Probador Certificado del ISTQB®

Programa de Estudio de Nivel Avanzado

Analista de Pruebas Técnicas

V4.0

Traducción realizada por el

Spanish Software Testing Qualifications Board

Versión: V04.00 ES 01.12

Basada en el Programa de Estudio

“Certified Tester - Advanced Level Syllabus, Technical Test Analyst, Version 4.0”

International Software Testing Qualifications Board

Nota sobre Derechos de Propiedad Intelectual

Historial de Revisiones

Versión	Fecha (AAAA/MM/DD)	Observaciones
v4.0	2021/06/30	Entrega de GA para la versión v4.0
v4.0	2021/04/28	Borrador actualizado en base a la retroalimentación de la revisión Beta.
2021 v4.0 Beta	2021/03/01	Borrador actualizado en base a la retroalimentación de la revisión Alfa.
2021 v4.0 Alpha	2020/12/07	Borrador para la revisión Alfa actualizado para: - Mejorar el texto en toda la extensión - Eliminar la subsección asociada a K3 TTA-2.6.1 (2.6 Prueba de camino base) y eliminar la LO - Eliminar la subsección asociada a K2 TTA-3.2.4 (3.2.4 Grafos de llamada) y eliminar la LO - Reescribir la subsección asociada a TTA-3.2.2 (3.2.2 Análisis del flujo de datos) y convertirla en K3 - Reescriba la sección asociada a TTA-4.4.1 y TTA-4.4.2 (4.4. Pruebas de fiabilidad) - Reescribir la sección asociada a la TTA-4.5.1 y TTA-4.5.2 (4.5 Pruebas de rendimiento) - Añada la sección 4.9 sobre perfiles operativos. - Reescribir la sección asociada a la TTA-2.8.1 (sección 2.7 Selección de técnicas de prueba de caja blanca) - Reescriba la TTA-3.2.1 para incluir la complejidad ciclomática (sin impacto en las preguntas del examen) - Reescribir la TTA-2.4.1 (CC/DM) para que sea consistente con otras LO de caja blanca (sin impacto en las preguntas del examen)
2019 v1.0	2019/10/18	Entrega de GA para la versión de 2019
2012	2012/10/19	Entrega de GA para la versión 2012

Índice general

Agradecimientos

Este documento ha sido elaborado por un equipo principal del Subgrupo de Trabajo de Nivel Avanzado - Analista de Pruebas Técnicas de International Software Testing Qualifications Board: Graham Bath (vicepresidente), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (presidente), Erik van Veenendaal.

Las siguientes personas participaron en la revisión, los comentarios y la votación de la versión 2019 de este programa de estudio:

Dani Almog	**Andrew Archer**	**Rex Black**
Armin Born	**Sudeep Chatterjee**	**Tibor Csöndes**
Wim Decoutere	**Klaudia Dusser-Zieger**	**Melinda Eckrich-Brájer**
Peter Foldhazi	**David Frei**	**Karol Frühauf**
Jan Giesen	**Attila Gyuri**	**Matthias Hamburg**
Tamás Horváth	**N. Khimanand**	**Jan te Kock**
Attila Kovács	**Claire Lohr**	**Rik Marselis**
Marton Matyas	**Judy McKay**	**Dénes Medzihradszky**
Petr Neugebauer	**Ingvar Nordström**	**Pálma Polyák**
Meile Posthuma	**Stuart Reid**	**Lloyd Roden**
Adam Roman	**Jan Sabak**	**Péter Sótér**
Benjamin Timmermans	**Stephanie van Dijck**	**Paul Weymouth**

Este documento ha sido elaborado por un equipo principal de International Software Testing Qualifications Board Advanced Level Working Group: Armin Born, Adam Roman, Stuart Reid.

La versión actualizada v4.0 de este documento fue producida por un equipo principal de International Software Testing Qualifications Board Advanced Level Working Group: Armin Born, Adam Roman, Christian Graf, Stuart Reid.

Las siguientes personas participaron en la revisión, los comentarios y la votación de la versión actualizada v4.0 de este programa de estudio:

Adél Vécsey-Juhász	**Jane Nash**	**Pálma Polyák**
Ágota Horváth	**Lloyd Roden**	**Paul Weymouth**
Benjamin Timmermans	**Matthias Hamburg**	**Péter Földházi**
Erwin Engelsma	**Meile Posthuma**	**Jr. Rik Marselis**
Gary Mogyorodi	**Nishan Portoyan**	**Sebastian Małyska**
Geng Chen	**Joan Killeen**	**Tal Pe'er**
Gergely Ágnecz	**Ole Chr. Hansen**	**Wang Lijuan**
		Zuo Zhenlei

El equipo principal agradece al equipo revisor y a los Comités Nacionales por sus aportaciones.

Este documento fue entregado formalmente por la Asamblea General del ISTQB® el 30 de junio de 2021.

Agradecimientos Spanish Software Testing Qualifications Board

Las siguientes personas han participado en la traducción de esta versión del programa de estudio:

Luisa Morales Gómez-Tejedor	**SSTQB**	**España**	**Responsable de revisión de la traducción.**
Gustavo Márquez Sosa	**SSTQB**	**España**	**Responsable de la traducción.**

0. Introducción a Este Programa de Estudio

0.1 Objetivo de este Documento

Este programa de estudio constituye la base de la cualificación internacional de prueba de software en su nivel avanzado para el Analista de Pruebas Técnicas. El ISTQB® proporciona este programa de estudio como sigue:

1. A los Comités Nacionales, para que lo traduzcan a su idioma local y para que acrediten a los proveedores de formación. Los Comités Nacionales pueden adaptar el programa de estudio a sus necesidades lingüísticas particulares y modificar las referencias para adaptarlas a sus publicaciones locales.
2. A las Comités de Examen, para desarrollar preguntas de examen en su idioma local adaptadas a los objetivos de aprendizaje del programa de estudio.
3. A los proveedores de formación, para que elaboren el material didáctico y determinen los métodos de enseñanza adecuados.
4. A los candidatos a la certificación, para preparar el examen (como parte de un curso de formación o de forma independiente).
5. A la comunidad internacional de ingeniería de sistemas y software, para que la actividad de pruebas de software y sistemas avance, y como referencia para la elaboración de libros y artículos.

El ISTQB® podrá autorizar que otras entidades utilicen este programa de estudio con otros fines, siempre y cuando soliciten y obtengan la correspondiente autorización previa por escrito.

