizar las actividades descritas en el escenario.

La respuesta B es incorrecta. El análisis de impacto puede determinar la magnitud o el riesgo de un cambio, pero no identifica los productos de trabajo de origen según la versión del software.

La respuesta C es incorrecta. La entrega continua ayuda a automatizar el lanzamiento del software proceso, pero no garantiza la integridad y el control de versiones de los productos de trabajo.

La respuesta D es incorrecta. Las retrospectivas son para mejorar los procesos; no proporcionan integridad ni control de versiones de los productos de trabajo.

Pregunta 5.16

(FL-5.5.1, K3)

Respuesta correcta: A

La respuesta A es correcta. El informe carece de información sobre los pasos para reproducir la prueba. Por ejemplo, es posible que el desarrollador no sepa por qué el informe dice que se aceptan contraseñas de longitud 6, 7 y 9, pero no de longitud 8. Los pasos a reproducir podrían permitirle al desarrollador verificar esto rápidamente.

La respuesta B es incorrecta: la versión del producto (en forma de la última fecha de compilación) aparece en el informe de defectos.

La respuesta C es incorrecta. Cuando creamos un informe de defectos utilizando una herramienta de gestión de defectos, lo más probable es que se asigne automáticamente un estado "abierto". Además, esta no es información tan crucial como la que se proporciona en la respuesta A.

La respuesta D es incorrecta. Esta información es útil para el evaluador, pero no es necesario incluirla en el informe de defectos y puede ser de poco valor para el desarrollador responsable de solucionar el defecto.

Respuestas a las preguntas del cap. 6

Pregunta 6.1

(FL-6.1.1, K2)

Respuesta correcta: B

Según el programa de estudios (Sección 6.1), herramientas de apoyo a la gestión de pruebas incluye herramientas de gestión de requisitos. Por tanto, la respuesta correcta es B.

Pregunta 6.2

(FI-6.1.2, K1)

Respuesta correcta: C, D

Según el plan de estudios, los beneficios son, en particular, una mayor coherencia y repetibilidad de las pruebas (C) y una evaluación objetiva mediante el uso de definiciones operativas de medición bien definidas (D). La dependencia excesiva de la herramienta (A), la dependencia del proveedor (B) y los errores al estimar el costo de mantenimiento de la herramienta (E) son riesgos. Por tanto, las respuestas correctas son C y D.

Soluciones a los ejercicios



Soluciones a los ejercicios del cap. 4

Ejercicio 4.1

(FL-4.2.1, K3)

- A) Tenemos dos dominios: banda = {Iron Maiden, Judas Priest, Black Sabbath} y tipo de entrada = {frente al escenario, lejos del escenario}. Cada elemento de cada dominio será una partición de equivalencia de un elemento separada. Así, la división del dominio "banda" queda de la siguiente manera: {Iron Maiden}, {Judas Priest}, {Black Sabbath}; la división del dominio "tipo de entrada" es la siguiente: {delante del escenario}, {fuera del escenario}.
- B) No hay particiones no válidas en el problema, porque la forma en que se seleccionan la banda y el tipo de boleto lo hace imposible: el usuario selecciona estos valores de listas desplegables predefinidas. Por supuesto, es posible considerar una situación en la que, después de enviar el formulario, esta consulta sea interceptada y modificada, por ejemplo, el nombre del equipo se cambia por uno inexistente. Aquí, sin embargo, nos centramos únicamente en pruebas puramente funcionales, restringiendo la interacción usuario-sistema solo a GUI, y no consideramos problemas de pruebas de seguridad de aplicaciones avanzadas.
- C) El conjunto de pruebas debe cubrir cada una de las tres particiones del dominio "banda" y cada una de las dos particiones del dominio "tipo de billete". Dado que cada caso de prueba cubre una partición de cada uno de estos dominios, tres casos de prueba serán suficientes, por ejemplo:

Banda TC1 = Iron Maiden, tipo de entrada = frente al escenario.

Banda TC2 = Judas Priest, tipo de entrada = fuera del escenario.

Banda TC3 = Black Sabbath, tipo de entrada = frente al escenario.

También cabe señalar que en cada caso de prueba, también es necesario especificar el resultado esperado. En nuestro caso será la cesión de una entrada de un tipo concreto para un concierto de una banda concreta.

Ejercicio 4.2

(FL-4.2.1, K3)

- A) El dominio de entrada (válido) son los números naturales mayores que 1. Podemos dividir este dominio en dos particiones: números primos y números compuestos, a saber, {2, 3, 5, 7, 11, 13, ...} y {4, 6, 8, 9, 10, 12, ...}.
- B) La interfaz no permite al usuario ingresar un valor que no sea un número natural.
 Por tanto, podemos considerar que no hay particiones inválidas en el problema.
 Sin embargo, si pudiéramos, por ejemplo, interceptar el mensaje que pasa por el valor de entrada al sistema y modificarlo en consecuencia, entonces podríamos forzar una valor no válido. Si esto es posible y si los evaluadores eligen tomar tal
 El enfoque de "piratería" depende, por supuesto, de una serie de factores, en particular la nivel requerido de seguridad de la aplicación. Si consideramos el problema sólo desde el
 Desde la perspectiva del usuario, podemos asumir con seguridad que el programa funcionará sólo en valores esperados correctos.
- C) Dado que no hay particiones válidas en el problema, dos pruebas son suficientes para cubrir todas las particiones de equivalencia: una en la que consideramos un número primo y otra en que consideramos un número compuesto, por ejemplo:

TC1: la entrada es un número primo (por ejemplo, 7),

TC2: la entrada es un número compuesto (por ejemplo, 12).

Ejercicio 4.3

(FL-4.2.2, K3)

El dominio bajo análisis es el monto total de compras. es un positivo número medible con dos decimales (es decir, hasta 1 centavo). Necesitamos encontrar aquellos de estos valores que, después de redondear, serán los valores límite para el importe redondeado (ver Tabla 1). El valor de entrada más pequeño posible que cumpla con las condiciones de la tarea es \$0.01.

El valor 300 es la cantidad más grande, que después de redondear (=300) dará el valor límite máximo para 0% de descuento. El valor 300.01 es la cantidad más pequeña que, después de redondear (=301), dará el valor límite mínimo para un 5% descuento, etc

Entonces tenemos los siguientes casos de prueba:

- TC1: monto = 0,01, resultado esperado: 0% de descuento.
- TC2: monto = 300, resultado esperado: 0% de descuento.
- TC3: importe = 300,01, resultado esperado: 5% de descuento.

Tabla 1 Valores límite antes y después del redondeo

Descuento	Valores límite para la cantidad redondeada		Valores límite para el importe anterior redondeo	
	Mínimo	Máximo	Mínimo	Máximo
0%	1	300	0,01	300
5%	301	800	300.01	800
10%	801	-	800.01	-

- TC4: monto = 800, resultado esperado: 5% de descuento.
- TC5: monto = 800.01, resultado esperado: 10% de descuento.

Ejercicio 4.4

(FL-4.2.2, K3)

A)

Particiones para el parámetro "ancho":

- Partición válida: {30, 31, ..., 99, 100}

Particiones para el parámetro "altura":

- Partición válida: {30, 31, ..., 59, 60}

Particiones para la "zona" (el precio del servicio dependerá de su valor):

- Partición válida por el precio de \$450: {900, 901, ..., 1600}
- Partición válida por el precio de \$500: {1601, 1602, ..., 6000}

Los valores de "área" se derivan del hecho de que las dimensiones mínimas de

la imagen mide 30 cm de ancho y 30 cm de alto, por lo que la superficie mínima es
 $30\text{cm} \times 30\text{cm} = 900\text{cm}^2$. Asimismo, las dimensiones máximas son 100 cm de ancho.
y 60 cm de altura, por lo que el área máxima es $60\text{ cm} \times 100\text{ cm} = 6000\text{ cm}^2$.

Valores límite:

- Para "ancho": (W1) 30, (W2) 100.
- Para "altura": (H1) 30, (H2) 60.
- Para "área": (A1) 900, (A2) 1600, (A3) 1601, (A4) 6000.

B)

Representaremos el caso de prueba como un par (w, h) , donde w y h son el ancho y altura (entrada), respectivamente. Necesitamos cubrir ocho valores límite con pruebas: W1, W2, H1, H2, A1, A2, A3 y A4. Tenga en cuenta que algunos valores límite para el área pueden ser obtenidos multiplicando los valores que son el límite de alto y ancho valores. Por ejemplo, $900 = 30 \times 30$ y $6000 = 100 \times 60$. Podemos usar esto para minimizar el número de casos de prueba.

Los casos de prueba y los valores límite cubiertos se muestran en la Tabla 2.

Tenga en cuenta que es imposible cubrir el valor límite de A3. Esto se debe a que 1601 es un número primo, por lo que no se puede expresar como producto de dos números mayores mayor o igual a 30.

Tabla 2 Valores límite cubiertos

TC	Aporte		Área	Valores límite cubiertos para:		
	Ancho	Altura		Ancho	Altura	Área
1	30	30	900	W1	H1	A1
2	100	60	6000	W2	H2	A4
3	40	40	1600			A2

Diseñamos tres casos de prueba TC1, TC2 y TC3, que cubren siete de los ocho valores límite identificados. El número más pequeño perteneciente a la partición {1601, 1602, ..., 6000} que se puede representar como una multiplicación de dos números que cumplen las restricciones dadas, es $1610 = 35 * 46$. Podríamos sumar el cuarto caso de prueba (35, 46) que ejerce este valor límite 'factible' para la partición {1601, 1602, ..., 6000}.

Ejercicio 4.5

(FL-4.2.3, K3)

A) Las condiciones que ocurren en nuestro problema son "puntos ≥ 85 " (valores posibles: Sí, NO) y "número de errores ≤ 2 " (valores posibles: Sí, NO).

B) El sistema puede realizar las siguientes acciones:

- Otorgar licencia de conducir (Sí, NO) • Repetir examen teórico (Sí, NO) • Repetir examen práctico (Sí, NO) • Clases de manejo adicionales (Sí, NO)

C) Todas las combinaciones de condiciones se muestran en la parte superior de la Tabla 3 y se pueden generar utilizando el método de "árbol" descrito anteriormente en este capítulo. Como tenemos dos condiciones y cada una de ellas toma uno de los dos valores posibles, tenemos $2^2 = 4$ combinaciones de sus valores. Todas las combinaciones son factibles.

D) La tabla de decisión completa se muestra en la Tabla 3.

Es fácil ver que las diversas acciones se derivan directamente de las disposiciones de la especificación. Por ejemplo, si el número de puntos del examen teórico es 85 o más, y el candidato ha cometido como máximo dos errores (columna 1), esto significa que se le debe otorgar una licencia de conducir (acción "otorgar una licencia de conducir" = Sí), y el candidato no debe repetir ningún examen ni tomar lecciones adicionales (otras acciones = NO).

E) Los casos de prueba de muestra generados a partir de la tabla de decisiones podrían verse como los siguientes:

Caso de prueba 1 (correspondiente a la columna 1)

Nombre: otorgar la licencia de conducir.

Condiciones previas: el candidato realizó los exámenes por primera vez.

Entrada: puntuación del examen teórico = 85 puntos, número de errores cometidos = 2.

Tabla 3 Tabla de decisión del

sistema de apoyo al examen de conducción

Condiciones

¿Puntos ≥ 85 ?	Sí	Sí	NO	NO
¿Errores ≤ 2 ?	Sí	NO	Sí	NO

Comportamiento

¿Otorgar licencia de conducir?	Sí	NO	NO	NO
¿Repetir el examen teórico?	NO	NO	Sí	Sí
¿Repetir el examen práctico?	NO	Sí	NO	Sí

¿Clases de conducción adicionales?	NO	NO	NO	SI

¿Clases de conducción adicionales?	NO	NO	NO	SI

Resultado esperado: concesión de una licencia de conducir, no es necesario repetir exámenes, no es necesario tomar lecciones de conducción adicionales.

Poscondiciones: candidato marcado como candidato que ya ha realizado los exámenes.

Caso de prueba 2 (correspondiente a la columna 2)

Nombre: aprobar examen teórico, reprobar examen práctico.

Condiciones previas: el candidato realizó los exámenes por primera vez.

Entrada: puntuación del examen teórico = 93 puntos, número de errores cometidos = 3.

Resultado esperado: licencia de conducir no concedida, el candidato debe repetir el examen práctico, no es necesario tomar lecciones de conducción adicionales.

Poscondiciones: candidato marcado como candidato que ya ha realizado los exámenes.

Caso de prueba 3 (correspondiente a la columna 3)

Nombre: reprobar examen teórico, aprobar examen práctico.

Condiciones previas: el candidato realizó los exámenes por primera vez.

Entrada: puntuación del examen teórico = 84 puntos, número de errores cometidos = 0.

Resultado esperado: no se concede el permiso de conducir, el candidato tiene que repetir el examen teórico, no tiene que tomar clases de conducción adicionales.

Poscondiciones: Candidato marcado como candidato que ya realizó los exámenes.

Caso de prueba 4 (correspondiente a la columna 4)

Nombre: reprobar ambos exámenes.

Condiciones previas: el candidato realizó los exámenes por primera vez.

Entrada: puntuación del examen teórico = 42 puntos, número de errores cometidos = 3.

Salida prevista: no se concede el permiso de conducir, el candidato debe repetir pruebas tanto teóricas como prácticas y además debe recibir lecciones de conducción adicionales.

Poscondiciones: candidato marcado como candidato que ya ha realizado los exámenes.

Tenga en cuenta que las condiciones posteriores no están descartadas aquí; tal vez el sistema tenga un conjunto de comportamientos completamente diferente hacia los candidatos que vuelven a realizar el examen.

Por ejemplo, el sistema podría comprobar entonces si el candidato realmente ha recibido lecciones de conducción adicionales, y esto sería parte de la entrada al sistema.

Ejercicio 4.6

(FL-4.2.3, K3)

En la figura 4.19, que describe el proceso, los rombos representan condiciones y los rectángulos representan acciones. Tenemos tres condiciones: "¿Tiene un pasajero una tarjeta dorada?" "¿Está llena la clase económica?" y "¿Está llena la clase ejecutiva?" Las posibles acciones son las siguientes: "¿Emitir una tarjeta de embarque?" "¿Tipo de asiento?" [Económica (E) o Business (B)] y "¿Se elimina al pasajero de la lista de pasajeros?"

La tabla de decisión correspondiente se muestra en la Tabla 4.

Las acciones se asignaron con base en el diagrama de la Fig. 4.19. Tenga en cuenta que la acción "tipo de asiento" no es una variable booleana; sus valores posibles son E, B y N/A ("no aplicable", lo que significa que no se puede asignar ningún asiento porque el pasajero se elimina de la lista de pasajeros).).

De acuerdo con las reglas comerciales, cuando un pasajero tiene una tarjeta dorada y la clase ejecutiva está llena, se le debe asignar un asiento en clase económica. Columnas 1 y

Tabla 4 Tabla de decisiones para el ejercicio 4.6

Condiciones	1	2	3	4	5	6	7	8
Has a gold card?	YES	YES	YES	YES	NO	NO	NO	NO
Economy class full?	YES	YES	NO	NO	YES	YES	NO	NO
Business class full?	YES	NO	YES	NO	YES	NO	YES	NO
Comportamiento								
Issue a boarding pass?	YES	YES	YES	YES	NO	YES	YES	YES
Type of seat	E	B	E	B	N/A	B	E	E
Remove from the passenger list?	NO	NO	NO	NO	YES	NO	NO	NO

3 corresponden a esta situación (tarjeta dorada = SI, clase ejecutiva completa = SI). Miremos detenidamente la columna 1. Describe la situación cuando la clase económica también está llena. Sin embargo, según las especificaciones, el sistema indica al pasajero que asigne un asiento en esta clase (ver celdas con fondo gris).

Hemos descubierto un error grave en la especificación. No sabemos como esto el problema debe resolverse. A continuación se muestran algunas posibles soluciones:

- Sacar a otro pasajero sin una tarjeta dorada de la clase económica y asignar su asiento al cliente en cuestión (la pregunta, sin embargo, es ¿qué pasa si todos los pasajeros de la clase económica tienen una tarjeta dorada? Podría decirse que esta es una situación muy improbable, pero posible: la especificación debe tener esto en cuenta).
- Agregue una condición adicional a la tabla: si el cliente tiene una tarjeta dorada y la clase ejecutiva está llena, considere si la clase económica está llena. De lo contrario, asignamos al pasajero un asiento en esta clase, como se describe en la columna 3. Sin embargo, si la clase económica está llena, entonces debemos tomar otra acción, como eliminar al pasajero de la lista.

Ejercicio 4.7

(FL-4.2.4, K3)

A) En el primer paso, analicemos en qué estados puede estar el sistema. Podemos identificar los siguientes ocho estados (básicamente se derivan directamente del análisis de escenarios):

- Pantalla de bienvenida: estado inicial del sistema, esperando la inserción de la tarjeta.
- Validación de tarjeta: el estado en el que el sistema verifica la tarjeta insertada.
- Fin: el estado al que pasará el sistema después de la validación si la tarjeta es incorrecta.
- Solicitar PIN: el estado en el que el sistema solicita ingresar el PIN por primera vez.
- Solicitar PIN por segunda vez: el estado en el que el sistema solicita ingresar el PIN para la segunda vez (después de ingresar el PIN incorrectamente la primera vez).

