

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

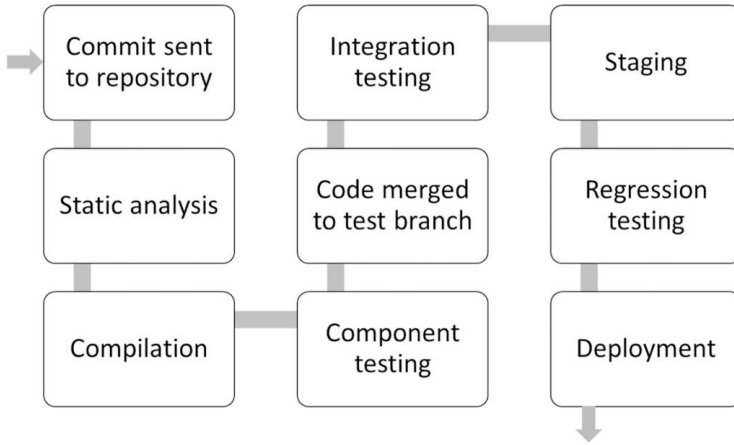


Fig. 2.12 Ejemplo de canal de implementación

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,<sup>2</sup> 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.

<sup>2</sup> La 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 ahorran 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


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 pruebas de referencia. 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.

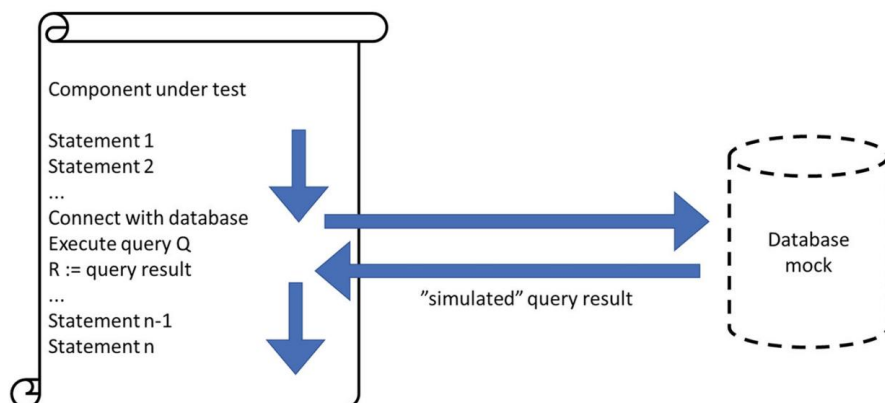


Fig. 2.13 Usando un objeto ficticio

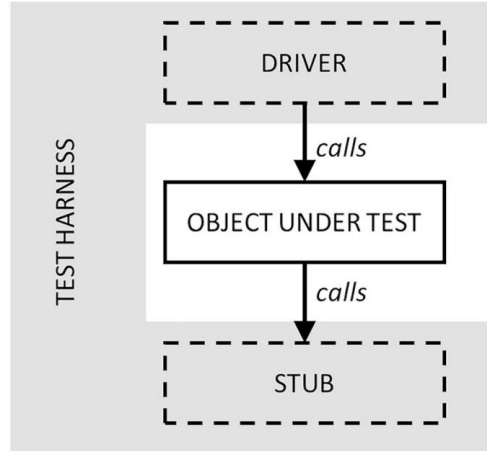
Consideremos el ejemplo de la figura 2.13. Supongamos que estamos probando un script que en algún momento tiene que enviar una consulta a la base de datos, recibir el resultado de esa consulta y procesarla de una determinada manera. En la prueba de componentes, no nos interesará la comunicación del script 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  se 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/nent 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 EI