0.2 El Probador Certificado de Nivel Avanzado en Prueba de Software

La cualificación de Nivel Avanzado consta de tres programas de estudio distintos relacionados con los siguientes roles:

- Jefe de Prueba.
- Analista de Prueba.
- Analista de Pruebas Técnicas.

La visión general del Analista de Pruebas Técnicas de Nivel Avanzado ISTQB® es un documento separado [CTAL_TTA_OVIEW] que incluye la siguiente información:

- Resultados del negocio.
- Matriz que muestra la trazabilidad entre los resultados del negocio y los objetivos de aprendizaje.
- Resumen.

0.3 Objetivos de Aprendizaje Objeto de Examen

Los Objetivos de Aprendizaje apoyan los resultados de negocio y se utilizan para crear el examen para conseguir la certificación de Analista de Pruebas Técnicas Avanzado.

Los niveles de conocimiento de los objetivos de aprendizaje específicos en los niveles K2, K3 y K4 se muestran al principio de cada capítulo y se clasifican de la siguiente manera:

- K2: Comprender

- K3: Aplicar
- K4: Analizar

0.4 Expectativas de Experiencia

Algunos de los objetivos de aprendizaje para el Analista de Pruebas Técnicas suponen que se dispone de experiencia básica en las siguientes áreas:

- Conceptos generales de programación.
- Conceptos generales de arquitectura de sistemas.

0.5 El Examen de Analista de Pruebas Técnicas de Nivel Avanzado

El examen de Analista de Pruebas Técnicas de Nivel Avanzado se basará en este programa de estudio. Las respuestas a las preguntas del examen pueden requerir el uso de materiales basados en más de una sección de este programa de estudio. Todas las secciones del programa de estudio son objeto de examen, excepto la introducción y los apéndices. Los estándares, libros y otros programas de estudio de ISTQB® se incluyen como referencias, pero su contenido no es objeto de examen más allá de lo que se resume en este programa de estudio.

El formato del examen es de selección múltiple. Consta de 45 preguntas. Para aprobar el examen hay que obtener, como mínimo, el 65% de los puntos totales.

Los exámenes pueden realizarse como parte de un curso de formación acreditado o de forma independiente (por ejemplo, en un centro de exámenes o en un examen público). La compleción de un curso de formación acreditado no es un requisito previo para el examen.

0.6 Requisitos de Acceso al Examen

Para poder presentarse al examen de certificación de Analista de Pruebas Técnicas de Nivel Avanzado se deberá haber obtenido la certificación de Probador Certificado de Nivel Básico.

0.7 Acreditación de Cursos

Un Comité Miembro del ISTQB® puede acreditar a los proveedores de formación cuyo material de curso siga este programa de estudio. Los proveedores de formación deberán obtener las directrices de acreditación del Comité Miembro u organismo que realice la acreditación. Un curso acreditado es reconocido como conforme a este programa de estudio y se le permite tener un examen ISTQB® como parte del curso.

0.8 Nivel de Detalle del Programa de Estudio

El nivel de detalle de este programa de estudio permite que los cursos y los exámenes sean consistentes a nivel internacional. Para lograr este objetivo, el programa de estudio consta de:

Objetivos generales de formación que describen la intención del Analista de Pruebas Técnicas de Nivel Avanzado

Una lista de términos que los estudiantes deben ser capaces de recordar

Objetivos de aprendizaje para cada área de conocimiento, que describen el resultado cognitivo de aprendizaje que debe alcanzarse

Una descripción de los conceptos clave, incluyendo referencias a fuentes como la literatura o los estándares aceptados

El contenido del programa de estudio no es una descripción de toda el área de conocimiento; refleja el nivel de detalle que se debe cubrir en los cursos de formación de nivel avanzado. Se concentra en el material que puede aplicarse a cualquier proyecto de software, utilizando cualquier ciclo de vida de desarrollo del software. El programa de estudio no contiene objetivos de aprendizaje específicos relacionados con ningún modelo de desarrollo de software en particular, pero sí analiza cómo se aplican estos conceptos en el desarrollo ágil de software, en otros tipos de modelos de desarrollo iterativo e incremental de software y en los modelos de desarrollo secuencial de software.

0.9 Organización del Programa de Estudio

Hay seis capítulos con contenido sujeto a examen. El encabezamiento de cada capítulo especifica el tiempo mínimo para el capítulo; no se proporciona la cronología por debajo del nivel del capítulo. Para los cursos de formación acreditados, el programa de estudio requiere un mínimo de 20 horas de formación, distribuidas en los seis capítulos de la siguiente manera:

- Capítulo 1: Tareas del Analista de Pruebas Técnicas en las Pruebas Basadas en Riesgos - (30 minutos)
- Capítulo 2: Técnicas de Prueba de Caja Blanca - (300 minutos)
- Capítulo 3: Análisis Estático y Dinámico - (180 minutos)
- Capítulo 4: Características de Calidad para la Prueba Técnica - (345 minutos)
- Capítulo 5: Revisiones - (165 minutos)
- Capítulo 6: Herramientas de Prueba y Automatización - (180 minutos)

1. Tareas del Analista de Pruebas Técnicas en las Pruebas Basadas en Riesgos - 30 minutos

Palabras clave	
riesgo de producto	**(“product risk”)**
riesgo de proyecto	**(“project risk”)**
evaluación del riesgo	**(“risk assessment”)**
identificación del riesgo	**(“risk identification”)**
mitigación del riesgo	**(“risk mitigation”)**
prueba basada en el riesgo	**(“risk-based testing”)**

Objetivos de Aprendizaje para “Tareas del Analista de Pruebas Técnicas en las Pruebas Basadas en Riesgos”

1.2 Tareas en la Prueba Basada en el Riesgo

APT-1.2.1	(K2)	Resumir los factores de riesgo genéricos que el analista de pruebas técnicas necesita tener en cuenta normalmente.
APT-1.2.2	(K2)	Resumir las actividades del analista de pruebas técnicas dentro de un enfoque de prueba basado en riesgos.

1.1 Introducción

El Jefe de Prueba tiene la responsabilidad general de establecer y gestionar una estrategia de pruebas basadas en el riesgo[1]. El Jefe de Prueba, con frecuencia, requerirá la participación del Analista de Pruebas Técnicas para garantizar la correcta implementación del enfoque basado en riesgos.