Tabla 5 Posibles eventos para cada estado de la máquina de estados

Estado	Posibles eventos y acciones.
Pantalla de bienvenida	Insertar tarjeta
Validación de tarjeta	CardOK (transición al estado Solicitar PIN) InvalidCard (acciones: devolver la tarjeta, mostrar el mensaje "error de tarjeta"; transición al estado Final)
Fin	—
Solicitar PIN	PinOK (transición al estado registrado) PIN no válido (acción: mensaje "Error de PIN"; transición al estado "Solicitar PIN por segunda vez")
Solicitar PIN por segunda vez	PinOK (transición al estado registrado) PIN no válido (acción: mensaje "Error de PIN"; transición al estado "Solicitar PIN por tercera vez")
Solicitar PIN por tercera vez	PinOK (transición al estado registrado) PIN no válido (acciones: mensaje "Error de PIN", mensaje "tarjeta bloqueada"; transición al estado "Tarjeta bloqueada")
registrado	—
Tarjeta bloqueada	—

- Solicitar PIN por tercera vez: estado en el que el sistema solicita ingresar el PIN por tercera vez (después de ingresar el PIN incorrectamente la segunda vez). • Registrado: el estado al que pasa el sistema después de ingresar correctamente el PIN primera, segunda o tercera vez.
- Tarjeta bloqueada: el estado al que pasa el sistema después de ingresar el PIN incorrectamente tres veces.

Tenga en cuenta que en este modelo de transición de estado, necesitamos definir hasta tres estados relacionados con la espera de la entrada del PIN, porque nuestro modelo de estado no tiene memoria: la representación del historial de eventos pasados es solo el estado en el que nos encontramos actualmente., para distinguir la cantidad de códigos PIN ingresados incorrectamente, necesitamos tres estados.

Ahora consideremos los posibles eventos que pueden ocurrir en nuestro sistema y las acciones que el sistema puede tomar para manejar estos eventos. Una de las formas más convenientes de hacer esto es analizar estados individuales y pensar en lo que puede suceder (según la especificación) mientras estamos en un estado determinado. Los resultados de nuestro análisis se presentan en la Tabla 5.

Con base en la Tabla 5, podemos diseñar un diagrama de transición de estado. Se muestra en la Fig. 1. B) El diagrama de transición que utiliza condiciones de guardia se muestra en la Fig. 2. Tenga en cuenta que al introducir condiciones de guardia, podemos reducir el número de estados. Ahora solo tenemos un estado relacionado con la solicitud de un PIN, y la cantidad de PIN ingresados incorrectamente se recuerda en una variable denominada "intentos". Que pasemos del estado "Solicitar PIN" al estado "Tarjeta bloqueada" o "Registrado" depende de cuántas veces se ingresó el PIN incorrecto. Tenga en cuenta que el bucle

Solicitar PIN (PIN no válido) Solicitar PIN

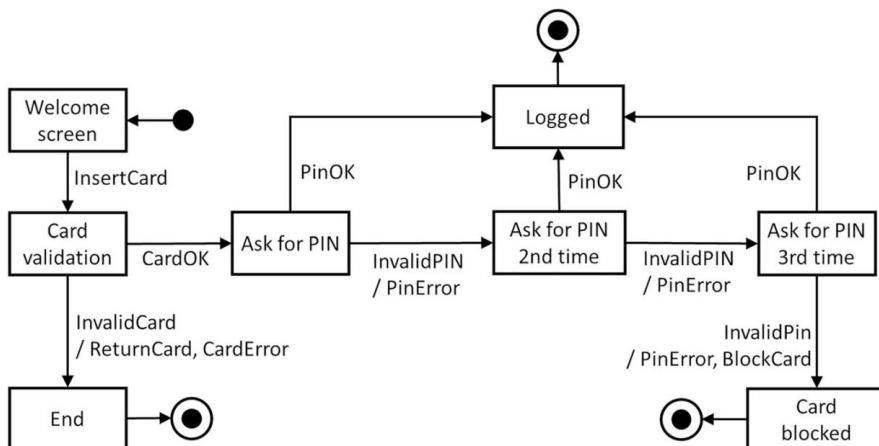


Fig. 1 Diagrama de transición para la verificación del PIN sin utilizar condiciones de protección

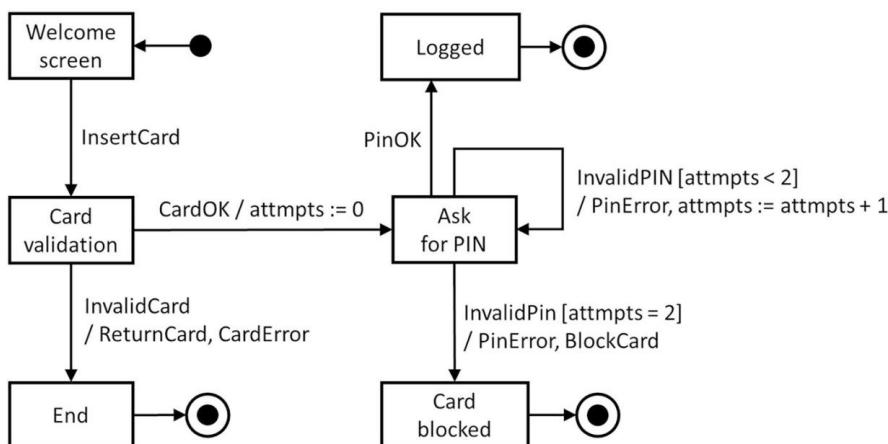


Fig. 2 Diagrama de transición para la verificación del PIN usando condiciones de guardia

Puede ejecutarse un máximo de dos veces. Esto se debe a que cada vez que ejecutamos esta transición de bucle, el valor de la variable "intento" aumenta en uno, y la condición de guardia nos permite ejercer este bucle sólo si esta variable tiene un valor menor que 2. Después de ingresar el PIN incorrectamente dos veces, esta variable tiene un valor de 2, y al momento del tercer intento fallido de ingresar el PIN, la condición de guardia es falsa.

En cambio, la condición de guardia en la transición a "Tarjeta bloqueada" se vuelve verdadera, por lo que después del tercer intento fallido de ingresar el PIN, el sistema no permanece en "Solicitar PIN" nuevamente sino que pasa al estado "Tarjeta bloqueada".

C) Antes de pasar al diseño de prueba, consideremos qué condiciones de prueba tenemos para ambos tipos de cobertura. En el caso de la cobertura de todos los estados, tenemos los siguientes elementos (estados) a cubrir: Pantalla de bienvenida, Validación de tarjeta, Fin, Solicitar PIN, Solicitar PIN por segunda vez, Solicitar PIN por tercera vez, Registrado y Tarjeta bloqueada. Por tanto, necesitamos cubrir ocho elementos de cobertura con pruebas. Para una cobertura de transiciones válida, debemos cubrir todas las flechas entre estados. Hay nueve: InsertCard, InvalidCard, CardOK, tres transiciones diferentes activadas por PinOK y tres transiciones diferentes activadas por InvalidPIN. Entonces, para lograr una cobertura de transiciones válida y completa, debemos cubrir nueve elementos de cobertura con nuestros casos de prueba.

Con respecto a la cobertura de todos los estados, observemos que en el diagrama de la Fig. 1, tenemos tres estados finales. Por lo tanto, necesitaremos al menos tres casos de prueba, ya que alcanzar el estado final finaliza la ejecución de la prueba. A continuación se muestran tres casos de prueba que logran una cobertura total en todos los estados:

TC1: Pantalla de bienvenida (InsertCard) Validación de tarjeta (InvalidCard) Fin

TC2: Pantalla de bienvenida (InsertCard) Validación de tarjeta (CardOK) Solicitar PIN (PinOK)

registrado

TC3: Pantalla de bienvenida (InsertCard) Validación de tarjeta (CardOK) Solicitar PIN

(PIN inválido) Solicitar PIN por segunda vez (PIN inválido) Solicitar PIN por tercera vez

(PinWrong) Tarjeta bloqueada

Estos tres casos cubren todos los estados (TC1 cubre tres de ellos, TC2 cubre dos adicionales y TC3 cubre tres adicionales), pero no cubren todas las transiciones.

Las dos transiciones no cubiertas son:

Solicitar PIN por segunda vez (PinOK) Registrado

Solicitar PIN por tercera vez (PinOK) Registrado

Necesitamos agregar dos nuevas pruebas que cubran estas dos transiciones, como por ejemplo:

PT4: Pantalla de bienvenida (InsertCard) Validación de tarjeta (CardOK) Solicitar PIN

(PIN no válido) Solicitar PIN por segunda vez (PinOK) Registrado

PT5: Pantalla de bienvenida (InsertCard) Validación de tarjeta (CardOK) Solicitar PIN

(PIN inválido) Solicitar PIN por segunda vez (PIN inválido) Solicitar PIN por tercera vez

(PinOK) Registrado

Nuevamente, tenga en cuenta que no podemos lograr una cobertura de transiciones válida con menos de cinco casos de prueba, porque tenemos cinco transiciones que llegan directamente a los estados finales. Esto significa que la ejecución de dicha transición finaliza el caso de prueba, por lo que dos de estas cinco transiciones no pueden estar dentro de un solo caso de prueba.

Ejercicio 4.8

(FL-4.2.4, K3)

A) El sistema tiene tres estados (S, W, B) y cinco eventos diferentes (¡Silencio!, ¡Ladra!, ¡Abajo!, Ve al gato, Está acariciado). Entonces tenemos $3 \times 5 = 15$ combinaciones de pares (estado, transición). Como podemos ver en el diagrama que hay seis transiciones válidas (seis flechas entre estados), debe haber $15 - 6 = 9$ transiciones no válidas:

- (S, ¡Silencio!) •
- (S, ¡Abajo!) • (B,
¡Ladra!) • (B,
¡Abajo!) • (B,
SeesCat) • (W,
¡Silencio!) • (W,
¡Ladra!) • (W, es
acariciado) • (W, ve
gato)

B) Las transiciones válidas se pueden probar usando un caso de prueba, por ejemplo:

S (¡Ladra!) B (¡Silencio!) S (Ve al gato) B (Me acarician) W (¡Abajo!) S (Me acarician) W

Este caso de prueba cubre las seis transiciones válidas. Como tenemos nueve transiciones no válidas, necesitamos agregar nueve casos de prueba, uno para cada transición no válida. Por ejemplo, para cubrir la transición no válida (W, Bark!), podemos diseñar el caso:

S (IsPetted) W (¡Ladra!)?

Después de alcanzar el estado W, intentamos activar el (inválido) "¡Ladrido!" evento. Si no es factible o si el sistema lo ignora, asumimos que la prueba pasa. Si, por el contrario, el mensaje "¡Ladra!" Se puede invocar el evento y el sistema cambia de estado, la prueba falla, ya que este no es el comportamiento esperado.

Tenga en cuenta que siempre comenzamos en el estado inicial, por lo que primero debemos ejercitarn las transiciones válidas para alcanzar el estado deseado (en nuestro caso, el estado W) y luego intentar activar la transición no válida (en nuestro caso, ¡Bark!). De manera similar, diseñamos casos de prueba para las ocho transiciones no válidas restantes.

Ejercicio 4.9

(FL-4.5.3, K3)

A continuación se muestran algunos ejemplos de casos de prueba que podemos diseñar:

- Registro exitoso con inicio de sesión válido (aún no utilizado en el sistema) y contraseña válida, por ejemplo, inicio de sesión john.smith@mymail.com, contraseña Abc123Def escrita dos veces en ambos campos, resultado esperado: el sistema acepta los datos y envía un correo electrónico a john.smith@mymail.com con un enlace para activar la cuenta (esta prueba verifica los criterios de aceptación AC1 y AC5).
- Intento de registro con inicio de sesión sintácticamente incorrecto (cubre los criterios de aceptación).
terión AC1). Conjuntos de datos de prueba:
 - John.Smith@post (parte demasiado corta después del signo @)
 - JohnSmith-mymail.com (sin signo @) –
 - @mymail.com (sin texto antes del signo @)
 - JohnSmith@mymail..com (dos puntos consecutivos después del signo @)

- Intente registrarse con un inicio de sesión válido pero existente. Resultado esperado: registro denegado, no se envió ningún correo electrónico (esta prueba verifica el criterio de aceptación AC2). •

Intente registrarse con un inicio de sesión válido, pero una contraseña incorrecta. Resultado esperado: registro denegado, no se envió ningún correo electrónico. Conjuntos de datos de prueba (que cubren los criterios de aceptación AC3 y AC4):

- Contraseña sintácticamente correcta, pero diferente en ambos campos de la contraseña (por ejemplo, Abc123Def y aBc123Def)
- Contraseña demasiado corta (p. ej., Ab12)
- Contraseña demasiado larga (por ejemplo, ABCD1234abcd1234)
- Contraseña sin dígitos (por ejemplo, ABCdef)
- Contraseña sin letras mayúsculas (por ejemplo, abc123)

Soluciones a los ejercicios del cap. 5

Ejercicio 5.1

(FL-5.1.4, K3)

La última iteración del póquer para el valor optimista (a) arrojó valores de 3, 3 y 5, lo que, según el procedimiento descrito, significa que los expertos llegaron a un consenso sobre el valor optimista. Es 3, ya que es el valor indicado por la mayoría de expertos.

De manera similar, para el valor más probable (m), los expertos llegaron a un consenso después de la primera iteración. El resultado es 5, ya que dos de los tres expertos indicaron este valor.

Para el valor pesimista (b), los expertos llegaron a un consenso sólo en la tercera iteración. El resultado es 13, como indicaron todos los expertos.

Después de estas sesiones de póquer, los expertos determinaron los valores de las variables a ser utilizado en la técnica de estimación de tres puntos, a saber:

- Valor optimista: $a = 3$ • Valor más probable: $m = 5$ • Valor pesimista: $b = 13$

Sustituyendo estos valores en la fórmula del método de los tres puntos, obtenemos:

$$mi = \frac{a + b - m}{3} = \frac{3 + 13 - 5}{3} = 6$$

lo que significa que el esfuerzo estimado es de 6 días-persona, con una desviación estándar de

$$DE = \frac{b - a}{6} = \frac{13 - 3}{6} = \frac{10}{6} = 1.66$$

Esto significa que el resultado final de la estimación es de 6 ± 1.66 días-persona, que es entre 4,33 y 7,66 días-persona.

Ejercicio 5.2

(FL-5.1.4, K3)

En el proyecto terminado, el esfuerzo para la fase de diseño fue de 20 días-persona (porque 4 personas hicieron el trabajo en 5 días). De manera similar, para la fase de implementación, el esfuerzo fue de $10 \times 18 = 180$ días-persona y para las pruebas de $4 \times 10 = 40$ días-persona.

Así, el esfuerzo total fue de 240 días-persona en una relación diseño/programación/pruebas igual a 1:9:2.

Sea x = número de días de trabajo de los diseñadores e y = número de días de trabajo de los desarrolladores. Entonces $66 - x - y$ es el número de días laborales de los probadores (porque se espera que el proyecto total dure 66 días). Dado el número de diseñadores, desarrolladores y probadores en el nuevo proyecto, el esfuerzo en las fases de diseño, implementación y prueba es, por tanto, $4x$, $6y$, $2(66-x-y)$, respectivamente.

De la relación, tenemos $4x : 6y : 2(66-x-y) = 1:9:2$. Entonces, tenemos $2^*4*x = 2^*(66-x-y)$ y $9^*4*x = 6^*y$, ya que las pruebas requieren el doble de esfuerzo que el diseño y la implementación requiere nueve veces más esfuerzo que la planificación. Ahora debemos resolver el sistema de dos ecuaciones:

$$8x = 132 - 2x - 2y$$

$$36x = 6y$$

De la segunda ecuación, tenemos $y = 6x$. Sustituyendo en la primera ecuación obtenemos:

$$8x = 132 - 2x - 12x, \text{ o } 22x = 132, \text{ de donde calculamos } x = 132/22 = 6:$$

Al sustituir $x = 6$ en la relación $y = 6x$, obtenemos que $y = 6^*6 = 36$.

Dado que x , y y $66-xy$ representan el número de días dedicados al diseño, la programación y las pruebas, respectivamente, obtenemos la respuesta final: utilizando el método de la relación, es decir, suponiendo que el esfuerzo en el nuevo proyecto aumentará. distribuirse proporcionalmente al esfuerzo en el proyecto finalizado:

- Debemos asignar $x = 6$ días para el diseño. •
- Debemos asignar $y = 36$ días para el desarrollo. • Debemos asignar $66-x-y = 66 - 42 = 24$ días para las pruebas.

Ejercicio 5.3

(FL-5.1.5, K3)

Si solo consideráramos las prioridades, el orden de ejecución de los casos de prueba sería el siguiente:

TC001 → TC003 → TC002 → TC004.

Sin embargo, también debemos tener en cuenta las dependencias lógicas: no podemos ejecutar TC001 antes de TC002 y TC003.

Las dependencias lógicas nos obligan a empezar por TC002 (entrada de datos personales), porque por sí solo no depende de ningún otro caso. Todavía no podemos ejecutar TC001 con el

máxima prioridad, porque depende no sólo de 002 sino también de 003. Entonces tenemos que ejecutar el caso 003 en el siguiente paso, lo que desbloqueará la posibilidad de ejecutar TC001. Al final, ejecutamos TC004.

Así, el orden final es: TC002 → TC003 → TC001 → TC004.

Ejercicio 5.4

(FL-5.1.5, K3)

Si tuviéramos que confiar únicamente en las prioridades del cliente, deberíamos implementar y probar los requisitos Req2, Req3 y Req6 primero (prioridad alta), luego Req1 y Req5 (prioridad media) y finalmente Req4 (prioridad baja). Sin embargo, independientemente de las prioridades, el primer requisito a implementar y probar debe ser Req1, ya que es el único requisito que no depende de otros requisitos. El único requisito que se puede implementar y probar a continuación es el Req3 (porque es el único que depende únicamente del Req1 ya implementado y probado). Con Req1 y Req3 probados, tenemos que implementar y probar Req2, ya que es el único requisito que depende de los requisitos ya implementados y probados.