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,

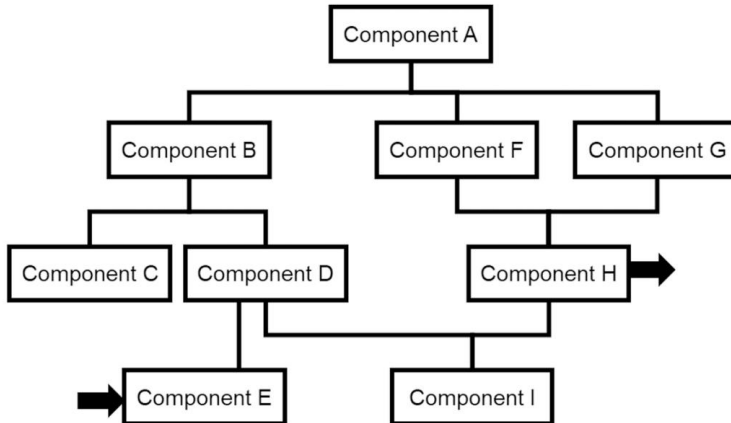


Fig. 2.15 Ejemplo de arquitectura de sistema

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 el  
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

(continuado)



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, al 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 confirman 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 costo mínimos. 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 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 de 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 típicos.

(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 verificara 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 un 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 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:

Requisito 1: cobertura =  $\frac{2}{2+5} = \frac{2}{7} = 28.6\%$

Requisito 2: cobertura =  $\frac{1}{1+5} = \frac{1}{6} = 16.7\%$

Requisito 3: cobertura =  $\frac{2}{2+5} = \frac{2}{7} = 28.6\%$

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



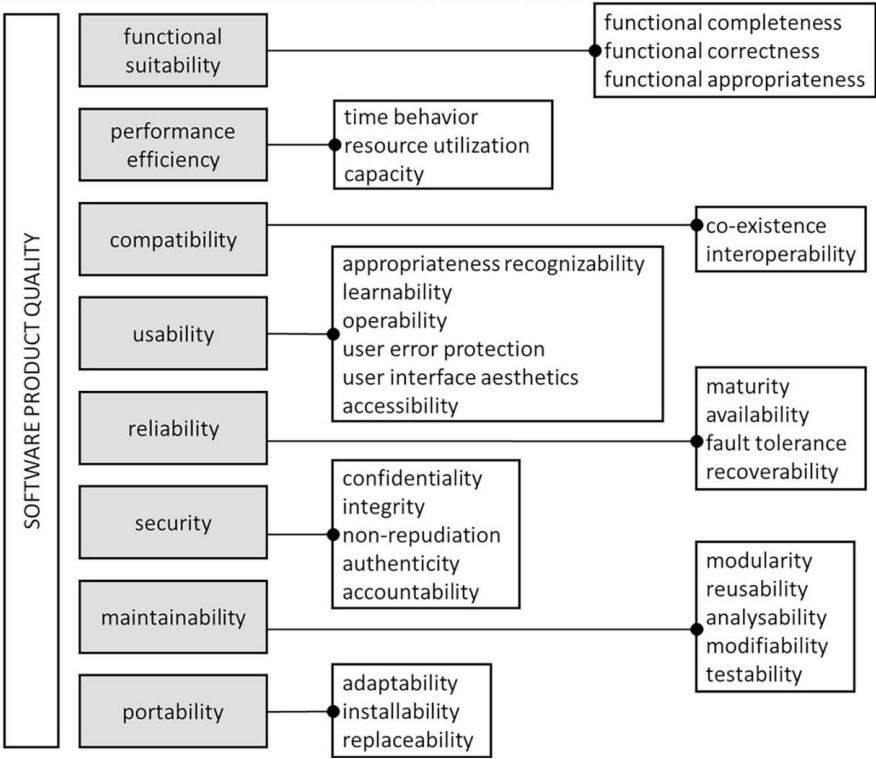


Fig. 2.16 ISO/IEC 25010—Modelos de calidad de sistemas y software (SQuaRE)

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 = tiempo medio hasta el fallo (MTTF) B = 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 $M = T1 - T2$ T1 = tiempo de finalización de la tarea T2 = 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	$M = 1 - A/BA$ = 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 cubierto 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