Los Analistas de Pruebas Técnicas trabajan dentro de un marco de prueba basada en el riesgo establecido por el Jefe de Prueba del proyecto. Contribuyen con su conocimiento de los riesgos de producto técnicos inherentes al proyecto, como los riesgos relacionados con la seguridad, la fiabilidad del sistema y el rendimiento. También deben contribuir a la identificación y el tratamiento de los riesgos de proyecto asociados a los entornos de prueba, como la adquisición y la configuración de los entornos de prueba para las pruebas de rendimiento, fiabilidad y seguridad.

1.2 Prueba Basada en el Riesgo

Los Analistas de Pruebas Técnicas participan activamente en las siguientes tareas de pruebas basadas en el riesgo:

- Identificación del riesgo
- Evaluación del riesgo
- Mitigación del riesgo

Estas tareas se realizan de forma iterativa a lo largo del proyecto para hacer frente a los riesgos emergentes y a las prioridades cambiantes, así como para evaluar y comunicar regularmente el estado del riesgo.

1.2.1 Identificación del Riesgo

Al recurrir a la muestra más amplia posible de implicados, el proceso de identificación del riesgo tiene más probabilidades de detectar el mayor número posible de riesgos significativos. Dado que los analistas de pruebas técnicas poseen unas competencias técnicas únicas, son especialmente adecuados para realizar entrevistas con expertos, hacer una tormenta de ideas con sus compañeros y analizar sus experiencias para determinar dónde se encuentran las áreas probables de riesgo de producto. En particular, los analistas de pruebas técnicas colaboran estrechamente con otros implicados, como desarrolladores, arquitectos, ingenieros de operaciones, propietarios de producto, oficinas de apoyo locales, expertos técnicos y técnicos de servicio, para determinar las áreas de riesgo técnico que afectan al producto y al proyecto. Involucrar a otros implicados asegura que se tengan en cuenta todos los puntos de vista y suele ser facilitado por los jefes de prueba.

Los riesgos que puede identificar el analista de pruebas técnicas suelen basarse en las características de calidad del producto [ISO 25010] enumeradas en el capítulo 4 de este programa de estudio.

1.2.2 Evaluación del Riesgo

Mientras que la identificación del riesgo consiste en identificar el mayor número posible de riesgos pertinentes, la evaluación del riesgo es el estudio de esos riesgos identificados para categorizar cada uno de ellos y determinar la probabilidad y el impacto asociados a ellos.

[1] "pruebas basadas en el riesgo" y "pruebas basadas en riesgos" son sinónimos.

La probabilidad de un riesgo de producto suele interpretarse como la probabilidad de que se produzca el fallo en el sistema sujeto a prueba. El Analista de Pruebas Técnicas contribuye a entender la probabilidad de cada riesgo de producto técnico, mientras que el Analista de Pruebas contribuye a entender el impacto potencial sobre el negocio en caso de que se produzca el problema.

Los riesgos de proyecto que se convierten en problemas pueden afectar al éxito general del proyecto. En general, es necesario tener en cuenta los siguientes factores genéricos de riesgo de proyecto:

- Conflicto entre los implicados respecto a los requisitos técnicos.
- Problemas de comunicación resultantes de la distribución geográfica de la organización de desarrollo.
- Herramientas y tecnología (incluidas las competencias pertinentes).
- Presión respecto al tiempo, los recursos y la gestión.
- Falta de un aseguramiento de la calidad previo.
- Altas tasas de cambio de los requisitos técnicos.

Los riesgos de producto que se convierten en problemas pueden dar lugar a un mayor número de defectos. Normalmente, es necesario tener en cuenta los siguientes factores genéricos de riesgo de producto:

- Complejidad de la tecnología.
- Complejidad del código.
- Cantidad de cambios en el código fuente (inserciones, eliminaciones, modificaciones).
- Gran número de defectos encontrados relacionados con las características de calidad técnica (historial de defectos).
- Problemas de interfaz técnica y de integración.

Dada la información de riesgo disponible, el Analista de Pruebas Técnicas propone una probabilidad del riesgo inicial de acuerdo con las directrices establecidas por el Jefe de Prueba. El valor inicial puede ser modificado por el Jefe de Prueba cuando se hayan tenido en cuenta todos los puntos de vista de los implicados. Normalmente, el Analista de Prueba determina el impacto del riesgo.

1.2.3 Mitigación del Riesgo

Durante el proyecto, los Analistas de Pruebas Técnicas influyen en la forma en que la prueba responde a los riesgos identificados. Por lo general, esto implica lo siguiente:

- Diseñar casos de prueba para aquellos riesgos que abordan áreas de alto riesgo y ayudar a evaluar el riesgo residual.
- Reducir el riesgo mediante la ejecución de los casos de prueba diseñados y la puesta en marcha de las medidas de mitigación y contingencia adecuadas, tal como se indica en el plan de prueba.
- Evaluar los riesgos basándose en la información adicional recopilada a medida que se desarrolla el proyecto, y utilizar esa información para implementar medidas de mitigación destinadas a disminuir la probabilidad de esos riesgos.

El Analista de Pruebas Técnicas cooperará a menudo con especialistas en áreas como la seguridad y el rendimiento para definir las medidas de mitigación del riesgo y los elementos de la estrategia de prueba. Puede obtener información adicional en los programas de estudios de los especialistas de

ISTQB®, como el programa de estudio de Prueba de Seguridad [CT_SEC_SYL] y el programa de estudio de Prueba de Rendimiento [CT_PT_SYL].

2. Técnicas de Prueba de Caja Blanca - 300 minutos

Palabras Clave

prueba de Interfaz de Programación de Aplicación	**(“API testing”)**
condición atómica	**(“atomic condition”)**
flujo de control	**(“control flow”)**
prueba de decisión	**(“decision testing”)**
prueba de condición/decisión modificada	**(“modified condition/decision testing”)**
prueba de condición múltiple	**(“multiple condition testing”)**
	(“safety integrity level”)
	(“statement testing”)
	(“white-box test technique”)

Objetivos de Aprendizaje para “Técnicas de Prueba de Caja Blanca”

2.2 Prueba de Sentencia

APT-2.2.1	(K3)	Diseñar casos de prueba para un objeto de prueba determinado aplicando pruebas de sentencia para lograr un nivel de cobertura definido.

2.3 Prueba de Decisión

APT-2.3.1	(K3)	Diseñar casos de prueba para un objeto de prueba determinado aplicando la técnica de diseño de prueba de decisión para alcanzar un nivel de cobertura definido.