Tenga en cuenta que hasta este momento las prioridades no han influido. Pero ahora podemos implementar y probar Req4 o Req5. Tengamos en cuenta que entre los requisitos aún no implementados ni probados, el requisito con mayor prioridad para el cliente es Req6 y depende de Req4. Por lo tanto, primero implementamos y probamos Req4, para "desbloquear" la posibilidad de implementar y probar el Req6 de alta prioridad lo antes posible. Al final, implementamos y probamos Req5.

El orden final de requisitos es el siguiente:

Solicitud1, Solicitud3, Solicitud2, Solicitud4, Solicitud6, Solicitud5.

El pedido tiene en cuenta las prioridades del cliente y al mismo tiempo tiene en cuenta las relaciones lógicas necesarias entre los requisitos.

Ejercicio 5.5

(FL-5.5.1, K3)

El informe de defectos debe contener al menos la información descrita en la Tabla 6.

Tabla 6 Contenido del informe de defectos

Identificador único	34.810
Título, resumen del defecto reportado	Cálculo incorrecto del importe a pagar 09.07.2023
Fecha del informe (= fecha de descubrimiento del error)	
Autor	carol cerveza
Descripción del defecto para permitir su reproducción y reparación.	Caso de prueba PT003
Resultado actual	\$48
Resultado Esperado	\$47.50
Prioridad para la eliminación de defectos	Normal
Una descripción de la no conformidad para ayudar a determinar su causa.	Parece que el sistema redondea el importe resultante al total de \$

Parte IV

Examen oficial de muestra

Esta parte del manual contiene un examen de muestra. Este es el examen de muestra oficial publicado por ISTQB®. La duración del examen es de 60 minutos, o 75 minutos si el examen no se realiza en el idioma nativo.

El documento de examen de muestra original de ISTQB® contiene las preguntas del examen en el orden de los objetivos de aprendizaje correspondientes en el programa de estudios. Esto puede hacer que sea más fácil responder algunas de las preguntas. En el examen real, las preguntas están dispuestas en orden aleatorio. Por lo tanto, en este capítulo, las preguntas de muestra del examen se colocan en orden aleatorio para reflejar mejor la realidad del examen.

Conjunto de exámenes A



Pregunta #1 (1 punto)

Estás probando un sistema que calcula la calificación final del curso para un estudiante determinado.

La calificación final se asigna en función del resultado final, según las siguientes reglas:

- 0–50 puntos: reprobado.
- 51–60 puntos: regular.
- 61–70 puntos: satisfactorio.
- 71–80 puntos: bueno.
- 81–90 puntos: muy bueno.
- 91–100 puntos: excelente.

Ha preparado el siguiente conjunto de casos de prueba:

	Resultado final	Evaluación final
TC1	91	Excelente
TC2	50	Fallido
TC3	81	Muy bien
TC4	60	Justo
TC5	70	Satisfactorio
TC6	80	Bien

¿Cuál es la cobertura del análisis de valor límite (BVA) de 2 valores para el resultado final?

¿Qué se logra con los casos de prueba existentes?

- a) 50%
- segundo) 60%
- c) 33,3%
- d) 100%

Seleccione UNA opción.

Pregunta #2 (1 punto)

¿Cuál de los siguientes es un beneficio de la retroalimentación temprana y frecuente?

- a) Mejora el proceso de prueba para futuros proyectos.
- b) Obliga a los clientes a priorizar sus requisitos en función de los riesgos acordados.
- c) Es la única manera de medir la calidad de los cambios.
- d) Ayuda a evitar malentendidos sobre los requisitos.

Seleccione UNA opción.

Pregunta #3 (1 punto)

Tu tienda favorita de alquiler diario de bicicletas acaba de presentar un nuevo sistema de gestión de relaciones con el cliente y te ha pedido a ti, uno de sus miembros más leales, que lo pruebes.

é'l. Las características implementadas son las siguientes:

- Cualquier persona puede alquilar una bicicleta, pero los socios reciben un 20% de descuento.
- Sin embargo, si no se cumple el plazo de devolución, el descuento ya no estará disponible.
- Despues de 15 alquileres, los miembros reciben un regalo: una camiseta.

La tabla de decisiones que describe las características implementadas es la siguiente:

	R1	R2	R3	R4	R5	R6	R7	R8
Condiciones								
ser miembro	t	t	t	t	F	F	F	F
Plazo incumplido	t	F	t	F	t	F	F	t
alquiler 15	F	F	t	t	F	F	t	t
Comportamiento								
20% de descuento		X		X				
Camiseta de regalo			X	X				X

Basado ÚNICAMENTE en la descripción de la función de Gestión de relaciones con el cliente. sistema de gestión, ¿cuál de las reglas anteriores describe una situación imposible?

- a) R4
- segundo) R2
- c) R6
- d) R8

Elija UNA respuesta.

Pregunta #4 (1 punto)

Debe actualizar uno de los scripts de prueba automatizados para que esté en línea con un nuevo requisito. ¿Qué proceso indica que crea una nueva versión del script de prueba? en el repositorio de pruebas?

- a) Gestión de la trazabilidad.
- b) Pruebas de mantenimiento.
- c) Gestión de la configuración.
- d) Ingeniería de requisitos.

Seleccione UNA opción.

Pregunta #5 (1 punto)

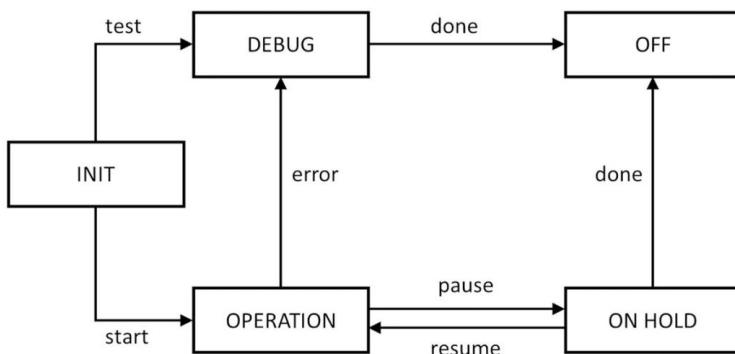
¿Cuál de las siguientes es una característica de las técnicas de prueba basadas en la experiencia?

- a) Los casos de prueba se crean basándose en información de diseño detallada.
- b) Los elementos probados dentro de la sección del código de interfaz se utilizan para medir la cobertura.
- c) Las técnicas dependen en gran medida del conocimiento del software y del dominio empresarial del evaluador.
- d) Los casos de prueba se utilizan para identificar desviaciones de los requisitos.

Seleccione UNA opción.

Pregunta #6 (1 punto)

Usted prueba un sistema cuyo ciclo de vida está modelado por el diagrama de transición de estado que se muestra a continuación. El sistema arranca en el estado INIT y finaliza su funcionamiento en el estado OFF.



¿Cuál es el número MÍNIMO de casos de prueba para lograr una cobertura de transiciones válida?

- a) 4 b)
- 2 c) 7
- d) 3

Seleccione UNA opción.

Pregunta #7 (1 punto)

Recibió el siguiente informe de defectos de los desarrolladores que indican que la anomalía descrita en este informe de prueba no es reproducible: La aplicación se cuelga el 3 de

mayo de 2022: John Doe:

rechazada La aplicación se cuelga después de

ingresar "Entrada de prueba: \$á" en el Campo de nombre en la pantalla de creación de nuevo usuario.

Intenté cerrar sesión, iniciar sesión con la cuenta test_admin01, el mismo problema. Probé con otras cuentas de administrador de prueba, el mismo problema. No se recibió ningún mensaje de error; log (ver adjunto) contiene una notificación de error fatal. Segundo el caso de prueba TC-1305, la aplicación debe aceptar la entrada proporcionada y crear el usuario. Por favor arregla con

De alta prioridad, esta característica está relacionada con REQ-0012, que es un nuevo requisito comercial crítico.

¿Qué información crítica FALTA en este informe de prueba que habría sido útil para los desarrolladores?

- a) Resultado esperado y resultado real. b)
Referencias y estado de los defectos. c)
Entorno de prueba y elemento de prueba. d)
Prioridad y severidad.

Seleccione UNA opción.

Pregunta #8 (1 punto)

¿Cómo añaden los evaluadores valor a la planificación de iteraciones y lanzamientos?

- a) Los testers determinan la prioridad de las historias de usuario a desarrollar. b) Los evaluadores se centran únicamente en los aspectos funcionales del sistema que se va a probar. c)
Los evaluadores participan en la identificación detallada de riesgos y la evaluación de riesgos del usuario. cuentos.
- d) Los evaluadores garantizan el lanzamiento de software de alta calidad mediante un diseño de prueba temprano durante la planificación del lanzamiento.

Seleccione UNA opción.

Pregunta #9 (1 punto)

Trabajas en un equipo que desarrolla una aplicación móvil para pedidos de comida. En la versión actual, el equipo decidió implementar la funcionalidad de pago.

¿Cuál de las siguientes actividades forma parte del análisis de pruebas?

- a) Estimar que probar la integración con el servicio de pago llevará 8 días-persona.
- b) Decidir que el equipo debería probar si es posible compartir adecuadamente el pago entre muchos usuarios. c)
Utilizar el análisis de valor límite (BVA) para derivar los datos de prueba para los casos de prueba que verifican el procesamiento correcto del pago para el monto mínimo permitido a pagar.
- d) Analizar la discrepancia entre el resultado real y el resultado esperado luego de ejecutar un caso de prueba que verifica el proceso de pago con tarjeta de crédito y reportar un defecto.

Seleccione UNA opción.

Pregunta #10 (1 punto)

¿Qué herramienta puede utilizar un equipo ágil para mostrar la cantidad de trabajo que se ha completado y la cantidad de trabajo total restante para una iteración determinada?

- a) Criterios de aceptación. b)
Informe de defectos. c)
Informe de finalización de la prueba.
- d) Cuadro de evolución.

Seleccione UNA opción.

Pregunta #11 (1 punto)

¿Cuál de las siguientes afirmaciones describe MEJOR el enfoque de desarrollo basado en pruebas de aceptación (ATDD)?

- a) En ATDD, los criterios de aceptación generalmente se crean en función del dado/cuándo/entonces formato.
- b) En ATDD, los casos de prueba se crean principalmente en las pruebas de componentes y se codifican orientado.
- c) En ATDD se crean pruebas, basadas en criterios de aceptación para impulsar el desarrollo. ment del software relacionado.
- d) En ATDD las pruebas se basan en el comportamiento deseado del software, lo que hace será más fácil para los miembros del equipo entenderlos.

Seleccione UNA opción.

Pregunta #12 (1 punto)

¿Qué elemento identifica correctamente un riesgo potencial al realizar la automatización de pruebas?

- a) Puede introducir regresiones desconocidas en la producción. b) Es posible que no se asignen adecuadamente los esfuerzos suficientes para mantener el software de prueba. c) Es posible que no se confíe suficientemente en las herramientas de prueba y el software de prueba asociado. d) Podrá reducir el tiempo destinado a las pruebas manuales.

Seleccione UNA opción.

Pregunta #13 (1 punto)

¿Cuáles DOS de las siguientes opciones son los criterios de salida para probar un sistema?

- a) Preparación del entorno de prueba. b) La capacidad del probador de iniciar sesión en el objeto de prueba. c) Se alcanza la densidad de defectos estimada. d) Los requisitos se traducen al formato dado/cuándo/entonces. e) Las pruebas de regresión están automatizadas.

Seleccione DOS opciones.

Pregunta #14 (1 punto)

¿Cuál de los siguientes NO es un beneficio de las pruebas estáticas?

- a) Tener una gestión de defectos menos costosa debido a la facilidad de detectar defectos más adelante en el SDLC.
- b) La reparación de defectos encontrados durante las pruebas estáticas es generalmente mucho menos costosa que corregir defectos encontrados durante las pruebas dinámicas.
- c) Encontrar defectos de codificación que podrían no haberse encontrado realizando únicamente pruebas dinámicas.
- d) Detectar lagunas e inconsistencias en los requisitos.

Seleccione UNA opción.

Pregunta #15 (1 punto)

¿Cuáles de las siguientes habilidades (i–v) son las MÁS importantes de un evaluador?

- i. Tener conocimiento del dominio. ii.

Creando una visión del producto. III.

Ser un buen jugador de equipo. IV.

Planificar y organizar el trabajo del equipo. v. Pensamiento crítico.

a) ii y iv son importantes; i, iii y v no lo son. b) i, iii y v son

importantes; ii y iv no lo son. c) i, ii y v son importantes; iii y

iv no lo son. d) iii y iv son importantes; i, ii y v no lo son.

Seleccione UNA opción.

Pregunta #16 (1 punto)

¿Cuál de los siguientes argumentos utilizaría para convencer a su gerente de organizar retrospectivas al final de cada ciclo de lanzamiento?

a) Las retrospectivas son muy populares hoy en día y los clientes agradecerían que las
Los agregamos a nuestros procesos.

b) La organización de retrospectivas le ahorrará dinero a la organización porque los representantes de los
usuarios finales no brindan comentarios inmediatos sobre el producto. c) Las debilidades
de los procesos identificadas durante la retrospectiva se pueden analizar y servir como una lista de tareas
pendientes para la mejora continua de los procesos de la organización.
programa.

d) Las retrospectivas abrazan cinco valores, incluido el coraje y el respeto, que son cruciales para mantener
la mejora continua en la organización.

Seleccione UNA opción.

Pregunta #17 (1 punto)

En su proyecto, hubo un retraso en el lanzamiento de una aplicación nueva y la ejecución de la prueba
comenzó tarde, pero usted tiene un conocimiento muy detallado del dominio y buenas habilidades analíticas.
La lista completa de requisitos aún no se ha compartido con el equipo, pero la gerencia solicita que se
presenten algunos resultados de las pruebas.

¿Qué técnica de prueba se adapta MEJOR a esta situación?

a) Pruebas basadas en listas de
verificación. b) Error al
adivinar. c) Pruebas
exploratorias. d) Pruebas de sucursales.

Seleccione UNA opción.

Pregunta #18 (1 punto)

Su equipo utiliza la técnica de estimación de tres puntos para estimar el esfuerzo de prueba de una nueva característica de alto riesgo. Se hicieron las siguientes estimaciones:

- Estimación más optimista: 2 horas-persona.
- Estimación más probable: 11 horas-persona.
- Estimación más pesimista: 14 horas-persona.

¿Cuál es la estimación final?

- a) 9 horas-persona. b)
14 horas-persona. c) 11
horas-persona. d) 10
horas-persona.

Seleccione UNA opción.

Pregunta #19 (1 punto)

¿Cuál de las siguientes afirmaciones NO es cierta para las pruebas de caja blanca?

- a) Durante las pruebas de caja blanca, se considera toda la implementación del software. b) Las métricas de cobertura de caja blanca pueden ayudar a identificar pruebas adicionales para aumentar el código de cobertura.
c) Las técnicas de prueba de caja blanca se pueden utilizar en pruebas estáticas. d) Las pruebas de caja blanca pueden ayudar a identificar brechas en la implementación de requisitos.

Seleccione UNA opción.

Pregunta #20 (1 punto)

Las revisiones que se utilizan en su organización tienen los siguientes atributos:

- Existe el papel de escribano.
- El objetivo principal es evaluar la calidad.
- La reunión es liderada por el autor del producto del trabajo.
- Hay preparación individual.
- Se elabora un informe de revisión.

¿Cuál de los siguientes tipos de revisión es MÁS probable que se utilice?

- a) Revisión informal. b)
Tutorial. c) Revisión
técnica. d) Inspección.

Seleccione UNA opción.

Pregunta #21 (1 punto)

Estás probando un formulario de búsqueda de apartamento simplificado que sólo tiene dos criterios de búsqueda:

- Planta (con tres opciones posibles: planta baja; primer piso; segundo o superior piso).
- Tipo de jardín (con tres opciones posibles: sin jardín; jardín pequeño; jardín grande).

Sólo los apartamentos de la planta baja podrán tener jardín. El formulario tiene un mecanismo de validación incorporado que no le permitirá utilizar criterios de búsqueda que violen esta regla.

Cada prueba tiene dos valores de entrada: tipo de piso y jardín. Desea aplicar partición de equivalencia (EP) para cubrir cada piso y cada tipo de jardín en sus pruebas.

¿Cuál es el número MÍNIMO de casos de prueba para lograr una cobertura de EP del 100%?

- a) 3b)
- 4c) 5d)
- 6

Seleccione UNA opción.

Pregunta #22 (1 punto)

¿Cuál de los siguientes NO es un ejemplo del enfoque de desplazamiento a la izquierda?

- a) Revisar los requisitos de los usuarios antes de que sean aceptados formalmente por la partes interesadas.
- b) Escribir una prueba de componente antes de escribir el código correspondiente. c) Ejecutar una prueba de eficiencia de desempeño para un componente durante el componente pruebas.
- d) Escribir un script de prueba antes de configurar el proceso de gestión de la configuración.

Seleccione UNA opción.

Pregunta #23 (1 punto)

¿Qué actividad de prueba admite una herramienta de preparación de datos?

- a) Seguimiento y control de las pruebas. b)
- Análisis y diseño de pruebas. c)
- Implementación y ejecución de pruebas. d) Finalización de la prueba.

Seleccione UNA opción.

Pregunta #24 (1 punto)

Su conjunto de pruebas logró una cobertura de declaraciones del 100 %. ¿Cuál es la consecuencia de este hecho?

- a) Cada instrucción del código que contiene un defecto ha sido ejecutada en menos una vez.
- b) Cualquier conjunto de pruebas que contenga más casos de prueba que su conjunto de pruebas también logrará Cobertura del 100% del estado de cuenta.
- c) Cada ruta del código se ha ejecutado al menos una vez.
- d) Cada combinación de valores de entrada ha sido probada al menos una vez.

Seleccione UNA opción.

Pregunta #25 (1 punto)

Considera la siguiente regla: "para cada actividad del SDLC hay una actividad de prueba correspondiente". ¿En qué modelos SDLC se aplica esta regla?