2.4 Prueba de Condición/Decisión Modificada

APT-2.4.1	(K3)	Diseñar casos de prueba para un objeto de prueba determinado aplicando la técnica de prueba de condición/decisión modificada para lograr una cobertura de condición/decisión modificada completa (CC/DM).

2.5 Prueba de Condición Múltiple

APT-2.5.1	(K3)	Diseñar casos de prueba para un objeto de prueba determinado aplicando la técnica de condición múltiple para lograr un nivel de cobertura definido.

2.6 Prueba de Camino Base (se ha eliminado de la versión v4.0 de este programa de estudio)

APT-2.6.1	(K3)	El APT-2.6.1 ha sido eliminado de la versión v4.0 de este programa de estudio.

2.7 Prueba de Interfaz de Programación de Aplicación

APT-2.7.1	(K3)	Comprender la aplicabilidad de la prueba de Interfaz de Programación de Aplicaciones y los tipos de defectos que detecta.

2.8 Selección de una Técnica de Prueba de Caja Blanca

APT-2.8.1	(K4)	Seleccionar una técnica de prueba de caja blanca adecuada según una situación de proyecto determinada.

2.1 Introducción

Este capítulo describe técnicas de prueba de caja blanca. Estas técnicas se aplican al código y a otras estructuras con un flujo de control, como los gráficos de flujo de los procesos de negocio.

Cada técnica específica permite obtener casos de prueba de forma sistemática y se concentra en un aspecto concreto de la estructura. Los casos de prueba generados por las técnicas satisfacen los criterios de cobertura que se establecen como objetivo y contra los que se miden. Alcanzar una cobertura completa o total (es decir, el 100%) no significa que todo el conjunto de pruebas esté completo, sino que la técnica utilizada ya no sugiere más pruebas útiles para la estructura considerada.

Las entradas de prueba se generan para asegurar que un caso de prueba ejercite una parte concreta del código (por ejemplo, una sentencia o un resultado de la decisión). Determinar las entradas de la prueba que harán que se practique una parte concreta del código puede ser un reto, especialmente si la parte del código que se va a practicar está al final de un subcamino de flujo de control largo con varias decisiones en él. Los resultados esperados se identifican basándose en una fuente externa a esta estructura, como una especificación de requisitos o de diseño u otra base de prueba.

En este programa de estudio se tendrán en cuenta las siguientes técnicas:

- Prueba de sentencia
- Prueba de decisión
- Prueba de condición/decisión modificada
- Prueba de Interfaz de Programación de Aplicación

El programa de estudio de Nivel Básico [CTFL_SYL] introduce la prueba de sentencia y la prueba de decisión. La prueba de sentencia se concentra en practicar las sentencias ejecutables del código, mientras que la prueba de decisión practica los resultados de la decisión.

Las técnicas de condición/decisión modificada y de condición múltiple enumeradas anteriormente se basan en predicados de decisión que contienen múltiples condiciones y encuentran tipos de defectos similares. Por muy complejo que sea un predicado de decisión, se evaluará como VERDADERO o FALSO, lo que determinará el camino recorrido por el código. Se detecta un defecto cuando no se toma el camino previsto porque un predicado de decisión no se evalúa como se esperaba.

Para más detalles sobre la especificación, y ejemplos, de estas técnicas, puede consultar [ISO 29119].

2.2 Prueba de Sentencia

La prueba de sentencia practica las sentencias ejecutables del código. La cobertura se mide como el número de sentencias ejecutadas por las pruebas dividido por el número total de sentencias ejecutables en el objeto de prueba, normalmente expresado como un porcentaje.

Aplicabilidad

Lograr una cobertura de sentencia completa debe considerarse como un mínimo para todo el código que se prueba, aunque esto no siempre es posible en la práctica.

Limitaciones/Dificultades

Debe tenerse en cuenta que la cobertura completa de sentencias debe ser un mínimo para todo el código que se pruebe, aunque esto no siempre es posible en la práctica debido a las limitaciones de tiempo y/o esfuerzo disponibles. Incluso los porcentajes elevados de cobertura de sentencias pueden

no detectar ciertos defectos en la lógica del código. En muchos casos, no es posible lograr una cobertura de sentencia del 100% debido a que el código es inalcanzable. Aunque el código inaccesible no se considera generalmente una buena práctica de programación, puede ocurrir, por ejemplo, si una sentencia de conmutación ("switch") debe tener un caso por defecto, pero todos los casos posibles se tratan de forma explícita.

2.3 Prueba de Decisión

La prueba de decisión practica los resultados de la decisión en el código. Para ello, los casos de prueba siguen los flujos de control de un punto de decisión (por ejemplo, para una sentencia IF, hay un flujo de control para el resultado verdadero y otro para el falso; para una sentencia CASE, puede haber varios resultados posibles; para una sentencia LOOP hay un flujo de control para el resultado verdadero de la condición del bucle y otro para el falso).

La cobertura se mide como el número de resultados de la decisión practicados por las pruebas dividido por el número total de resultados de la decisión en el objeto de prueba, normalmente expresado como un porcentaje. Se debe tener en cuenta que un solo caso de prueba puede practicar varios resultados de la decisión.

En comparación con las técnicas de condición/decisión modificada y de condición múltiple que se describen a continuación, la prueba de decisión considera toda la decisión como un todo y evalúa sólo los resultados VERDADERO y FALSO, independientemente de la complejidad de su estructura interna.

La prueba de rama suele utilizarse indistintamente con la prueba de decisión, ya que con las mismas pruebas se pueden abarcar todas las ramas y cubrir todos los resultados de la decisión. Las pruebas de rama practican las ramas en el código, donde una rama se considera normalmente como una arista del grafo del flujo de control. Para los programas sin decisiones, la definición de cobertura de decisión anterior da como resultado una cobertura de 0/0, que no está definida, independientemente del número de pruebas que se ejecuten, mientras que la rama única desde el punto de entrada hasta el punto de salida (suponiendo un punto de entrada y otro de salida) dará como resultado una cobertura de rama del 100%. Para abordar esta diferencia entre las dos medidas, la norma ISO 29119-4 exige que se ejecute al menos una prueba en el código sin decisiones para lograr una cobertura de decisión del 100%, con lo que la cobertura de decisión del 100% y la cobertura de rama del 100% son equivalentes para casi todos los programas. Muchas herramientas de prueba que proporcionan medidas de cobertura, incluidas las utilizadas para probar sistemas relacionados con la seguridad, emplean un enfoque similar.

Aplicabilidad

Este nivel de cobertura debe tenerse en cuenta cuando el código que se prueba es importante o incluso crítico (véanse las tablas de la sección 2.8.2 para los sistemas relacionados con la seguridad). Esta técnica puede utilizarse para el código y para cualquier modelo que implique puntos de decisión, como los modelos de procesos de negocio.

Limitaciones/Dificultades

Las pruebas de decisión no tienen en cuenta los detalles de cómo se toma una decisión con condiciones múltiples y pueden fallar en la detección de defectos causados por combinaciones de los resultados de la condición.

2.4 Prueba de Condición/Decisión Modificada

En comparación con la prueba de decisión, que considera toda la decisión en su conjunto y evalúa los resultados de VERDADERO y FALSO, la prueba de condición/decisión modificada considera cómo se estructura una decisión cuando incluye múltiples condiciones (cuando una decisión se compone de una sola condición atómica, es simplemente una prueba de decisión).

Cada predicado de decisión se compone de una o más condiciones atómicas, cada una de las cuales se evalúa con un valor booleano. Éstas se combinan lógicamente para determinar el resultado de la decisión. Esta técnica comprueba que cada una de las condiciones atómicas afecta de forma independiente y correcta al resultado de la decisión global.

Cada predicado de decisión se compone de una o más condiciones atómicas, cada una de las cuales se evalúa con un valor booleano. Éstas se combinan lógicamente para determinar el resultado de la decisión. Esta técnica comprueba que cada una de las condiciones atómicas afecta de forma independiente y correcta al resultado de la decisión global.

Esta técnica proporciona un nivel de cobertura más fuerte que la cobertura de sentencia y decisión cuando hay decisiones que contienen múltiples condiciones. Suponiendo N condiciones atómicas únicas e independientes entre sí, el CC/DM para una decisión puede lograrse normalmente practicando la decisión N+1 veces. La prueba de condición/decisión modificada requieren pares de pruebas que muestren que el cambio del resultado de una condición atómica única puede afectar independientemente al resultado de la decisión. Tenga en cuenta que un solo caso de prueba puede ejercitar varias combinaciones de condiciones y, por lo tanto, no siempre es necesario ejecutar N+1 casos de prueba separados para lograr la CC/DM.

Aplicabilidad

Esta técnica se utiliza en las industrias aeroespacial y de la automoción, así como en otros sectores industriales para los sistemas de seguridad física. Se utiliza cuando se prueba software en el que un fallo puede provocar una catástrofe. La prueba de condición/decisión modificada puede ser un término medio razonable entre la prueba de decisión y la prueba de condición múltiple (debido al gran número de combinaciones que hay que probar). Es más rigurosa que la prueba de decisión pero requiere que se practiquen muchas menos condiciones de prueba que la prueba de condición múltiple cuando hay varias condiciones atómicas en la decisión.

Limitaciones/Dificultades

Lograr el CC/DM puede ser complicado cuando hay múltiples apariciones de la misma variable en una decisión con múltiples condiciones; cuando esto ocurre, las condiciones pueden estar "acopladas". Dependiendo de la decisión, puede que no sea posible variar el valor de una condición de manera que sólo ella haga cambiar el resultado de la decisión. Un enfoque para abordar este problema es especificar que sólo se prueben las condiciones atómicas no acopladas mediante la prueba de condición/decisión modificada. El otro enfoque consiste en analizar cada decisión en la que se produce el acoplamiento.

Algunos compiladores y/o intérpretes están diseñados de forma que muestran un comportamiento de cortocircuito cuando evalúan una sentencia de decisión compleja en el código. Es decir, el código que se ejecuta puede no evaluar una expresión completa si el resultado final de la evaluación puede determinarse tras evaluar sólo una parte de la expresión. Por ejemplo, si se evalúa la decisión "A y B", no hay razón para evaluar B si A ya se ha evaluado como FALSO. Ningún valor de B puede cambiar el resultado final, por lo que el código puede ahorrar tiempo de ejecución al no evaluar B. El cortocircuito puede afectar a la capacidad de alcanzar el CC/DM, ya que algunas pruebas requeridas pueden no ser realizables. Por lo general, es posible configurar el compilador para que desactive la optimización de cortocircuito para las pruebas, pero esto puede no estar permitido para las aplicaciones de seguridad crítica, en las que el código probado y el entregado deben ser idénticos.

2.5 Prueba de Condición Múltiple

En raras ocasiones, puede ser necesario probar todas las posibles combinaciones de condiciones atómicas que pueda contener una decisión. Este nivel de prueba se denomina prueba de condición múltiple. Suponiendo N condiciones atómicas únicas e independientes entre sí, la cobertura de condición múltiple completa para una decisión puede lograrse practicándola 2^N veces. Tenga en cuenta que un solo caso de prueba puede practicar varias combinaciones de condiciones y, por lo

tanto, no siempre es necesario ejecutar 2^N casos de prueba distintos para lograr una cobertura de condición múltiple del 100%.

La cobertura se mide como el número de combinaciones de condiciones atómicas practicadas sobre todas las decisiones del objeto de prueba, normalmente expresado como un porcentaje.

Aplicabilidad

Esta técnica se utiliza para probar software de alto riesgo y software embebido del que se espera que funcione de forma fiable sin colapsar durante largos periodos de tiempo.

Limitaciones/Dificultades

Dado que el número de casos de prueba puede obtenerse directamente de una tabla de verdad que contenga todas las condiciones atómicas, este nivel de cobertura puede determinarse fácilmente. Sin embargo, el gran número de casos de prueba necesarios para la prueba de condición múltiple hace que la prueba de condición / decisión modificada sea más factible para situaciones en las que hay varias condiciones atómicas en una decisión.

Si el compilador utiliza el cortocircuitado[2], el número de combinaciones de condiciones que pueden practicarse, a menudo, se reducirá dependiendo del orden y la agrupación de las operaciones lógicas que se realicen sobre las condiciones atómicas.

2.6 Prueba de Camino Base

Este capítulo ha sido eliminado de la versión v4.0 de este programa de estudio.

2.7 Prueba de Interfaz de Programación de Aplicación

Una Interfaz de Programación de Aplicación (IPA por su acrónimo en español o “API” por crónimo en inglés)[3] es una interfaz definida que permite a un programa llamar a otro sistema de software, que le proporciona un servicio, como el acceso a un recurso remoto. Los servicios más habituales son los servicios web, los buses de servicios empresariales, las bases de datos, los mainframes y las Interfaces de Usuario Web.