- a) Sólo en modelos SDLC secuenciales.
- b) Sólo en modelos SDLC iterativos.
- c) Sólo en modelos SDLC iterativos e incrementales.
- d) En modelos SDLC secuenciales, incrementales e iterativos.

Seleccione UNA opción.

Pregunta #26 (1 punto)

Considera las siguientes categorías de pruebas (1 a 4) y los cuadrantes de pruebas ágiles (A a D):

1. Pruebas de usabilidad.
 2. Pruebas de componentes.
 3. Pruebas funcionales.
 4. Pruebas de confiabilidad.
- A. Cuadrante de pruebas ágiles Q1: frente a la tecnología, apoyando al equipo de desarrollo.
B. Cuadrante de pruebas ágiles Q2: orientación empresarial, apoyo al equipo de desarrollo.
C. Cuadrante de pruebas ágiles Q3: de cara al negocio, critica el producto.
D. Cuadrante de pruebas ágiles Q4: frente a la tecnología, criticar el producto.

¿Cómo se asignan las siguientes categorías de pruebas a los cuadrantes de pruebas ágiles?

- a) 1C, 2A, 3B, 4D.
- b) 1D, 2A, 3C, 4B.
- c) 1C, 2B, 3D, 4A.
- d) 1D, 2B, 3C, 4A.

Seleccione UNA opción.

Pregunta #27 (1 punto)

¿Qué tipos de fallas (1 a 4) se ajustan MEJOR a qué niveles de prueba (A a D)?

1. Fallas en el comportamiento del sistema al desviarse de las necesidades comerciales del usuario.
 2. Fallos en la comunicación entre componentes.
 3. Fallos en la lógica de un módulo.
 4. Fallas en reglas de negocio no implementadas correctamente.
- A. Prueba de componentes.
B. Pruebas de integración de componentes.
C. Pruebas del sistema.
D. Pruebas de aceptación.
- a) 1D, 2B, 3A, 4C. b) 1D,
2B, 3C, 4A. c) 1B, 2A, 3D,
4C. d) 1C, 2B, 3A, 4D.

Seleccione UNA opción.

Pregunta #28 (1 punto)

¿Cuál de las siguientes opciones describe MEJOR la forma en que se pueden documentar los criterios de aceptación?

- a) Realizar retrospectivas para determinar las necesidades reales de las partes interesadas.
con respecto a una historia de usuario determinada.
- b) Usar el formato dado/cuándo/entonces para describir una condición de prueba de ejemplo relacionada con una historia de usuario
- determinada. c) Utilizar la comunicación verbal para reducir el riesgo de que otros malinterpreten los criterios de aceptación. d) Documentar los riesgos relacionados con una historia de usuario determinada en un plan de prueba para facilitar las pruebas basadas en riesgos de una historia de usuario determinada.

Seleccione UNA opción.

Pregunta #29 (1 punto)

¿Cuál de las siguientes opciones describe MEJOR el concepto detrás de la adivinación errónea?

- a) La adivinación de errores implica utilizar su conocimiento y experiencia sobre defectos encontrados en el pasado y errores típicos cometidos por los desarrolladores. b)
- Adivinar errores implica utilizar su experiencia personal de desarrollo y los errores que cometió como desarrollador.
- c) La adivinación de errores requiere que usted imagine que es el usuario del objeto de prueba y para adivinar errores que el usuario podría cometer al interactuar con él.
- d) La adivinación de errores requiere que usted duplique rápidamente la tarea de desarrollo para identificar el tipo de errores que podría cometer un desarrollador.

Seleccione UNA opción.

Pregunta #30 (1 punto)

¿Cuáles DOS de las siguientes tareas pertenecen PRINCIPALMENTE a una función de prueba?

- a) Configurar entornos de prueba.
- b) Mantener la cartera de productos.
- c) Diseñar soluciones a nuevos requerimientos.
- d) Crear el plan de pruebas.
- e) Informe sobre la cobertura alcanzada.

Seleccione DOS opciones.

Pregunta #31 (1 punto)

Están probando una aplicación móvil que permite a los usuarios encontrar un restaurante cercano según el tipo de comida que quieran comer. Considere la siguiente lista de casos de prueba, prioridades (es decir, un número menor significa una prioridad más alta) y dependencias:

Número de caso de prueba	Condición de prueba cubierta	Prioridad	Dependencia lógica
TC 001	Seleccionar tipo de comida	3	Ninguno
TC 002	Seleccionar restaurante	2	TC 001
TC 003	Obtener dirección	1	TC 002
TC 004	Llamar al restaurante	2	TC 002
TC 005	Hacer reservacion	3	TC 002

¿Cuál de los siguientes casos de prueba debería ejecutarse como el tercero?

- a) TC 003.
- b) TC 005.
- c) TC 002.
- d) TC 001.

Seleccione UNA opción.

Pregunta #32 (1 punto)

Se le ha asignado como probador de un equipo que produce un nuevo sistema de forma incremental.

Ha notado que no se han realizado cambios en la prueba de regresión existente.

casos durante varias iteraciones y no se identificaron nuevos defectos de regresión. Su

El gerente está contento, pero tú no. ¿Qué principio de prueba explica su escepticismo?

- a) Las pruebas se desgastan.
- b) Falacia de ausencia de errores.
- c) Los defectos se agrupan.
- d) Es imposible realizar pruebas exhaustivas.

Seleccione UNA opción.

Pregunta #33 (1 punto)

Durante un análisis de riesgos, se identificaron y evaluaron los siguientes riesgos:

- Riesgo: el tiempo de respuesta es demasiado largo para generar un informe.
- Probabilidad de riesgo, media; impacto del riesgo, alto.
- Respuesta al riesgo:
 - Un equipo de pruebas independiente realiza pruebas de rendimiento durante las pruebas del sistema.
 - Una muestra seleccionada de usuarios finales realiza pruebas de aceptación alfa y beta antes del lanzamiento.

¿Qué medida se propone tomar frente a este riesgo analizado?

- a) Aceptación del riesgo.
- b) Plan de contingencia. c)
- Mitigación de riesgos. d)
- Transferencia de riesgos.

Seleccione UNA opción.

Pregunta #34 (1 punto)

Considere la siguiente historia de usuario:

Como editor

Quiero revisar el contenido antes de su publicación para poder asegurarme de que la gramática sea correcta.

y sus criterios de aceptación:

- El usuario puede iniciar sesión en el sistema de gestión de contenidos con el rol de "Editor". • El editor puede ver páginas de contenido existentes.
- El editor puede editar el contenido de la página.
- El editor puede agregar comentarios de marcado.
- El editor puede guardar los cambios.
- El editor puede reasignarlo al rol de "propietario del contenido" para realizar actualizaciones.

¿Cuál de los siguientes es el MEJOR ejemplo de una prueba ATDD para esta historia de usuario?

- a) Pruebe si el editor puede guardar el documento después de eliminar el contenido de la página.
- b) Probar si el propietario del contenido puede iniciar sesión y realizar actualizaciones del contenido.
- c) Probar si el editor puede programar la publicación del contenido editado.
- d) Pruebe si el editor puede reasignarse a otro editor para realizar actualizaciones.

Seleccione UNA opción.

Pregunta #35 (1 punto)

Está probando una historia de usuario con tres criterios de aceptación: AC1, AC2 y AC3.

AC1 está cubierto por el caso de prueba TC1, AC2 por TC2 y AC3 por TC3. La ejecución de la prueba

El historial tuvo tres ejecuciones de prueba en tres versiones consecutivas del software de la siguiente manera:

	Ejecución 1	Ejecución 2	Ejecución 3
TC1	(1) Fallido	(4) Aprobado	(7) Aprobado
TC2	(2) Aprobado	(5) fallido	(8) Aprobado
TC3	(3) fallido	(6) fallido	(9) Aprobado

Las pruebas se repiten una vez que se le informa que todos los defectos encontrados en la ejecución de la prueba son corregido y una nueva versión del software está disponible.

¿Cuáles de las pruebas anteriores se ejecutan como pruebas de regresión?

- a) Sólo 4, 7, 8, 9
- b) Sólo 5, 7
- c) Sólo 4, 6, 8, 9
- d) Sólo 5, 6

Seleccione UNA opción.

Pregunta #36 (1 punto)

¿Cómo está presente el enfoque de equipo completo en las interacciones entre los evaluadores y representantes comerciales?

- a) Los representantes comerciales deciden los enfoques de automatización de pruebas.
- b) Los evaluadores ayudan a los representantes comerciales a definir la estrategia de prueba.
- c) Los representantes comerciales no forman parte del enfoque de equipo completo.
- d) Los evaluadores ayudan a los representantes comerciales a crear pruebas de aceptación adecuadas.

Seleccione UNA opción.

Pregunta #37 (1 punto)

¿Cuál de estas afirmaciones NO es un factor que contribuye a que las revisiones sean exitosas?

- a) Los participantes deberán dedicar el tiempo adecuado a la revisión.
- b) Dividir productos de trabajo grandes en partes pequeñas para reducir el esfuerzo requerido intenso.
- c) Los participantes deben evitar comportamientos que puedan indicar aburrimiento, exasperación, u hostilidad hacia otros participantes.
- d) Las fallas encontradas deben ser reconocidas, valoradas y manejadas objetivamente.

Seleccione UNA opción.

Pregunta #38 (1 punto)

¿Cuál de las siguientes opciones muestra un ejemplo de actividades de prueba que contribuyen al éxito?

- a) Tener probadores involucrados durante diversas actividades del ciclo de vida de desarrollo de software (SDLC) ayudará a detectar defectos en los productos de trabajo.
- b) Los evaluadores intentan no molestar a los desarrolladores mientras codifican, para que los desarrolladores escriban un mejor código.
- c) Los probadores que colaboran con los usuarios finales ayudan a mejorar la calidad de los informes de defectos durante la integración de componentes y las pruebas del sistema.
- d) Los probadores certificados diseñarán casos de prueba mucho mejores que los probadores no certificados.

Seleccione UNA opción.

Pregunta #39 (1 punto)

¿Cuál de las siguientes afirmaciones describe un objetivo de prueba válido?

- a) Acreditar que no existen defectos no solucionados en el sistema sometido a prueba.
- b) Acreditar que no habrá fallas luego de la implementación del sistema en producción. c) Reducir el nivel de riesgo del objeto de prueba y generar confianza en la calidad. nivel.
- d) Verificar que no existen combinaciones de insumos no probadas.

Seleccione UNA opción.

Pregunta #40 (1 punto)

¿Cuál de los siguientes factores (i – v) tiene una influencia SIGNIFICATIVA en el proceso de prueba?

- i. El SDLC.
- ii. El número de defectos detectados en proyectos anteriores. III. Los riesgos del producto identificados. IV.
- Los nuevos requisitos reglamentarios obligan. v. El número de testers certificados en la organización.

- a) i y ii tienen influencia significativa; iii, iv y v no. b) i, iii y iv tienen influencia significativa; ii y v no lo han hecho. c) ii, iv y v tienen influencia significativa; I y III no lo han hecho. d) iii y v tienen una influencia significativa; I, II y IV no.

Seleccione UNA opción.

Preguntas de muestra adicionales



La regla general de ISTQB® para publicar preguntas de muestra requiere que se publique al menos una pregunta de muestra para cada objetivo de aprendizaje. Por lo tanto, si hay más objetivos de aprendizaje que preguntas de examen, las preguntas que cubren objetivos de aprendizaje no incluidos en el conjunto de examen de muestra se publican por separado como preguntas complementarias. Esta sección contiene las preguntas complementarias oficiales.

Pregunta #A1 (1 punto)

Se le asignó la tarea de analizar y solucionar las causas de las fallas en un nuevo sistema que se lanzará.

¿Qué actividad estás realizando?

- a) Depuración. b)
- Pruebas de software. c)
- Elicitación de requisitos. d) Gestión de defectos.

Seleccione UNA opción.

Pregunta #A2 (1 punto)

En muchas organizaciones de software, el departamento de pruebas se denomina departamento de Garantía de Calidad (QA). ¿Esta oración es correcta o no y por qué?

- a) Es correcto. Las pruebas y el control de calidad significan exactamente lo mismo. b) Es correcto. Estos nombres se pueden usar indistintamente porque tanto las pruebas como el control de calidad centran sus actividades en los mismos problemas de calidad. c) No es correcto. Las pruebas son algo más; Las pruebas incluyen todas las actividades relacionadas con la calidad. El control de calidad se centra en los procesos relacionados con la calidad. d) No es correcto. El control de calidad se centra en los procesos relacionados con la calidad, mientras que las pruebas se concentran en demostrar que un componente o sistema es adecuado para su propósito y detectar defectos.

Seleccione UNA opción.

Pregunta #A3 (1 punto)

Un teléfono que suena en un cubículo vecino distrae a un programador, lo que le hace programar incorrectamente la lógica que verifica el límite superior de una variable de entrada.

Más tarde, durante la prueba del sistema, un evaluador nota que este campo de entrada acepta valores de entrada no válidos.

¿Cuál de las siguientes opciones describe correctamente un límite superior codificado incorrectamente?

- a) La causa raíz. b) Un fracaso. c) Un error. d) Un defecto.

Seleccione UNA opción.

Pregunta #A4 (1 punto)

Considere el siguiente software de prueba.

Carta de Prueba #04.018 Duración de la sesión: 1 h

Explorar:	Página de registro
Con:	Diferentes conjuntos de datos de entrada incorrectos
Para descubrir:	Defectos relacionados con la aceptación del proceso de registro con entrada incorrecta

¿Qué actividad de prueba produce este software de prueba como resultado?

- a) Planificación de pruebas. b) Seguimiento y control de las pruebas. c) Análisis de pruebas. d) Diseño de pruebas.

Seleccione UNA opción.

Pregunta #A5 (1 punto)

¿Cuál de los siguientes es el MEJOR ejemplo de cómo la trazabilidad respalda las pruebas?

- a) La realización del análisis de impacto de un cambio dará información sobre la finalización de las pruebas. b) El análisis de la trazabilidad entre los casos de prueba y los resultados de las pruebas proporcionará información.
- información sobre el nivel estimado de riesgo residual.
- c) Realizar el análisis de impacto de un cambio ayudará a seleccionar la prueba adecuada.
- Casos para pruebas de regresión.
- d) Analizar la trazabilidad entre la base de prueba, los objetos de prueba y los casos de prueba ayudará a seleccionar los datos de prueba para lograr la cobertura supuesta del objeto de prueba.

Seleccione UNA opción.

Pregunta #A6 (1 punto)

¿Cuál de las siguientes opciones explica MEJOR un beneficio de la independencia de las pruebas?

- a) El uso de un equipo de prueba independiente permite a la dirección del proyecto asignar la responsabilidad de la calidad del producto final al equipo de prueba. b) Si se puede permitir un equipo de prueba externo a la organización, entonces existen claros beneficios en términos de que este equipo externo no se deje influenciar tan fácilmente por las preocupaciones de entrega de la gestión de proyectos y la necesidad de cumplir plazos de entrega estrictos. c) Un equipo de pruebas independiente puede trabajar por separado de los desarrolladores, no necesita distraerse con los cambios en los requisitos del proyecto y puede restringir la comunicación con los desarrolladores para informar sobre defectos a través del sistema de gestión de defectos. d) Cuando las especificaciones contienen ambigüedades e inconsistencias, se hacen suposiciones sobre su interpretación, y un evaluador independiente puede ser útil para cuestionar esas suposiciones y la interpretación hecha por el desarrollador.

Seleccione UNA opción.

Pregunta #A7 (1 punto)

Trabajas como tester en el equipo que sigue el modelo V. ¿Cómo afecta la elección de este modelo de ciclo de vida de desarrollo de software (SDLC) al momento de las pruebas?

- a) Las pruebas dinámicas no se pueden realizar al principio del SDLC. b) Las pruebas estáticas no se pueden realizar al principio del SDLC. c) La planificación de pruebas no se puede realizar en una fase temprana del SDLC. d) Las pruebas de aceptación se pueden realizar al principio del SDLC.

Seleccione UNA opción.

Pregunta #A8 (1 punto)

¿Cuáles de las siguientes son ventajas de DevOps?

- i. Lanzamiento de producto más rápido y tiempo de comercialización más rápido. ii. Aumenta la necesidad de realizar pruebas manuales repetitivas. III. Disponibilidad constante de software ejecutable. IV. Reducción del número de pruebas de regresión asociadas a la refactorización de código. v. Configurar el marco de automatización de pruebas es económico ya que todo está automatizado.
- a) i, ii y iv son ventajas; iii y v no lo son. b) lii yv son ventajas; i, ii y iv no lo son. c) i y iii son ventajas; ii, iv y v no lo son. d) ii, iv y v son ventajas; I y III no lo son.

Seleccione UNA opción.

Pregunta #A9 (1 punto)

Trabajas como tester en un proyecto sobre una aplicación móvil para pedidos de comida para uno de tus clientes. El cliente le envió una lista de requisitos. Uno de ellos, con alta prioridad, dice

El pedido debe procesarse en menos de 10 segundos en el 95% de los casos.

Creó un conjunto de casos de prueba en los que se realizaron varios pedidos aleatorios, se midió el tiempo de procesamiento y los resultados de las pruebas se compararon con los requisitos.

¿Qué tipo de prueba realizó?

- a) Funcional, porque los casos de prueba cubren los requisitos comerciales del usuario para el sistema.
- b) No funcionales, porque miden el desempeño del sistema. c) Funcional, porque los casos de prueba interactúan con la interfaz de usuario. d) Estructural, porque necesitamos conocer la estructura interna del programa para medir el tiempo de procesamiento de pedidos.

Seleccione UNA opción.

Pregunta #A10 (1 punto)

La estrategia de prueba de su organización sugiere que una vez que se retire un sistema, se debe probar la migración de datos. ¿Como parte de qué tipo de prueba es MÁS probable que se realice esta prueba?