La prueba de Interfaz de Programación de Aplicación es un tipo de prueba más que una técnica. En ciertos aspectos, la prueba de Interfaz de Programación de Aplicación (“API”) es bastante similar a la prueba de una interfaz gráfica de usuario (IGU). Se concentra en la evaluación de los valores de entrada y los datos devueltos.

Las pruebas negativas suelen ser cruciales cuando se aborda una Interfaz de Programación de Aplicación (“API”). Los programadores que utilizan las interfaces de programación de aplicaciones para acceder a servicios externos a su propio código pueden intentar utilizar las interfaces de las API de formas para las que no fueron concebidas. Eso significa que una gestión robusta de los errores es esencial para evitar un funcionamiento incorrecto. Puede ser necesario realizar pruebas combinatorias de muchas interfaces porque las IPA se utilizan a menudo junto con otras Interfaces de Programación de Aplicación (“API”), y porque una sola interfaz puede contener varios parámetros, cuyos valores pueden combinarse de muchas maneras.

[2] “cortocircuitado” es la traducción del término “short-circuiting”.

[3] “Interfaz de Programación de Aplicaciones (IPA)” es la traducción del términio “Application Programming Interface (“API”)”. Es conveniente observar que el acrónimo del término en inglés se utiliza de forma extensiva En este programa de estudio se utilizará la traducción del término en inglés y el acrónimo en inglés, es decir, **Interfaz de Programación de Aplicaciones (“API”)**.

Con frecuencia, las Interfaces de Programación de Aplicación (“API”) están poco acopladas, lo que da lugar a la posibilidad muy real de que se pierdan transacciones o se produzcan fallos de sincronización. Esto hace necesario probar a fondo los mecanismos de recuperación y reintento. Una organización que proporciona una interfaz IPA debe asegurar que todos los servicios tienen una disponibilidad muy alta; esto suele requerir pruebas estrictas de fiabilidad por parte del editor de la IPA, así como el apoyo de la infraestructura.

Aplicabilidad

La prueba de Interfaz de Programación de Aplicación (“API”) es cada vez más importante para probar sistemas de sistemas a medida que los sistemas individuales se distribuyen o utilizan el procesamiento remoto como forma de descargar parte del trabajo a otros procesadores.

La prueba de Interfaz de Programación de Aplicación (“API”) es cada vez más importante para probar sistemas de sistemas a medida que los sistemas individuales se distribuyen o utilizan el procesamiento remoto como forma de reducir la carga de trabajo derivándola a otros procesadores Algunos ejemplos son:

- Invocaciones de sistemas operativos.
- Arquitecturas Orientadas a Servicio (AOS o “SOA” por su acrónimo en inglés).
- Invocaciones de sistemas operativos.
- Invocaciones a Procedimientos Remotos (IPR o "RPC" por su acrónimo en inglés).
- Servicios web.

El uso de contenedores de software tiene como resultado la división de un programa de software en varios contenedores que se comunican entre sí mediante mecanismos como los mencionados anteriormente. La prueba de Interfaz de Programación de Aplicación (“API”) también deben dirigirse a estas interfaces.

Limitaciones/Dificultades

Probar una Interfaz de Programación de Aplicación (“API”) directamente suele requerir que un Analista de Pruebas Técnicas utilice herramientas especializadas. Dado que normalmente no hay una interfaz gráfica directa asociada a una API, pueden ser necesarias herramientas para configurar el entorno inicial, reunir los datos, invocar la Interfaz de Programación de Aplicación (“API”) y determinar el resultado.

Cobertura

La prueba de Interfaz de Programación de Aplicación (“API”) es una descripción de un tipo de prueba; no denota ningún nivel específico de cobertura. Como mínimo, la prueba de la API debe incluir la realización de llamadas a la IPA tanto con valores de entrada realistas como con entradas inesperadas para comprobar el tratamiento de excepciones. Las pruebas de Interfaz de Programación de Aplicación (“API”) más minuciosas pueden asegurar que se practiquen las entidades invocables al menos una vez, o que se llamen todas las funciones posibles al menos una vez.

La Transferencia de Estado Representativo es un estilo arquitectónico. Los servicios web RESTful permiten a los sistemas solicitantes acceder a los recursos web mediante un conjunto uniforme de operaciones sin estado. Existen varios criterios de cobertura para las Interfaz de Programación de Aplicación (“API”) web RESTful, el estándar de facto para la integración de software [Web-7]. Pueden dividirse en dos grupos: criterios de cobertura de entrada y criterios de cobertura de salida. Entre otros, los criterios de entrada pueden exigir la ejecución de todas las operaciones posibles de la Interfaz de Programación de Aplicación (“API”), el uso de todos los parámetros posibles de la IPA y la cobertura de secuencias de operaciones de la IPA. Entre otros, los criterios de salida pueden requerir

la generación de todos los códigos de estado correctos y erróneos, y la generación de respuestas que contengan recursos que presenten todas las propiedades (o todos los tipos de propiedades).

Tipos de Defectos

Los tipos de defectos que se pueden encontrar al probar las Interfaz de Programación de Aplicación ("API") son bastante dispares. Los problemas de interfaz son comunes, al igual que los problemas de tratamiento de datos, los problemas de sincronización, la pérdida de transacciones, la duplicación de transacciones y los problemas en el tratamiento de excepciones.

2.8 Selección de una Técnica de Prueba de Caja Blanca

La técnica de prueba de caja blanca seleccionada se especifica normalmente en términos de un nivel de cobertura requerido, que se consigue aplicando la técnica de prueba. Por ejemplo, un requisito para lograr una cobertura de sentencia del 100% suele conducir al uso de pruebas de sentencia. Sin embargo, las técnicas de prueba de caja negra suelen aplicarse primero, luego se mide la cobertura y la técnica de prueba de caja blanca sólo se utiliza si no se ha alcanzado el nivel de cobertura de caja blanca requerido. En algunas situaciones, las pruebas de caja blanca pueden utilizarse de manera menos formal para proporcionar una indicación de dónde puede ser necesario aumentar la cobertura (por ejemplo, creando pruebas adicionales cuando los niveles de cobertura de caja blanca son particularmente bajos). Las pruebas de sentencia serían normalmente suficientes para esa medición informal de la cobertura.