- a) Pruebas de mantenimiento.
- b) Pruebas de regresión. c)
- Pruebas de componentes. d)
- Pruebas de integración.

Seleccione UNA opción.

Pregunta #A11 (1 punto)

La siguiente es una lista de los productos de trabajo producidos en el SDLC.

- i. Requisitos comerciales. ii.
- Cronograma.
- III. Presupuesto de prueba. IV. Código ejecutable de terceros. v. Historias de usuarios y sus criterios de aceptación.

¿Cuáles de ellos se pueden revisar?

- a) i y iv pueden revisarse; ii, iii y v no pueden. b) se pueden revisar i, ii, iii y iv; No puedo. c) se pueden revisar i, ii, iii y v; No puedo. d) iii, iv y v pueden ser revisados; I y II no pueden.

Seleccione UNA opción.

Pregunta #A12 (1 punto)

Decida cuáles de las siguientes afirmaciones (i–v) son verdaderas para las pruebas dinámicas y cuáles son verdaderas para las pruebas estáticas.

i. Los comportamientos externos anormales son más fáciles de identificar con esta prueba. ii. Las discrepancias con respecto a un estándar de codificación son más fáciles de encontrar con esta prueba.

III. Identifica fallas causadas por defectos cuando se ejecuta el software. IV. Su objetivo de prueba es identificar defectos lo antes posible. v. La cobertura faltante para requisitos de seguridad críticos es más fácil de encontrar y solucionar.

a) i, iv y v son verdaderos para pruebas estáticas; ii y iii son ciertos para las pruebas dinámicas. b) i, iii y iv son verdaderos para pruebas estáticas; ii y v son ciertos para pruebas dinámicas. c) ii y iii son ciertos para pruebas estáticas; i, iv y v son verdaderos para pruebas dinámicas. d) ii, iv y v son verdaderos para pruebas estáticas; i, iii y iv son válidos para pruebas dinámicas.

Seleccione UNA opción.

Pregunta #A13 (1 punto)

¿Cuál de las siguientes afirmaciones sobre revisiones formales es VERDADERA?

a) Algunas revisiones no requieren más de un rol. b) El proceso de revisión tiene varias actividades. c) La documentación a revisar no se distribuye antes de la reunión de revisión, con excepción del producto del trabajo para tipos de revisión específicos. d) Los defectos encontrados durante la revisión no se reportan ya que no se encuentran mediante pruebas dinámicas.

Seleccione UNA opción.

Pregunta #A14 (1 punto)

¿Qué tarea puede asumir la dirección durante una revisión formal?

a) Asumir la responsabilidad general de la revisión. b) Decidir qué se va a revisar. c) Velar por el eficaz desarrollo de las reuniones de revisión y mediar, en su caso. d) Registrar información de revisión, como decisiones de revisión.

Seleccione UNA opción.

Pregunta #A15 (1 punto)

Un sistema de almacenamiento de vino utiliza un dispositivo de control que mide la temperatura de la celda del vino. T (medido en °C, redondeado al grado más cercano) y alerta al usuario si se desvía del valor óptimo de 12, de acuerdo con las siguientes reglas:

- Si $T = 12$, el sistema dice "Temperatura óptima". • Si $T < 12$, el sistema dice: "¡La temperatura es demasiado baja!" • Si $T > 12$, el sistema dice: "¡La temperatura es demasiado alta!"

Desea utilizar el análisis de valor límite (BVA) de 3 valores para verificar el comportamiento del dispositivo de control. Una entrada de prueba es una temperatura en °C proporcionada por el dispositivo.

¿Cuál es el conjunto MÍNIMO de entradas de prueba que logra el 100% de la cobertura deseada?

- a) 11, 12, 13 b) 10,
12, 14 c) 10, 11,
12, 13, 14 d) 10, 11, 13, 14

Seleccione UNA opción.

Pregunta #A16 (1 punto)

¿Cuál de las siguientes afirmaciones sobre las pruebas de rama es CORRECTA?

- a) Si un programa incluye solo sucursales incondicionales, entonces se puede lograr una cobertura de sucursales del 100% sin ejecutar ningún caso de prueba. b) Si los casos de prueba ejercen todas las ramas incondicionales del código, entonces 100% Se logra cobertura de sucursales.
- c) Si se logra una cobertura del 100% del estado de cuenta, entonces también se alcanza una cobertura del 100% de la sucursal lograda.
- d) Si se logra una cobertura del 100% de las sucursales, entonces todos los resultados de las decisiones en cada se ejercen las declaraciones de decisión contenidas en el código.

Seleccione UNA opción.

Pregunta #A17 (1 punto)

Está probando una aplicación móvil que permite a los clientes acceder y administrar sus cuentas bancarias. Está ejecutando un conjunto de pruebas que implica evaluar cada pantalla y cada campo de cada pantalla con una lista general de mejores prácticas de interfaz de usuario derivadas de un libro popular sobre el tema que maximiza el atractivo, la facilidad de uso y la accesibilidad para dichas aplicaciones.

¿Cuál de las siguientes opciones categoriza MEJOR la técnica de prueba que está utilizando?

- a) Caja negra. b)
Exploratorio. c) Basado
en listas de verificación. d)
- Error al adivinar.

Seleccione UNA opción.

Pregunta #A18 (1 punto)

¿Cuál de las siguientes opciones describe MEJOR el enfoque colaborativo para la redacción de historias de usuario?

- a) Las historias de usuario son creadas por evaluadores y desarrolladores y luego aceptadas por representantes comerciales. b) Las historias de usuario son creadas por representantes comerciales, desarrolladores y evaluadores juntos.
- c) Las historias de usuario son creadas por representantes comerciales y verificadas por desarrolladores y probadores.

- d) Las historias de usuario se crean de manera que sean independientes, negociables, valiosas, estimable, pequeño y comprobable.

Seleccione UNA opción.

Pregunta #A19 (1 punto)

Considere la siguiente parte de un plan de prueba.

Las pruebas se realizarán mediante pruebas de componentes e integración de componentes. pruebas. La normativa exige demostrar que el 100% de cobertura de sucursales es logrados para cada componente clasificado como crítico.

¿A qué parte del plan de prueba pertenece esta parte?

- a) Comunicación.
- b) Registro de riesgos.
- c) Contexto de las pruebas.
- d) Enfoque de prueba.

Seleccione UNA opción.

Pregunta #A20 (1 punto)

El equipo utiliza el póker de planificación para estimar el esfuerzo de prueba para un nuevo requisito. característica. Hay una regla en su equipo que dice que si no hay tiempo para llegar a un acuerdo total y la variación en los resultados es pequeña, aplicando reglas como "aceptar el número con se puede aplicar la mayor cantidad de votos".

Después de dos rondas, no se alcanzó consenso, por lo que se inició la tercera ronda.

Puede ver los resultados de la estimación de la prueba en la siguiente tabla.

	Estimaciones de los miembros del equipo						
La ronda 1	21	2	5	34	13	8	2
La ronda 2	13	8	8	34	13	8	5
Ronda 3	13	8	13	13	13	13	8

¿Cuál de los siguientes es el MEJOR ejemplo del siguiente paso?

- a) El propietario del producto debe intervenir y tomar una decisión final.
- b) Acepte 13 como estimación de prueba final, ya que tiene la mayoría de los votos.
- c) No es necesaria ninguna otra acción. Se ha llegado a un consenso.
- d) Eliminar la nueva característica de la versión actual porque no se ha logrado consenso. sido alcanzado.

Seleccione UNA opción.

Pregunta #A21 (1 punto)

¿Cuál de las siguientes afirmaciones NO es cierta con respecto a la pirámide de prueba?

- a) La pirámide de pruebas enfatiza tener una mayor cantidad de pruebas en la prueba inferior niveles.
- b) Cuanto más cerca de la cima de la pirámide, más formal será la automatización de sus pruebas. debiera ser.

- c) Por lo general, las pruebas de componentes y las pruebas de integración de componentes se automatizan utilizando herramientas basadas en API. d) Para las pruebas del sistema y las pruebas de aceptación, las pruebas automatizadas generalmente se crean utilizando herramientas basadas en GUI.

Seleccione UNA opción.

Pregunta #A22 (1 punto)

Durante el análisis de riesgos, el equipo consideró el siguiente riesgo: El sistema permite un descuento demasiado alto para un cliente. El equipo estimó que el impacto del riesgo era muy alto.

¿Qué se puede decir sobre la probabilidad de riesgo?

- a) También es muy alto. Un impacto de alto riesgo siempre implica una alta probabilidad de riesgo. b) Es muy bajo. Un impacto de alto riesgo siempre implica una baja probabilidad de riesgo. c) No se puede decir nada sobre la probabilidad del riesgo. El impacto del riesgo y la probabilidad del riesgo son independientes. d) La probabilidad del riesgo no es importante con un impacto de riesgo tan alto. uno no necesita para definirlo.

Seleccione UNA opción.

Pregunta #A23 (1 punto)

La siguiente lista contiene riesgos que se han identificado para el desarrollo de un nuevo producto de software:

- i. La dirección traslada a dos evaluadores experimentados a otro proyecto. ii. El sistema no cumple con los estándares de seguridad funcional. III. El tiempo de respuesta del sistema excede los requisitos del usuario. IV. Las partes interesadas tienen expectativas inexactas. v. Las personas con discapacidad tienen problemas al utilizar el sistema.

¿Cuáles de ellos son riesgos del proyecto?

- a) i y iv son riesgos del proyecto; ii, iii y v no son riesgos del proyecto. b) iv y v son riesgos del proyecto; i, ii y iii no son riesgos del proyecto. c) i y iii son riesgos del proyecto; ii, iv y v no son riesgos del proyecto. d) ii y v son riesgos del proyecto; i, iii y iv no son riesgos del proyecto.

Seleccione UNA opción.

Pregunta #A24 (1 punto)

¿Cuál de los siguientes es un ejemplo de cómo el análisis de riesgos del producto influye en la minuciosidad y el alcance de las pruebas?

- a) El director de pruebas monitorea e informa diariamente el nivel de todos los riesgos conocidos para que las partes interesadas puedan tomar una decisión informada sobre la fecha de lanzamiento. b) Uno de los riesgos identificados fue la "Falta de soporte de bases de datos de código abierto", por lo que el equipo decidió integrar el sistema con una base de datos de código abierto. c) Durante el análisis de riesgo cuantitativo, el equipo estimó el nivel total de todos los riesgos identificados y lo informó como el riesgo residual total antes de la prueba.

- d) La evaluación de riesgos reveló un nivel muy alto de riesgos de desempeño, por lo que se decidió realizar pruebas detalladas de eficiencia del desempeño al principio del SDLC.

Seleccione UNA opción.

Pregunta #A25 (1 punto)

¿Cuáles DOS de las siguientes opciones son métricas comunes utilizadas para informar sobre el nivel de calidad del objeto de prueba?

- a) Número de defectos encontrados durante las pruebas del sistema.
b) Esfuerzo total en el diseño de pruebas dividido por el número de casos de prueba diseñados. c)
Número de procedimientos de prueba ejecutados. d)
Número de defectos encontrados dividido por el tamaño de un producto de trabajo. e) Tiempo
necesario para reparar un defecto.

Seleccione DOS opciones.

Pregunta #A26 (1 punto)

¿Cuál de los siguientes datos contenidos en un informe de progreso de prueba es MENOS útil para los representantes comerciales?

- a) Impedimentos para realizar las
pruebas. b) Cobertura de sucursales
lograda. c) Progreso de
la prueba. d) Nuevos riesgos dentro del ciclo de pruebas.

Seleccione UNA opción.

Conjunto de exámenes A: Respuestas



Pregunta #1

FL-4.2.2 (K3)

Respuesta correcta: a

Hay 12 valores límite para los valores del resultado final: 0, 50, 51, 60, 61, 70, 71, 80, 81, 90, 91 y 100. Los casos de prueba cubren seis de ellos (TC1, 91; TC2, 50; TC3, 81; Por lo tanto, los casos de prueba cubren 6/12 = 50%.

a) Es correcto. b)

No es correcto. c) No es correcto. d) No es correcto.

Pregunta #2

FL-3.2.1 (K1)

Respuesta correcta: d.

a) No es correcto. La retroalimentación puede mejorar el proceso de prueba, pero si uno solo quiere mejorar proyectos futuros, no es necesario que la retroalimentación llegue temprano o con frecuencia.

b) No es correcto. La retroalimentación no se utiliza para priorizar los requisitos. c) No es correcto. La calidad de los cambios se puede medir de múltiples maneras. d) Es correcto. La retroalimentación temprana y frecuente permite la comunicación temprana de posibles problemas de calidad.

Pregunta #3

FL-4.2.3 (K3)

Respuesta correcta: d.

a) No es correcto. Un miembro que no haya cumplido el plazo puede obtener un descuento y una camiseta de regalo después de 15 alquileres de bicicletas. b) No es correcto. Un miembro sin una fecha límite incumplida puede obtener un descuento pero ningún regalo

Camiseta hasta que alquilaron una bicicleta 15 veces.

- c) No es correcto. Los no miembros no pueden obtener un descuento, incluso si no se perdieron una fecha límite todavía.
- d) Es correcto. No habrá descuento como no socio que además haya incumplido un plazo, pero sólo los socios podrán recibir una camiseta de regalo. Por tanto, la acción no es correcta.

Pregunta #4

FL-5.4.1 (K2)

Respuesta correcta:c

- a) No es correcto. La trazabilidad es la relación entre dos o más productos de trabajo, no entre diferentes versiones del mismo producto de trabajo. b) No es correcto. Las pruebas de mantenimiento consisten en probar cambios; no está estrechamente relacionado con el control de versiones.
- c) Es correcto. Para respaldar las pruebas, la gestión de la configuración puede implicar el control de versiones de todos los elementos de prueba.
- d) No es correcto. La ingeniería de requisitos es la obtención, documentación y gestión de requisitos; no está estrechamente relacionado con el control de versiones del script de prueba.

Pregunta #5

FL-4.1.1 (K2)

Respuesta correcta:c

- a) No es correcto. Ésta es una característica común de las técnicas de prueba de caja blanca. Las condiciones de prueba, los casos de prueba y los datos de prueba se derivan de una base de prueba que puede incluir código, arquitectura de software, diseño detallado o cualquier otra fuente de información sobre la estructura del software.
- b) No es correcto. Ésta es una característica común de las técnicas de prueba de caja blanca. La cobertura se mide en función de los elementos probados dentro de una estructura seleccionada y la técnica aplicada a la base de la prueba. c) Es correcto. Esta es una característica común de las técnicas de prueba basadas en la experiencia. Este conocimiento y experiencia incluyen el uso esperado del software, su entorno, defectos probables y la distribución de aquellos defectos utilizados para definir las pruebas. d) No es correcto. Ésta es una característica común de las técnicas de prueba de caja negra. Los casos de prueba se pueden utilizar para detectar brechas dentro de los requisitos y la implementación de los mismos, así como desviaciones de los requisitos.

Pregunta #6

FL-4.2.4 (K3)

Respuesta correcta: d

Las transiciones de "prueba" y "error" no pueden ocurrir en un caso de prueba. Tampoco pueden ambas transiciones "terminadas". Esto significa que necesitamos al menos tres casos de prueba para lograr la cobertura de transición. Por ejemplo:

TC1: prueba, hecho

TC2: ejecutar, error, hecho

TC3: ejecutar, pausar, reanudar, pausar, listo

Por eso

- a) No es correcto b) No es correcto c) No es correcto d) Es correcto

Pregunta #7

FL-5.5.1 (K3)

Respuesta correcta:c

- a) No es correcto. El resultado esperado es "la aplicación debería aceptar la entrada proporcionada y crear el usuario". El resultado real es "La aplicación se cuelga después de ingresar 'Entrada de prueba'. \$ä."
- b) No es correcto. Hay una referencia al caso de prueba y al requisito relacionado, y establece que se rechaza el defecto. Además, el estado del defecto no sería de mucha ayuda para los desarrolladores.
- c) Es correcto. No sabemos en qué entorno de prueba se detectó la anomalía y tampoco sabemos qué aplicación (y su versión) está afectada. d) No es correcto. El informe de defectos afirma que la anomalía es urgente y que es un problema global (es decir, muchas, si no todas, las cuentas de administración de pruebas se ven afectadas) y afirma que el impacto es alto para las partes interesadas del negocio.

Pregunta #8

FL-5.1.2 (K1)

Respuesta correcta:c

- a) No es correcto. Las prioridades para las historias de usuarios están determinadas por los representantes comerciales, representante junto con el equipo de desarrollo.
- b) No es correcto. Los evaluadores se centran en aspectos funcionales y no funcionales del sistema a probar.
- c) Es correcto. Según el plan de estudios, esta es una de las formas en que los evaluadores agregan valor a Planificación de iteraciones y lanzamientos.
- d) No es correcto. El diseño de prueba inicial no forma parte de la planificación del lanzamiento. El diseño temprano de las pruebas no garantiza automáticamente el lanzamiento de software de calidad.

Pregunta #9

FL-1.4.1 (K2)

Respuesta correcta:b

- a) No es correcto. Estimar el esfuerzo de la prueba es parte de la planificación de la prueba.
- b) Es correcto. Este es un ejemplo de definición de condiciones de prueba, que es parte de la prueba. análisis.
- c) No es correcto. El uso de técnicas de prueba para derivar elementos de cobertura es parte de la prueba. diseño.
- d) No es correcto. Informar los defectos encontrados durante las pruebas dinámicas es parte de la prueba. ejecución.

Pregunta #10

FL-5.3.3 (K2)

Respuesta correcta: d.