Al especificar un nivel de cobertura de caja blanca requerido, es una buena práctica especificarlo sólo al 100%. La razón es que si se exigen niveles de cobertura más bajos, esto significa normalmente que las partes del código que no se ejercitan mediante pruebas son las más difíciles de probar, y estas partes son normalmente también las más complejas y propensas a errores. Así, al solicitar y conseguir, por ejemplo, una cobertura del 80%, puede significar que el código que incluye la mayoría de los defectos detectables se deja sin probar. Por esta razón, cuando los criterios de cobertura de caja blanca se especifican en los estándares, casi siempre se especifican al 100%. Las definiciones estrictas de los niveles de cobertura han hecho que a veces este nivel de cobertura sea impracticable. Sin embargo, las que se dan en la norma ISO 29119-4 permiten descontar de los cálculos los elementos de cobertura no factibles, lo que convierte la cobertura del 100% en un objetivo alcanzable.

Cuando se especifica la cobertura de caja blanca requerida para un objeto de prueba, también es necesario especificarla sólo para un único criterio de cobertura (por ejemplo, no es necesario exigir tanto el 100% de cobertura de sentencia como el 100% de CC/DM). Con los criterios de salida al 100% es posible relacionar algunos criterios de salida en una jerarquía de subsunción, donde se muestra que los criterios de cobertura subsumen a otros criterios de cobertura. Se dice que un criterio de cobertura subsume a otro si, para todos los componentes y sus especificaciones, cada conjunto de casos de prueba que satisface el primer criterio también satisface el segundo. Por ejemplo, la cobertura de rama subsume la cobertura de sentencia porque si se logra la cobertura de rama (al 100%), entonces se garantiza la cobertura de sentencia al 100%. En el caso de las técnicas de prueba de caja blanca tratadas en este programa, podemos decir que la cobertura de rama y de decisión subsume la cobertura de enunciado, CC/DM subsume la cobertura de decisión y de rama, y la cobertura de condición múltiple subsume a CC/DM (si consideramos que la cobertura de rama y de decisión son iguales al 100%, entonces podemos decir que se subsumen mutuamente).

Es importante tener en cuenta que, al determinar los niveles de cobertura de caja blanca que deben alcanzarse para un sistema, es bastante normal definir diferentes niveles para las distintas partes del sistema. Esto se debe a que las distintas partes de un sistema contribuyen de forma diferente al riesgo. Por ejemplo, en un sistema de aviónica, a los subsistemas asociados al entretenimiento en vuelo se les asignaría un nivel de riesgo menor que a los asociados al control de vuelo. La prueba de interfaces es común para todos los tipos de sistemas y normalmente se requiere para todos los niveles de integridad de los sistemas relacionados con la seguridad (para más información sobre los niveles de integridad, véase el apartado 2.8.2). El nivel de cobertura requerido para las pruebas de Interfaz de Programación de Aplicación ("API") aumentará normalmente en función del riesgo

asociado (por ejemplo, el mayor nivel de riesgo asociado a una interfaz pública puede requerir pruebas de API más rigurosas).

La selección de la técnica de prueba de caja blanca que se va a utilizar se basa generalmente en la naturaleza del objeto de prueba y sus riesgos percibidos. Si se considera que el objeto de prueba está relacionado con la seguridad (es decir, que un fallo podría causar daños a las personas o al medio ambiente), son aplicables los estándares normativos, que definirán los niveles de cobertura de caja blanca requeridos (véase el apartado 2.8.2). Si el objeto de prueba no está relacionado con la seguridad, la elección de los niveles de cobertura de caja blanca que deben alcanzarse es más subjetiva, pero debería seguir basándose en gran medida en los riesgos percibidos, como se describe en la sección 2.8.1.

2.8.1 Sistemas No Relacionados con la Seguridad Física

A la hora de seleccionar técnicas de prueba de caja blanca para sistemas no relacionados con la seguridad física, se suelen tener en cuenta los siguientes factores (sin ningún orden de prioridad):

- Contrato - Si el contrato exige que se alcance un nivel de cobertura concreto, el hecho de no alcanzarlo puede dar lugar a un incumplimiento del contrato.
- Cliente - Si el cliente solicita un nivel de cobertura concreto, por ejemplo, como parte de la planificación de la prueba, entonces no alcanzar este nivel de cobertura puede causar problemas con el cliente.
- Norma reglamentaria - Para algunos sectores de la industria (por ejemplo, el financiero) se aplica un estándar reglamentario que define los criterios de cobertura de caja blanca requeridos para los sistemas de misión crítica. Consultar la sección 8.2 para información sobre la cobertura de las normas reglamentarias relativas a los sistemas relacionados con la seguridad física.
- Estrategia de prueba - Si la estrategia de prueba de la organización especifica requisitos para la cobertura de código de caja blanca, no alinearse con la estrategia de la organización puede suponer un riesgo de reprobación por parte de la alta dirección.
- Estilo de codificación - Si el código está escrito sin condiciones múltiples dentro de las decisiones, entonces sería un desperdicio requerir niveles de cobertura de caja blanca tales como CC/DM y cobertura de condición múltiple.
- Información histórica sobre defectos - Si los datos históricos sobre la efectividad de la consecución de un determinado nivel de cobertura sugieren que sería apropiado utilizarlo para este objeto de prueba, sería arriesgado ignorar los datos disponibles. Se debe tener en cuenta que esos datos pueden estar disponibles en el proyecto, la organización o el sector.
- Competencias y experiencia - Si los probadores disponibles para realizar las pruebas no tienen la suficiente experiencia y competencia en una técnica de caja blanca concreta, es posible que se malinterprete y se introduzca un riesgo innecesario si se selecciona esa técnica.
- Herramientas - La cobertura de caja blanca sólo puede medirse en la práctica mediante el uso de herramientas de cobertura. Si no se dispone de dichas herramientas que soporten una medida de cobertura determinada, la selección de esa medida para su realización introduciría un alto nivel de riesgo.

Al seleccionar las pruebas de caja blanca para los sistemas no relacionados con la seguridad física, el Analista de Pruebas Técnicas tiene más libertad para recomendar la cobertura de caja blanca adecuada para los sistemas no relacionados con la seguridad física que para los sistemas relacionados con la seguridad física. Estas elecciones suelen ser un compromiso entre los riesgos percibidos y el coste, los recursos y el tiempo necesarios para tratar estos riesgos mediante prueba

de caja blanca. En algunas situaciones, pueden ser más apropiados otros tratamientos, que podrían aplicarse mediante otros enfoques de prueba de software o de otro tipo (por ejemplo, diferentes enfoques de desarrollo).