- a) No es correcto. Los criterios de aceptación son las condiciones utilizadas para decidir si el La historia del usuario está lista. No pueden mostrar el progreso del trabajo.
- b) No es correcto. Los informes de defectos informan sobre los defectos. No muestran trabajo progreso.
- c) No es correcto. El informe de finalización de la prueba se puede crear una vez finalizada la iteración, por lo que no mostrará el progreso continuamente dentro de una iteración. d) Es correcto. Los gráficos de evolución son una representación gráfica del trabajo que queda por hacer en comparación con el tiempo restante. Se actualizan diariamente, por lo que pueden mostrar continuamente el avance del trabajo.

Pregunta #11

FL-2.1.3 (K1)

Respuesta correcta:c

- a) No es correcto. Se utiliza con mayor frecuencia en el desarrollo impulsado por el comportamiento (BDD). b) No es correcto. Es la descripción del desarrollo basado en pruebas (TDD). c) Es correcto. En el desarrollo basado en pruebas de aceptación (ATDD), las pruebas se escriben a partir de criterios de aceptación como parte del proceso de diseño. d) No es correcto. Se utiliza en BDD.

Pregunta #12

FL-6.2.1 (K1)

Respuesta correcta:b

- a) No es correcto. La automatización de pruebas no introduce regresiones desconocidas en producción.
- b) Es correcto. La asignación incorrecta de esfuerzos para mantener el software de prueba es un riesgo.
- c) No es correcto. Las herramientas de prueba deben seleccionarse de manera que ellas y su software de prueba puedan ser confiado.
- d) No es correcto. El objetivo principal de la automatización de pruebas es reducir las pruebas manuales. Entonces, esto es un beneficio, no un riesgo.

Pregunta #13

FL-5.1.3 (K2)

Respuesta correcta: c, e

- a) No es correcto. La preparación del entorno de prueba es un criterio de disponibilidad de recursos; por tanto, pertenece a los criterios de entrada. b) No es correcto. Este es un criterio de disponibilidad de recursos; por tanto, pertenece a la criterio para entrar.
- c) Es correcto. La densidad estimada de defectos es una medida de diligencia; por lo tanto, pertenece a los criterios de salida.
- d) No es correcto. Los requisitos traducidos a un formato determinado dan como resultado requisitos comprobables; por tanto, pertenece a los criterios de entrada.

- e) Es correcto. La automatización de las pruebas de regresión es un criterio de finalización; por lo tanto, pertenece a los criterios de salida.

Pregunta #14

FL-3.1.2 (K2)

Respuesta correcta: una

- a) Es correcto. La gestión de defectos no es menos costosa. Encontrar y reparar defectos más adelante en SDLC es más costoso.
b) No es correcto. Este es un beneficio de las pruebas estáticas. c) No es correcto. Este es un beneficio de las pruebas estáticas. d) No es correcto. Este es un beneficio de las pruebas estáticas.

Pregunta #15

FL-1.5.1 (K2)

Respuesta correcta:b

- i. Es verdad. Tener conocimiento del dominio es una habilidad importante del evaluador.
ii. Es falso. Esta es una tarea del analista de negocios junto con el negocio representante. III.
Es verdad. Ser un buen jugador de equipo es una habilidad importante. IV. Es falso. Planificar y organizar el trabajo del equipo es tarea del responsable de pruebas o, sobre todo en un proyecto de desarrollo de software ágil, de todo el equipo y no sólo del tester. v. Es cierto. El pensamiento crítico es una de las habilidades más importantes de los evaluadores.

Por tanto b es correcto.

Pregunta #16

FL-2.1.6 (K2)

Respuesta correcta:c

- a) No es correcto. Las retrospectivas son más útiles para identificar oportunidades de mejora y tienen poca importancia para los clientes. b) No es correcto. Los representantes comerciales no dan comentarios sobre el producto en sí. Por lo tanto, no hay ningún beneficio económico para la organización. c) Es correcto. Las retrospectivas realizadas periódicamente, cuando se realizan actividades de seguimiento adecuadas, son fundamentales para la mejora continua del desarrollo y las pruebas.
d) No es correcto. El coraje y el respeto son valores de Extreme Programming y no están estrechamente relacionados con las retrospectivas.

Pregunta #17**FL-4.4.2 (K2)****Respuesta correcta:**

- a) No es correcto. Este es un nuevo producto. Probablemente aún no tenga una lista de verificación y es posible que no conozca las condiciones de la prueba debido a que faltan requisitos. b) No es correcto. Este es un nuevo producto. Probablemente no tengas suficiente información para realizar conjeturas erróneas correctas.
- c) Es correcto. Las pruebas exploratorias son más útiles cuando hay pocas especificaciones conocidas. fificaciones, y/o hay un cronograma apremiante para las pruebas.
- d) No es correcto. Las pruebas de sucursales llevan mucho tiempo y su gerencia está preguntando sobre algunos resultados de las pruebas ahora. Además, las pruebas de rama no implican conocimiento del dominio.

Pregunta #18**FL-5.1.4 (K3)****Respuesta correcta:**

- d En la técnica de estimación de tres puntos

$$E = \frac{\text{optimista} + \text{probable} + \text{pesimista}}{3} = \frac{6 + 2 + 4 + 8 + 11 + 14}{6} = 10: \quad d$$

Por tanto d es correcto.

Pregunta #19**FL-4.3.3 (K2)****Respuesta correcta:** d.

- a) No es correcto. La ventaja fundamental de las técnicas de prueba de caja blanca es que durante las pruebas se tiene en cuenta toda la implementación del software. b) No es correcto. Las medidas de cobertura de caja blanca proporcionan una medida objetiva de la cobertura y proporcionan la información necesaria para permitir que se generen pruebas adicionales para aumentar esta cobertura.
- c) No es correcto. Se pueden utilizar técnicas de prueba de caja blanca para realizar revisiones (estática pruebas).
- d) Es correcto. Ésta es la debilidad de las técnicas de prueba de caja blanca. No pueden identificar la implementación faltante porque se basan únicamente en la estructura del objeto de prueba, no en la especificación de requisitos.

Pregunta #20**FL-3.2.4 (K2)****Respuesta correcta:**

- b Considerando los atributos:

- Existe una función de escribano, especificada para recorridos, revisiones técnicas e inspecciones; por lo tanto, las revisiones que se realizan no pueden ser revisiones informales. • El propósito es evaluar la calidad; el propósito de evaluar la calidad es uno de los objetivos más importantes de un recorrido.

- La reunión de revisión está dirigida por el autor del producto del trabajo; esto no está permitido para las inspecciones y normalmente no se hace en las revisiones técnicas. Se necesita un moderador en los recorridos y se le permite realizar revisiones informales.
- Los revisores individuales encuentran posibles anomalías durante la preparación: todo tipo de revisiones pueden incluir revisores individuales (incluso revisiones informales). • Se produce un informe de revisión: todos los tipos de revisiones pueden producir un informe de revisión.

aunque las revisiones informales no requieren documentación.

Por tanto b es correcto.

Pregunta #21

FL-4.2.1 (K3)

Respuesta correcta:

b "Jardín pequeño" y "jardín grande" solo pueden ir con "planta baja", por lo que necesitamos dos casos de prueba con "planta baja", que cubran estas dos particiones "tipo jardín".

Necesitamos dos casos de prueba más para cubrir las otras dos particiones de "piso" y una partición restante "tipo jardín" de "sin jardín". Necesitamos un total de cuatro casos de prueba:

TC1 (planta baja, pequeño jardín).

TC2 (planta baja, amplio jardín).

TC3 (primer piso, sin jardín).

TC4 (segundo piso o superior, sin jardín).

a) No es correcto. b) Es

correcto. c) No es

correcto. d) No es

correcto.

Pregunta #22

FL-2.1.5 (K2)

Respuesta correcta: d.

a) No es correcto. La revisión temprana es un ejemplo del enfoque de desplazamiento a la izquierda. b) No es correcto. TDD es un ejemplo del enfoque de desplazamiento a la izquierda.

c) No es correcto. Las pruebas no funcionales tempranas son un ejemplo del desplazamiento hacia la izquierda acercarse.

d) Es correcto. Los scripts de prueba deben estar sujetos a la gestión de configuración, por lo que no tiene sentido crear los scripts de prueba antes de configurar este proceso.

Pregunta #23

FL-6.1.1 (K2)

Respuesta correcta:c

a) No es correcto. El monitoreo de pruebas implica la verificación continua de todas las actividades y la comparación del progreso real con el plan de pruebas. El control de pruebas implica tomar las acciones necesarias para cumplir con los objetivos de prueba del plan de pruebas. Durante estas actividades no se preparan datos de prueba.

b) No es correcto. El análisis de pruebas incluye analizar la base de la prueba para identificar las condiciones de la prueba y priorizarlas. El diseño de la prueba incluye la elaboración de la prueba.

condiciones en casos de prueba y otro software de prueba. Los datos de prueba no se preparan durante estas actividades.

- c) Es correcto. La implementación de la prueba incluye la creación o adquisición del software de prueba.
necesario para la ejecución de la prueba (por ejemplo, datos de prueba).
- d) No es correcto. Las actividades de finalización de las pruebas ocurren en los hitos del proyecto (por ejemplo, lanzamiento, final de la iteración, finalización del nivel de prueba), por lo que es demasiado tarde para preparar los datos de la prueba.

Pregunta #24 FL-4.3.1

(K2)

Respuesta correcta: una

- a) Es correcto. Dado que se logra una cobertura del 100% de los extractos, todos los extractos, incluidos los defectuosos, deben haber sido ejecutados y evaluados al menos una vez b) No es correcto. La cobertura depende de lo que se prueba, no de la cantidad de casos de prueba. Por ejemplo, para el código "if (x==0) y=1", un caso de prueba (x=0) logra una cobertura de declaración del 100%, pero dos casos de prueba (x=1) y (x=2) juntos solo logran 50% de cobertura del estado de cuenta.
- c) No es correcto. Si hay un bucle en el código, puede haber un número infinito de rutas posibles, por lo que no es posible ejecutar todas las rutas posibles en el código. d) No es correcto. No es posible realizar pruebas exhaustivas (consulte la sección de los siete principios de las pruebas en el programa de estudios). Por ejemplo, para el código "entrada x; print x" cualquier prueba con x arbitraria logra una cobertura de declaración del 100%, pero cubre un valor de entrada.

Pregunta #25

FL-2.1.2 (K1)

Respuesta correcta: d.

- a) No es correcto. b) No es correcto. c) No es correcto.
- d) Es correcto; Esta regla es válida para todos los modelos SDLC.

Pregunta #26 FL-5.1.7

(K2)

Respuesta correcta: una

Las pruebas de usabilidad se encuentran en el tercer trimestre (1—C).

La prueba de componentes se encuentra en Q1 (2—A).

Las pruebas funcionales se encuentran en el segundo trimestre (3—B).

Las pruebas de confiabilidad se encuentran en el cuarto trimestre (4—D).

Por tanto a es correcto.

Pregunta #27**FL-2.2.1 (K2)**

Respuesta correcta: una

La base de prueba para las pruebas de aceptación son las necesidades comerciales del usuario (1D).

La comunicación entre componentes se prueba durante las pruebas de integración de componentes.

(2B).

Se pueden encontrar fallas en la lógica durante las pruebas de componentes (3A).

Las reglas comerciales son la base de prueba para las pruebas del sistema (4C).

Por tanto a es correcto.

Pregunta #28**FL-4.5.2 (K2)**

Respuesta correcta:b

a) No es correcto. Las retrospectivas se utilizan para capturar lecciones aprendidas y mejorar el proceso de desarrollo y prueba, no para documentar los criterios de aceptación.

b) Es correcto. Esta es la forma estándar de documentar los criterios de aceptación. c) No es correcto. La comunicación verbal no permite documentar físicamente los criterios de aceptación como parte de una historia de usuario (aspecto "tarjeta" en el modelo 3C). d) No es correcto. Los criterios de aceptación están relacionados con una historia de usuario, no con un plan de prueba. Además, los criterios de aceptación son las condiciones que deben cumplirse para decidir si la historia del usuario está completa. Los riesgos no son tales condiciones.

Pregunta #29**FL-4.4.1 (K2)**

Respuesta correcta: una

a) Es correcto. El concepto básico detrás de la adivinación de errores es que el evaluador intenta adivinar qué errores pudo haber cometido el desarrollador y qué defectos puede haber en el objeto de prueba basándose en experiencias pasadas (y, a veces, en listas de verificación). b) No es correcto. Aunque los evaluadores que solían ser desarrolladores pueden usar su experiencia personal para ayudarlos a adivinar errores, la técnica de prueba no se basa en conocimientos previos de desarrollo. c) No es correcto. La adivinación de errores no es una técnica de usabilidad para adivinar cómo los usuarios

Es posible que no interactúe con el objeto de prueba.

d) No es correcto. Duplicar la tarea de desarrollo tiene varios defectos que la hacen poco práctica, como que el evaluador tenga habilidades equivalentes a las del desarrollador y el tiempo necesario para realizar el desarrollo. No es un error adivinar.

Pregunta #30**FL-1.4.5 (K2)**

Respuesta correcta: a, e

a) Es correcto. Esto lo hacen los probadores. b) No es correcto. La cartera de pedidos del producto es creada y mantenida por el producto. dueño.

c) No es correcto. Esto lo hace el equipo de desarrollo.

d) No es correcto. Esta es una función gerencial. e) Es correcto. Esto lo hacen los probadores.

Pregunta #31

FL-5.1.5 (K3)

Respuesta correcta:

primero debe ir un Test TC 001, seguido del TC 002, para satisfacer las dependencias.

Luego, TC 003 para satisfacer la prioridad y luego TC 004, seguido por TC 005. Por lo tanto:

a) Es correcto. b)

No es correcto. c) No es

correcto. d) No es

correcto.

Pregunta #32

FL-1.3.1 (K2)

Respuesta correcta: una

a) Es correcto. Este principio significa que si las mismas pruebas se repiten una y otra vez, eventualmente estas pruebas ya no encontrarán ningún defecto nuevo. Probablemente esta sea la razón por la que todas las pruebas también pasaron en esta versión.

b) No es correcto. Este principio habla de la creencia errónea de que simplemente encontrar y corregir una gran cantidad de defectos garantizará el éxito de un sistema. c) No es correcto. Este principio dice que una pequeña cantidad de componentes generalmente contienen la mayoría de los defectos.

d) No es correcto. Este principio establece que probar todas las combinaciones de entradas y condiciones previas no es factible.

Pregunta #33

FL-5.2.4 (K2)

Respuesta correcta:c

a) No es correcto. No aceptamos el riesgo; Se proponen acciones concretas. b) No es correcto. No se proponen planes de contingencia. c) Es correcto. Las acciones propuestas están relacionadas con las pruebas, que son una forma de riesgo.

mitigación. d)

No es correcto. El riesgo no se transfiere sino que se mitiga.

Pregunta #34

FL-4.5.3 (K3)

Respuesta correcta: una

a) Es correcto. Esta prueba cubre dos criterios de aceptación: uno sobre la edición del documento y otro sobre guardar cambios.

b) No es correcto. Los criterios de aceptación cubren las actividades del editor, no las actividades del propietario del contenido. c) No

es correcto. Programar el contenido editado para su publicación puede ser una buena idea. característica, pero no está cubierta por los criterios de aceptación.

- d) No es correcto. Los criterios de aceptación establecen la reasignación de un editor al propietario del contenido, no a otro editor.

Pregunta #35

FL-2.2.3 (K2)

Respuesta correcta:b

Debido a que TC1 y TC3 fallaron en la Ejecución 1 [es decir, prueba (1) y prueba (3)], prueba (4) y La prueba (6) son pruebas de confirmación.

Debido a que TC2 y TC3 fallaron en la Ejecución 2 [es decir, pruebas (5) y (6)], prueba (8) y prueba (9) también son pruebas de confirmación.

TC2 pasó en la Ejecución 1 [es decir, prueba (2)], por lo que la prueba (5) es una prueba de regresión.

TC1 pasó en la Ejecución 2 [es decir, prueba (4)], por lo que la prueba (7) también es una prueba de regresión.

Por tanto b es correcto.

Pregunta #36

FL-1.5.2 (K1)

Respuesta correcta: d.

a) No es correcto. El enfoque de automatización de pruebas lo definen los evaluadores con la ayuda de desarrolladores y representantes comerciales.

b) No es correcto. La estrategia de prueba se decide en colaboración con los desarrolladores. c) No es correcto. Los evaluadores, desarrolladores y representantes comerciales forman parte de este enfoque de equipo completo. d) Es correcto. Los evaluadores

trabajarán en estrecha colaboración con representantes comerciales para garantizar que se alcancen los niveles de calidad deseados. Esto incluye apoyarlos y colaborar con ellos para ayudarlos a crear pruebas de aceptación adecuadas.

Pregunta #37

FL-3.2.5 (K1)

Respuesta correcta: d.

a) No es correcto. El tiempo adecuado para las personas es un factor de éxito. b) No es correcto.

Dividir los productos de trabajo en pequeñas partes adecuadas es un éxito factor.

c) No es correcto. Evitar conductas que puedan indicar aburrimiento, exasperación, etc. es un factor de éxito. d) Es correcto. Durante las revisiones

se pueden encontrar defectos, no fallos.

Pregunta #38

FL-1.2.1 (K2)

Respuesta correcta: una

a) Es correcto. Es importante que los evaluadores participen desde el comienzo del ciclo de vida de desarrollo de software (SDLC). Aumentará la comprensión de las decisiones de diseño y detectará defectos tempranamente.

b) No es correcto. Tanto los desarrolladores como los evaluadores comprenderán mejor cada uno de ellos.

productos de trabajo de otros y cómo probar el código.

- c) No es correcto. Si los evaluadores pueden trabajar estrechamente con los diseñadores de sistemas, les dará información sobre cómo realizar la prueba.
- d) No es correcto. Las pruebas no tendrán éxito si no se comprueba el cumplimiento de los requisitos legales.

Pregunta #39

FL-1.1.1 (K1)

Respuesta correcta:c

- a) No es correcto. Es imposible probar que ya no existen defectos en el sistema bajo prueba. Consulte el principio de prueba 1.
- b) No es correcto. Ver principio de prueba 7. c) Es correcto. Las pruebas encuentran defectos y fallas, lo que reduce el nivel de riesgo y al mismo tiempo brinda más confianza en el nivel de calidad del objeto de prueba. d) No es correcto. Es imposible probar todas las combinaciones de entradas (ver prueba principio 2).

Pregunta #40

FL-1.4.2 (K2)

Respuesta correcta:b

- i. Es verdad. El SDLC influye en el proceso de prueba. ii. Es falso. El número de defectos detectados en proyectos anteriores puede tener alguna influencia, pero esto no es tan significativo como i, iii y iv.
- III. Es verdad. Los riesgos del producto identificados son uno de los factores más importantes. influyendo en el proceso de prueba.
- IV. Es verdad. Los requisitos reglamentarios son factores importantes que influyen en la prueba. proceso.
- v. Es falso. El entorno de prueba debe ser una copia del entorno de producción, pero no tiene una influencia significativa en el proceso de prueba.

Por tanto b es correcto.

Preguntas de muestra adicionales: respuestas



Pregunta #A1

FL-1.1.2 (K2)

Respuesta correcta: una

- a) Es correcto. La depuración es el proceso de encontrar, analizar y eliminar los Causas de fallas en un componente o sistema.
- b) No es correcto. Las pruebas son el proceso relacionado con la planificación, preparación y evaluación de un componente o sistema y productos de trabajo relacionados para determinar que satisfacen requisitos específicos, demostrar que son aptos para su propósito y detectar defectos. No está relacionado con solucionar las causas de las fallas. c) No es correcto. La obtención de requisitos es el proceso de recopilar, capturar y consolidar requisitos de fuentes disponibles. No está relacionado con solucionar las causas de las fallas.
- d) No es correcto. La gestión de defectos es el proceso de reconocer, registrar, clasificar, investigar, resolver y eliminar defectos. No está relacionado con solucionar las causas de las fallas.

Pregunta #A2

FL-1.2.2 (K1)

Respuesta correcta: d.

- a) No es correcto. Ver justificación d. b) No es correcto. Ver justificación d. c) No es correcto. Ver justificación d. d) Es correcto. Las pruebas y el control de calidad no son lo mismo. Las pruebas son el proceso que consta de todas las actividades del ciclo de vida de desarrollo de software (SDLC), tanto estáticas como dinámicas, relacionadas con la planificación, preparación y evaluación de un componente o sistema y productos de trabajo relacionados para determinar que satisfacen requisitos específicos, para demostrar que son aptos para su propósito y para detectar defectos. El aseguramiento de la calidad se centra en establecer, introducir, monitorear, mejorar y adherirse a los procesos relacionados con la calidad.

Pregunta #A3

FL-1.2.3 (K2)

Respuesta correcta: d.

- a) No es correcto. La causa principal es la distracción que experimentó el programador mientras programaba.
- b) No es correcto. Aceptar entradas no válidas es un fracaso. c) No es correcto. El error es el pensamiento erróneo que resultó en poner el defecto en el código. d)

Es correcto. El problema en el código es un defecto.

Pregunta #A4

FL-1.4.3 (K2)

Respuesta correcta: d

El software de prueba considerado es una carta de prueba. Las cartas de prueba son el resultado del diseño de la prueba. Por tanto d es correcto.

Pregunta #A5

FL-1.4.4 (K2)

Respuesta correcta:c

- a) No es correcto. La realización del análisis de impacto no proporcionará información sobre la integridad de las pruebas. Analizar el análisis de impacto de los cambios ayudará a seleccionar los casos de prueba adecuados para su ejecución.
- b) No es correcto. La trazabilidad no proporciona información sobre el nivel estimado de riesgo residual si los casos de prueba no se remontan a riesgos.
- c) Es correcto. Realizar el análisis de impacto de los cambios ayuda a seleccionar los casos de prueba para la prueba de regresión. d) No es correcto.

Analizar la trazabilidad entre la base de prueba, los objetos de prueba y los casos de prueba no ayuda a seleccionar datos de prueba para lograr la cobertura supuesta del objeto de prueba. La selección de datos de prueba está más relacionada con el análisis y la implementación de la prueba, no con la trazabilidad.

Pregunta #A6

FL-1.5.3 (K2)

Respuesta correcta: d.

- a) No es correcto. La calidad debe ser responsabilidad de todos los que trabajan en el proyecto y no es responsabilidad exclusiva del equipo de prueba.
 - b) No es correcto. En primer lugar, no es un beneficio si un equipo de pruebas externo no cumple con los plazos de entrega y, en segundo lugar, no hay razón para creer que los equipos de pruebas externos sentirán que no tienen que cumplir con plazos de entrega estrictos. c) No es correcto.
- Es una mala práctica que el equipo de pruebas trabaje en completo aislamiento y esperaríamos que un equipo de pruebas externo se ocupara de cambiar los requisitos del proyecto y comunicarse bien con los desarrolladores. d) Es correcto. Las especificaciones nunca son perfectas, lo que significa que el desarrollador deberá hacer suposiciones. Un evaluador independiente es útil porque puede

Cuestionar y verificar las suposiciones y posterior interpretación realizadas por el desarrollador.

Pregunta #A7

FL-2.1.1 (K2)

Respuesta correcta: una

- a) Es correcto. En los modelos de desarrollo secuencial, en las fases iniciales, los evaluadores participan en revisiones de requisitos, análisis de pruebas y diseño de pruebas. El código ejecutable generalmente se crea en las fases posteriores, por lo que las pruebas dinámicas no se pueden realizar en las primeras etapas del SDLC. b) No es correcto. Las pruebas estáticas siempre se pueden realizar al principio del SDLC. c) No es correcto. La planificación de pruebas debe realizarse temprano en el SDLC antes de que comienza el proyecto de prueba.
- d) No es correcto. Las pruebas de aceptación se pueden realizar cuando hay un producto en funcionamiento. En los modelos SDLC secuenciales, el producto funcional generalmente se entrega tarde en el SDLC.

Pregunta #A8

FL-2.1.4 (K2)

Respuesta correcta:c

- i. Es verdad. Un lanzamiento más rápido del producto y un tiempo de comercialización más rápido es una ventaja de DevOps.
- ii. Es falso. Normalmente, necesitamos menos esfuerzo para las pruebas manuales debido al uso de herramientas de prueba. automatización.
- III. Es verdad. La disponibilidad constante de software ejecutable es una ventaja. IV. Es falso. Se necesitan más pruebas de regresión. v. Es falso. No todo está automatizado y se debe configurar un marco de automatización de pruebas es caro.

Por tanto c es correcto.

Pregunta #A9

FL-2.2.2 (K2)

Respuesta correcta:b

- a) No es correcto. El hecho de que el requisito sobre el rendimiento del sistema provenga directamente del cliente y que el rendimiento sea importante desde el punto de vista empresarial (es decir, alta prioridad) no hace que estas pruebas sean funcionales, porque no verifican "qué" hace el sistema. , sino "cómo" (es decir, qué tan rápido se procesan los pedidos). b) Es correcto. Este es un ejemplo de prueba de rendimiento, un tipo de prueba no funcional.
- pruebas.
- c) No es correcto. Según el escenario, no sabemos si interactuar con la interfaz de usuario es parte de las condiciones de prueba. Pero incluso si lo hicieramos, el principal objetivo de estas pruebas es comprobar el rendimiento, no la usabilidad.

- d) No es correcto. No necesitamos conocer la estructura interna del código para realizar las pruebas de rendimiento.
Se pueden ejecutar pruebas de eficiencia del desempeño sin conocimientos estructurales.

Pregunta #A10

FL-2.3.1 (K2)

Respuesta correcta: una

- a) Es correcto. Cuando se retira un sistema, esto puede requerir pruebas de migración de datos,
que es una forma de prueba de mantenimiento.
- b) No es correcto. Las pruebas de regresión verifican si una solución afectó accidentalmente el comportamiento de
otras partes del código, pero ahora estamos hablando de migración de datos a un nuevo sistema. c) No es
correcto. Las pruebas
de componentes se centran en componentes individuales de hardware o software, no en la migración de datos. d)
No es correcto. Las pruebas de integración se
centran en las interacciones entre componentes y/o sistemas, no en la migración de datos.

Pregunta #A11

FL-3.1.1 (K1)

Respuesta correcta: c

Solo no se puede revisar el código ejecutable de terceros. Por tanto la respuesta correcta es c.

Pregunta #A12

FL-3.1.3 (K2)

Respuesta correcta: d.

- i. Estos comportamientos son fácilmente detectables mientras el software se está ejecutando. Por eso,
Se utilizarán pruebas dinámicas para identificarlos.
- ii. Este es un ejemplo de desviaciones de los estándares, que es un defecto típico que se
más fácil de encontrar con pruebas
estáticas. III. Si el software se ejecuta durante la prueba, se trata de una prueba dinámica.
- IV. Identificar defectos lo antes posible es el objetivo de las pruebas tanto estáticas como dinámicas.
- v. Este es un ejemplo de lagunas en la trazabilidad o cobertura de la base de la prueba, que es un defecto típico
que se encuentra más fácilmente con las pruebas estáticas.

Por tanto d es correcto.

Pregunta #A13

FL-3.2.2 (K2)

Respuesta correcta:b

- a) No es correcto. En todo tipo de revisiones hay más de un rol, incluso en
los informales. b)

Es correcto. Hay varias actividades durante el proceso de revisión formal.

- c) No es correcto. La documentación a revisar debe distribuirse lo antes posible.
- d) No es correcto. Se deben informar los defectos encontrados durante la revisión.

Pregunta #A14

FL-3.2.3 (K1)

Respuesta correcta:b

- a) No es correcto. Esta es la tarea del líder de revisión. b) Es correcto. Esta es la tarea de la dirección en una revisión formal. c) No es correcto. Esta es la tarea del moderador. d) No es correcto. Ésta es la tarea del escriba.

Pregunta #A15

FL-4.2.2 (K3)

Respuesta correcta:

c Hay tres particiones de equivalencia: {..., 10, 11}, {12} y {13, 14, ...}. Los valores de frontera son 11, 12 y 13. En el análisis del valor de frontera de tres puntos para cada frontera, necesitamos probar la frontera y sus dos vecinos, entonces:

- Para 11, probamos 10, 11, 12. • Para 12, probamos 11, 12, 13. • Para 13, probamos 12, 13, 14.

En total, necesitamos probar 10, 11, 12, 13 y 14.

- a) No es correcto. b)

No es correcto. c) Es correcto. d) No es correcto.

Pregunta #A16

FL-4.3.2 (K2)

Respuesta correcta: d.

- a) No es correcto. En este caso, todavía se necesita un caso de prueba ya que hay al menos una rama (incondicional) por cubrir. b) No es correcto.

Cubrir sólo ramas incondicionales no implica cubrir todas las ramas condicionales.

- c) No es correcto. El 100% de cobertura de sucursal implica el 100% de cobertura del estado de cuenta, no de otro modo. Por ejemplo, para una decisión IF sin ELSE, una prueba es suficiente para lograr una cobertura de declaración del 100%, pero solo logra una cobertura de sucursal del 50%. d) Es correcto. Cada resultado de decisión corresponde a una rama condicional, por lo que una cobertura de rama del 100 % implica una cobertura de decisión del 100 %.

Pregunta #A17

FL-4.4.3 (K2)

Respuesta correcta:c

- a) No es correcto. El libro proporciona orientación general y no es un documento de requisitos formales, una especificación o un conjunto de casos de uso, historias de usuarios o negocios.
- procesos.
- b) No es correcto. Si bien se podría considerar la lista como un conjunto de cartas de prueba, se parece más a la lista de condiciones de prueba que se deben verificar. c) Es correcto. La lista de mejores prácticas de interfaz de usuario es la lista de condiciones de prueba para ser controlado sistemáticamente.
- d) No es correcto. Las pruebas no se centran en fallos que puedan ocurrir sino en el conocimiento de lo que es importante para el usuario, en términos de usabilidad.

Pregunta #A18

FL-4.5.1 (K2)

Respuesta correcta:b

- a) No es correcto. La redacción colaborativa de historias de usuario significa que todas las partes interesadas crean las historias de usuario de forma colaborativa para obtener una visión compartida.
- b) Es correcto. La escritura colaborativa de historias de usuario significa que todas las partes interesadas crean el historias de usuarios de forma colaborativa, para obtener la visión compartida.
- c) No es correcto. La redacción colaborativa de historias de usuario significa que todas las partes interesadas crean las historias de usuario de forma colaborativa para obtener una visión compartida.
- d) No es correcto. Esta es la lista de propiedades que debe tener cada historia de usuario, no la descripción del enfoque basado en colaboración.

Pregunta #A19

FL-5.1.1 (K2)

Respuesta correcta: d.

- a) No es correcto. El párrafo contiene información sobre los niveles de prueba y los criterios de salida, que forman parte del enfoque de prueba. b) No es correcto.
El párrafo contiene información sobre los niveles de prueba y los criterios de salida, que forman parte del enfoque de prueba. c) No es correcto. El párrafo contiene información sobre los niveles de prueba y los criterios de salida, que forman parte del enfoque de prueba. d) Es correcto. El párrafo contiene información sobre los niveles de prueba y los criterios de salida, que forman parte del enfoque de prueba.

Pregunta #A20

FL-5.1.4 (K3)

Respuesta correcta:b

- a) No es correcto. Esta debe ser una actividad de equipo y no estar anulada por un solo equipo miembro.
- b) Es correcto. Si las estimaciones de las pruebas no son las mismas pero la variación en los resultados es pequeña, se pueden aplicar reglas como "aceptar el número con más votos".

c) No es correcto. Aún no hay consenso ya que algunos dicen 13 y otros dicen 8. d) No es correcto. Una función no debe eliminarse sólo porque el equipo no puede

ponerse de acuerdo sobre las estimaciones de la prueba.

Pregunta #A21

FL-5.1.6 (K1)

Respuesta correcta:b

a) No es correcto. La pirámide de pruebas enfatiza tener una mayor cantidad de pruebas en los niveles de prueba más bajos. b) Es

correcto. No es cierto que cerca de la cima de la pirámide, la automatización de pruebas deba estar
Más formal.

c) No es correcto. Por lo general, las pruebas de componentes y las pruebas de integración de componentes se
automatizan mediante herramientas basadas

en API. d) No es correcto. Para las pruebas del sistema y las pruebas de aceptación, las pruebas automatizadas son
normalmente se crea utilizando herramientas basadas en GUI.

Pregunta #A22

FL-5.2.1 (K1)

Respuesta correcta:c

a) No es correcto. El impacto del riesgo y la probabilidad del riesgo son independientes. b)

No es correcto. El impacto del riesgo y la probabilidad del riesgo son independientes. c) Es

correcto. El impacto del riesgo y la probabilidad del riesgo son independientes. d) No

es correcto. Necesitamos ambos factores para calcular el nivel de riesgo.

Pregunta #A23

FL-5.2.2 (K2)

Respuesta correcta: una

i. Riesgo del proyecto.

ii. Riesgo del producto.

III. Riesgo del producto.

IV. Riesgo del proyecto.

v. Riesgo del producto.

Por tanto a es correcto.

Pregunta #A24

FL-5.2.3 (K2)

Respuesta correcta: d.

a) No es correcto. Este es un ejemplo de una actividad de monitoreo de riesgos, no de análisis de riesgos. b) No
es correcto. Este es un ejemplo de una decisión arquitectónica, no relacionada con
pruebas.

c) No es correcto. Este es un ejemplo de cómo realizar un análisis de riesgo cuantitativo y no está relacionado con
la minuciosidad o el alcance de las pruebas. d) Es correcto. Esto
muestra cómo el análisis de riesgos afecta la minuciosidad de las pruebas (es decir, el nivel de detalle).

Pregunta #A25

FL-5.3.1 (K1)

Respuesta correcta: a, d

- a) Es correcto. El número de defectos encontrados está relacionado con la calidad del objeto de prueba.
- b) No es correcto. Esta es la medida de la eficiencia de la prueba, no la calidad del objeto de prueba. c) No es correcto. El número de casos de prueba ejecutados no nos dice nada sobre la calidad; los resultados de las pruebas podrían ser suficientes.
- d) Es correcto. La densidad de defectos está relacionada con la calidad del objeto de prueba. e) No es correcto. El tiempo de reparación es una métrica del proceso. No nos dice nada sobre la calidad del producto.

Pregunta #A26

FL-5.3.2 (K2)

Respuesta correcta:b

- a) No es correcto. Los impedimentos para las pruebas pueden ser de alto nivel y estar relacionados con el negocio, por lo que esta es una información importante para las partes interesadas del negocio. b) Es correcto. Las pruebas de rama son una métrica técnica utilizada por desarrolladores y evaluadores técnicos. Esta información no es de interés para los representantes comerciales. c) No es correcto. El progreso de la prueba está relacionado con el proyecto, por lo que puede resultar útil para los representantes comerciales. d) No es correcto. Los riesgos afectan la calidad del producto, por lo que puede resultar útil para los representantes comerciales.

Referencias

1. ISO/IEC/IEEE 29119-1 - Ingeniería de software y sistemas - Pruebas de software - Parte 1: Conceptos y definiciones, 2022.
2. ISO/IEC/IEEE 29119-2 - Ingeniería de software y sistemas - Pruebas de software - Parte 2: Prueba procesos, 2021.
3. ISO/IEC/IEEE 29119-3 - Ingeniería de software y sistemas - Pruebas de software - Parte 3: Documentación de prueba, 2013.
4. ISO/IEC/IEEE 29119-4 - Ingeniería de software y sistemas - Pruebas de software - Parte 4: Prueba técnicas, 2021.
5. ISO/IEC 25010 - Ingeniería de sistemas y software - Requisitos de calidad de sistemas y software-mentos y evaluación (SQuaRE) - Modelos de calidad de sistemas y software, 2011.
6. ISO/IEC 20246 - Ingeniería de software y sistemas - Revisiones de productos de trabajo, 2017.
7. ISO 31000 - Gestión de Riesgos, 2018.
8. "Probador certificado ISTQB - Programa de estudios de nivel básico v4.0", 2023.
9. "Estructura y reglas del examen ISTQB", 2021.
10. G. Myers, *El arte de las pruebas de software* (John Wiley and Sons, 2011)
11. A. Roman, *Pruebas basadas en el pensamiento. El enfoque más razonable para el control de calidad*, Naturaleza Springer, 2018.
12. J. Buxton y B. Randell, *Redaktorzy Técnicas de ingeniería de software. Informe sobre una conferencia patrocinado por el Comité Científico de la OTAN*, p. 16, 1969.
13. Z. Manna i R. Waldinger, "La lógica de la programación informática", *IEEE Transactions on Ingeniería de software*, volumen 4, n.º 3, págs. 199-229, 1978.
14. B. Boehm, *Economía de la ingeniería de software* (Prentice Hall, 1981)
15. A. Enders, "An Analysis of Errors and Their Causes in System Programs", *IEEE Transactions on Software Engineering*, tom 1, nr 2, págs. 140-149, 1975.
16. B. Beizer, *Técnicas de prueba de software*, Van Nostrand Reinhold, 1990.
17. C. Kaner, J. Bach y B. Pettichord, *Lecciones aprendidas en pruebas de software: una experiencia basada en el contexto Enfoque*, Wiley, 2011.
18. "UML 2.5 - Manual de referencia del lenguaje de modelado unificado", 2017. [En línea]. Disponible: www.omg.org/spec/UML/2.5.1.
19. "Plan de prueba de aceptación", [en línea]. Disponible: <https://ssdip.bip.gov.pl/fobjects/download/13576/zalacznik-nr-1-do-opz-szablon-ptn-pdf.html>.
20. V. Stray, R. Florea i L. Paruch, "Exploración de los factores humanos del probador de software ágil", *Software Quality Journal*, tom 30, n.º 1, págs. 1-27, 2021.

21. L. Crispin y J. Gregory, *Pruebas ágiles: una guía práctica para evaluadores y equipos ágiles*, Pearson Educación, 2008.
22. R. Pressman, *Ingeniería de software. El enfoque de un profesional*, McGraw Hill, 2019.
23. T. Linz, *Pruebas en Scrum: una guía para el aseguramiento de la calidad del software en el mundo ágil* (Rocky Rincón, 2014)
24. G. Adzic, *Especificación con el ejemplo: cómo los equipos exitosos entregan el software adecuado* (Manning Publications, 2011)
25. D. ea Chelimsky, *El libro de RSpec: Desarrollo impulsado por el comportamiento con RSpec*, Cucumber y Friends, The Pragmatic Bookshelf, 2010.
26. M. Gärtner, *ATDD con el ejemplo: una guía práctica para el desarrollo basado en pruebas de aceptación* (Pearson Education, 2011)
27. G. Kim, J. Humble, P. Debois y J. Willis, *El manual de DevOps*, IT Revolution Press, 2016.
28. "Probador certificado ISTQB - Programa de estudios de nivel avanzado - Analista de pruebas", 2021.
29. "Probador certificado ISTQB - Programa de estudios de nivel avanzado - Analista de pruebas técnicas", 2021.
30. "Probador certificado ISTQB - Programa de estudios de nivel avanzado - Probador de seguridad", 2016.
31. C. Jones y O. Bonsignour, *La economía de la calidad del software*, Addison-Wesley, 2012.
32. S. Reid, "Revisões de software utilizando ISO/IEC 20246", 2018. [En línea]. Disponible: <http://www.stureid.info/wp-content/uploads/2018/01/Software-Reviews.pdf>.
33. S. Nazir, N. Fatima i S. Chuprat, "Beneficios de la revisión del código moderno: hallazgos primarios de una revisión sistemática de la literatura", w ICSIM '20: Actas de la 3.^a Conferencia Internacional sobre Ingeniería de Software y Gestión de la Información, 2020.
34. T. Gilb y D. Graham, *Inspección de software*, Addison-Wesley, 1993.
35. M. Fagan, "Diseño e inspección de códigos para reducir errores en el desarrollo de programas", tom 15, n.^o 3, págs. 182-211, 1975.
36. P. Johnson, *Introducción a las revisiones técnicas formales* (University College of London Press, 1996)
37. D. O'Neill, „Experimento nacional de calidad del software. Una lección de medición 1992-1997, 23º Taller Anual de Ingeniería de Software”, Centro de Vuelo Espacial Goddard de la NASA, 1998.
38. K. Wiegers, *Revisões por pares en software: una guía práctica* (Addison-Wesley Professional, 2001)
39. E. contra Veenendaal, *The Testing Practitioner*, UTN Publishers, 2004.
40. ISO, ISO 26262 - Vehículos de carretera - Seguridad funcional, 2011.
41. C. Sauer, "La eficacia de las revisiones técnicas del desarrollo de software: un programa de investigación motivado por el comportamiento", *IEEE Transactions on Software Engineering*, tom 26, nr 1, 2000.
42. "Glosario de términos de prueba de ISTQB", [en línea]. Disponible: <http://glossary/istqb.org>.
43. R. Craig y S. Jaskiel, *Testomg de software sistemático*, Artech House, 2002.
44. L. Copeland, *Guía para profesionales sobre diseño de pruebas de software* (Artech House, 2004)
45. T. Koomen, L. van der Aalst, B. Broekman i M. Vroon, *TMap Next para pruebas basadas en resultados*, Editores UTN, 2006.
46. P. Jorgensen, *Pruebas de software, un enfoque artesanal*, CRC Press, 2014.
47. P. Ammann i J. Offutt, *Introducción a las pruebas de software*, Cambridge University Press, 2016.
48. I. Forgács i A. Kovács, *Diseño de pruebas prácticas: selección de diseños de pruebas tradicionales y automatizados técnicas*, BCS, The Chartered Institute for IT, 2019.
49. G. O'Regan, *Guía concisa para pruebas de software* (Springer Nature, 2019)
50. A. Watson, D. Wallace i T. McCabe, "Pruebas estructuradas: una metodología de prueba utilizando la métrica de complejidad ciclomática", Departamento de Comercio de EE. UU., Administración de Tecnología, NIST, 1996.
51. B. Hetzel, *La guía completa para pruebas de software* (John Wiley and Sons, 1998)
52. A. Whittaker, *Cómo descifrar software* (Pearson, 2002)
53. J. Whittaker y H. Thompson, *Cómo romper la seguridad del software*, Addison-Wesley, 2003.
54. M. Andrews i J. Whittaker, *Cómo romper el software web: pruebas funcionales y de seguridad de Aplicaciones web y servicios web*, Addison-Wesley Professional, 2006.

55. J. Whittaker, Pruebas exploratorias de software. Consejos, Trucos, Recorridos y Técnicas para Guiar el Examen Diseño, Addison-Wesley, 2009.
56. C. Kaner, J. Falk y H. Nguyen, Pruebas de software informático, Wiley, 1999.
57. E. Hendrickson, ¡Explorelo!: Reduzca el riesgo y aumente la confianza con pruebas exploratorias. El programador pragmático (2013)
58. B. Brykczynski, "Una encuesta de listas de verificación de inspección de software", ACM SIGSOFT Software Notas de ingeniería, tom 24, n.º 1, págs. 82-89, 1999.
59. J. Nielsen, "Mejora del poder explicativo de las heurísticas de usabilidad", en Actas de la Conferencia SIGCHI sobre factores humanos en sistemas informáticos: celebración de la interdependencia, 1994.
60. A. Gawande, El manifiesto de la lista de verificación: cómo hacer las cosas bien (Metropolitan Books, 2009)
61. M. Cohn, Historias de usuarios aplicadas al desarrollo de software ágil (Addison-Wesley, 2004)
62. B. Wake, "INVERTIR en buenas historias y tareas INTELIGENTES", 2003. [En línea]. Disponible: <https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>.
63. G. Adzic, Cerrar la brecha de comunicación: especificación con ejemplos y pruebas de aceptación ágiles (Neuri Limited, 2009)
64. D. Jackson, M. Thomas i L. Millett, Redaktorzy, Software para sistemas confiables: ¿evidencia suficiente? Comité de Sistemas de Software Certificablemente Confiables, Consejo Nacional de Investigación, 2007.
65. S. Kan, Métricas y modelos en ingeniería de calidad de software (Addison-Wesley, 2003)
66. L. Westfall, Manual del ingeniero de calidad de software certificado (ASQ Quality Press, 2009)
67. M. Cohn, Tener éxito con Agile: desarrollo de software utilizando Scrum (Addison-Wesley, 2009)
68. B. Marick, "Exploración a través del ejemplo", 2003. [En línea]. Disponible: <http://www.exampler.es/old-blog/2003/08/21.1.html#agile-testing-project-1>.
69. K. Schwaber y M. Beedle, Desarrollo ágil de software con Scrum, Prentice-Hall, 2002.
70. R. Neeham, "Experiencia operativa con el sistema de acceso múltiple de Cambridge", w Computer Ciencia y Tecnología, Publicación de la conferencia, 1969.
71. Pandian, CR (2007) Gestión de Riesgos de Software Aplicada. Una guía para proyectos de software Gerentes, Publicaciones Auerbach, Boca Raton
72. Van Veenendaal, E (ed.) (2012) Pruebas prácticas basadas en riesgos, El enfoque PRISMA, Editores UTN: Países Bajos

Índice

A

Criterios de aceptación (CA), 56, 232
 Desarrollo basado en pruebas de aceptación (ATDD), 77, 92
 Pruebas de aceptación, 110
 Revisión ad hoc, 162
 Cobertura de todos los estados, 203
 Cobertura de todas las transiciones, 204
 Prueba alfa, 112
 anomalía, 146
 Autor (reseñas), 150

B

Desarrollo impulsado por el comportamiento (BDD), 77, 90
 Pruebas beta, 112
 Pruebas de caja negra, 122
 Técnica de prueba de caja negra, 51, 172
 Curva de Boehm, 43
 Análisis de valor límite, 187
 Sucursal, 214
 Cobertura de sucursales, 214
 Pruebas de sucursales, 214–217
 Cheque de amigos, 154
 Error, ver defecto
 Gráfico de quemado, 291

C

Solicitud de cambio, 60
 Lista de verificación, 226
 Revisión basada en listas de verificación, 162

Pruebas basadas en listas de verificación, 226
 Enfoque de prueba basado en la colaboración, 229
 Pruebas de integración de componentes, 103
 Rama condicional, 214
 Gestión de configuración, 292
 Sesgo de confirmación, 64
 Pruebas de confirmación, 124
 Entrega continua, 94
 Despliegue continuo, 94
 Integración continua, 94
 Pruebas de aceptación contractual, 111
 Directiva de control, 55
 Cobertura, 47, 60
 Priorización basada en cobertura, 270
 Artículo de cobertura, 51, 57, 181, 193, 198, 203, 214

D

Tablero, 291
 Depuración, 29
 Minimización de la tabla de decisiones, 198
 Prueba de tabla de decisiones, 194
 Defecto, 29, 36
 Gestión de defectos, 295
 Enmascaramiento de defectos, 183
 Informe de defecto, 56, 59, 295
 Agrupación de defectos, 44
 DevOps, 92–96
 Canalización de DevOps, 94
 Diseño impulsado por dominio, 77
 Conducto, 52, 58, 100
 Pruebas dinámicas, 135

mi

- Cobertura de cada elección, 184
- Pruebas tempranas, 43
- Criterios de entrada, 54, 258
- Partición de equivalencia (EP), 177
- errores, 36
- Error al adivinar, 220
- Hipótesis de error, 171
- Estimando, 260
- Estimación basada en ratios, 261
- Pruebas exhaustivas, 42
- Criterios de salida, 54, 258
- Técnica de prueba basada en la experiencia, 51, 173, 219.
- Pruebas exploratorias, 223
- Extrapolación, 262
- Programación extrema, 77, 88

F

- Facilitador, 150
- Fracaso, 29, 36, 39
- Falso negativo, 39
- Falso positivo, 39
- Fallo, ver defecto
- Desarrollo impulsado por funciones (FDD), 77
- Revisión formal, 145
- Tabla de transición completa, 202
- Requisito funcional, 115
- Pruebas funcionales, 115

GRAM

- Diagrama de Gantt, 268
- Condición de guardia, 200

I

- Análisis de impacto, 129
- Modelo incremental, 77
- Revisión individual, 146
- Revisión informal, 154
- Inspección, 155
- Estrategia de integración, 105
- Pruebas de integración, 103
- Planificación de iteraciones, 257
- Modelo iterativo, 77, 78.

K

- Kanban, 77, 84

I

- TI ajustada, 77
- Pruebas de aceptación de cumplimiento legal, 112

METRO

- Pruebas de mantenimiento, 127
- Gerente (opiniones), 150
- Métrica, 287
- Error, ver Error
- Objeto simulado, 52, 58
- Moderador, ver Facilitador

norte

- Pruebas no funcionales, 117
- Cobertura del conmutador N, 204

oh

- Pruebas de aceptación operativa, 111

PAG

- Revisión de pareja, 154
- Regla de Pareto, 44
- Revisión por pares, 152
- Lectura basada en perspectiva, 163
- Planificación del póquer, 262
- Riesgo del producto, 280
- Riesgo del proyecto, 279
- Modelo de prototipo, 77, 82

q

- Calidad, 34
- Garantía de calidad (QA), 36
- Control de calidad (QC), 36

R

- Registrador, ver Escribano
- Pruebas de regresión, 125
- Planificación de lanzamiento, 257
- Requisitos, 50
- Priorización basada en requisitos, 270
- retrospectiva, 97
- Revisión, 143
- Revisor, 151
- Líder de revisión, 151

Índice

405

Reunión de revisión, 149	nivel alto, 57 nivel
Proceso de revisión, 145	bajo, 58
Riesgo, 55, 279	Finalización de la prueba, 52
Análisis de riesgos, 281	Informe de finalización de la prueba, 60, 288
Evaluación de riesgos, 282	Condición de prueba, 50, 56
Priorización basada en riesgos, 270	Control de prueba, 48, 286
Pruebas basadas en riesgos, 278	Datos de prueba, 51, 57, 58
Control de riesgos, 284	Diseño de pruebas, 51
Identificación de riesgos, 281	Desarrollo basado en pruebas (TDD), 77, 89
Impacto del riesgo, 279	Esfuerzo de prueba, 259
Nivel de riesgo, 279	Entorno de prueba, 52, 57, 58
Probabilidad de riesgo, 279	Probador, 63
Gestión de riesgos, 277	Ejecución de prueba, 52
Matriz de riesgos, 282	Calendario de ejecución de pruebas, 58, 268
Mitigación de riesgos, 284	Prueba primera, 88
Monitoreo de riesgos, 286	Arnés de prueba, 100
Registro de riesgos, 54	Implementación de pruebas, 51
Revisión basada en roles, 163	Pruebas, 27
Causa raíz, 40	Prueba de independencia, 67
S	Principios de prueba, 41
Escenarios y simulacros, 163	Cuadrantes de prueba, 276
Escriba, 150	Nivel de prueba, 99, 122
Melé, 77, 83	Registro de prueba, 59
Modelo secuencial, 77	Responsable de pruebas, 62
Virtualización de servicios, 52, 58	Monitoreo de pruebas, 48, 286
Mayús-izquierda, 43, 96	Objeto de prueba, 27, 99
Simulador, 52, 58	Objetivo de la prueba, 28
Habilidades, 35, 64	Plan de prueba, 48, 54, 253
Ciclo de vida de desarrollo de software (SDLC), 76, 87	Planificación de pruebas, 48, 253
Modelo espiral, 77, 81	Procedimiento de prueba, 51, 58
Cobertura de declaración, 214	Proceso de prueba, 47, 53
Prueba de declaración, 212	Informe de progreso de la prueba, 55, 288
Tabla de estados, 202	Pirámide de prueba, 275
Diagrama de transición de estado, 200, 202	Resultado de la prueba, 39
Pruebas de transición estatal, 199	Guion de prueba, 58
Ánalysis estático, 134	Conjunto de prueba, 58
Pruebas estáticas, 134	Estrategia de prueba, 255
Cobertura estructural, 120	Técnica de prueba, 171
Talón, 52, 58, 100	Tipo de prueba, 99, 122
Subsunción, 218	Software de prueba, 54
Pruebas de integración de sistemas, 103	Estimación de tres puntos, 265 BVA de 3
Pruebas del sistema, 107	valores, 189
t	Herramientas, 307
Tablero de tareas, 291	Trazabilidad, 60 BVA
Velocidad del equipo, 257	de 2 valores, 189
Revisión técnica, 154	Ud.
Ánalysis de pruebas, 49	Rama incondicional, 214
Enfoque de prueba, 253	Proceso unificado (UP), 77, 81
Automatización de pruebas, 307	Caso de uso, 208
Base de prueba, 49	Pruebas de casos de uso, 208
Caso de prueba (TC), 51, 57, 59	Pruebas de aceptación de usuario (UAT), 111
	Historia de usuario (EE. UU.), 229
	Puntos de historia de usuario, 264

V

Validación, 27, 46

Cobertura de transiciones válidas, 203

Verificación, 27, 46

modelo V, 77, 80

Técnica de prueba de caja blanca, 51, 172, 211–219

Enfoque de equipo completo, 66

Delphi de banda ancha, 262

Estructura de desglose del trabajo (EDT), 260

Producto del trabajo, 54, 135

W.

Tutorial, 154

Modelo cascada, 77, 79

Pruebas de caja blanca, 120

Cobertura del interruptor Z 0, 203