2.8.2 Sistemas Relacionados con la Seguridad Física

Cuando el software que se está probando forma parte de un sistema relacionado con la seguridad física, normalmente habrá que utilizar una norma reglamentaria que defina los niveles de cobertura necesarios que deben alcanzarse. Estos estándares suelen exigir que se realice un análisis del peligro para el sistema y los riesgos resultantes se utilizan para asignar niveles de integridad a las diferentes partes del sistema. Los niveles de cobertura requeridos se definen para cada uno de los niveles de integridad.

La IEC 61508 (seguridad funcional de los sistemas programables, electrónicos y relacionados con la seguridad [IEC 61508]) es un estándar internacional general que se utiliza para estos fines. En teoría, podría utilizarse para cualquier sistema relacionado con la seguridad, sin embargo, algunas industrias han creado variantes específicas (por ejemplo, la ISO 26262 [ISO 26262] se aplica a los sistemas de automoción) y algunas industrias han creado sus propios estándares (por ejemplo, DO-178C [DO-178C] para el software aéreo). En el programa de estudio del probador de software de automoción ISTQB® [CT_AuT_SYL] se ofrece información adicional sobre la ISO 26262.

La norma IEC 61508 define cuatro niveles de integridad de la seguridad física[4] (ISF, o “SIL” por su acrónimo en inglés), cada uno de los cuales se define como un nivel relativo de reducción del riesgo proporcionado por una función de seguridad, correlacionado con la frecuencia y severidad de los peligros percibidos. Cuando un objeto de prueba realiza una función relacionada con la seguridad, cuanto mayor sea el riesgo de fallo, mayor será la fiabilidad del objeto de prueba. La siguiente tabla muestra los niveles de fiabilidad asociados a los SIL. Observe que el nivel de fiabilidad para el SIL 4 para el caso de funcionamiento continuo es extremadamente alto, ya que corresponde a un Tiempo Medio Entre Fallos (TMEF) superior a 10.000 años.

IEC 61508 SIL	Operación Continua (probabilidad de un fallo peligroso por hora)	Bajo Demanda (probabilidad de fallo bajo demanda)
1	$\geq 10^{-6}$ a $< 10^{-5}$	$\geq 10^{-2}$ a $< 10^{-1}$
2	$\geq 10^{-7}$ a $< 10^{-6}$	$\geq 10^{-3}$ a $< 10^{-2}$
3	$\geq 10^{-8}$ a $< 10^{-7}$	$\geq 10^{-4}$ a $< 10^{-3}$
4	$\geq 10^{-9}$ a $< 10^{-8}$	$\geq 10^{-5}$ a $< 10^{-4}$

Las recomendaciones para los niveles de cobertura de caja blanca asociados a cada SIL se muestran en la siguiente tabla. Cuando una entrada aparece como "Muy recomendada", en la práctica, alcanzar ese nivel de cobertura se considera normalmente obligatorio. Por el contrario, cuando una entrada sólo se muestra como 'Recomendada', muchos profesionales la tienen en cuenta como opcional y evitan alcanzarla aportando una justificación adecuada. Así, un objeto de prueba asignado a SIL 3 se prueba normalmente para lograr una cobertura de rama del 100% (logra una cobertura de sentencia del 100% de forma automática, como muestra la ordenación de las subsumas).

IEC 61508 SIL	100% de Cobertura de Sentencia	100% Cobertura de Rama	100% CC/DM
1	Recomendado	Recomendado	Recomendado
2	Altamente	Recomendado	Recomendado

[4] “nivel integridad de la seguridad física” es la traducción del término “safety integrity levels”

	recomendado		
3	Altamente recomendado	Altamente recomendado	Recomendado
4	Altamente recomendado	Altamente recomendado	Altamente recomendado

Se debe tener en cuenta que los anteriores SIL y los requisitos sobre los niveles de cobertura de la norma IEC 61508 son diferentes en la norma ISO 26262, y diferentes de nuevo en la norma DO-178C.

3. Análisis Estático y Dinámico - 180 minutos

Palabras Clave

anomalía	**("anomaly")**
análisis del flujo de control	**("control flow analysis")**
complejidad ciclomática	**("cyclomatic complexity")**
análisis del flujo de datos	**("data flow analysis")**
par definición-uso	**("definition-use pair")**
análisis dinámico	**("dynamic analysis")**
fuga de memoria	**("memory leak")**
análisis estático	**("static analysis")**
puntero sin referencia	**("wild pointer")**

Objetivos de Aprendizaje para "Análisis Estático y Dinámico"

3.2 Análisis Estático		
APT-3.2.1	(K3)	Utilizar el análisis del flujo de control para detectar si el código tiene anomalías en el flujo de control y para medir la complejidad ciclomática.
APT-3.2.2	(K3)	Utilizar el análisis del flujo de datos para detectar si el código presenta anomalías en el flujo de datos.
APT-3.2.3	(K3)	Proponer formas de mejorar la mantenibilidad del código aplicando el análisis estático.
APT-3.2.4		Nota: TTA-3.2.4 ha sido eliminado de la versión v4.0 de este programa de estudio.
3.2 Análisis Dinámico		
APT-3.3.1	(K3)	Aplicar el análisis dinámico para lograr un objetivo específico.

3.1 Introducción

El análisis estático (véase el apartado 3.2) es una forma de prueba que se realiza sin ejecutar el software. La calidad del software es evaluada por una herramienta o por una persona basándose en su forma, estructura, contenido o documentación. Esta visión estática del software permite un análisis

detallado sin tener que crear los datos y las precondiciones necesarias para ejecutar los casos de prueba.

Además del análisis estático, las técnicas de prueba estática también incluyen diferentes formas de revisión. Las que son relevantes para el Analista de Pruebas Técnicas se tratan en el capítulo 5.

El análisis dinámico (véase el apartado 3.3) requiere la ejecución real del código y se utiliza para encontrar defectos que se detectan más fácilmente cuando el código se está ejecutando (por ejemplo, las fugas de memoria). El análisis dinámico, al igual que el análisis estático, puede depender de herramientas o de una persona que monitorice el sistema en ejecución en busca de indicadores como un rápido aumento del uso de la memoria.

3.2 Análisis Estático

El objetivo del análisis estático es detectar defectos reales o potenciales en el código y la arquitectura del sistema y para mejorar su mantenibilidad.

3.2.1 Análisis del Flujo de Control

El análisis del flujo de control es una técnica estática en la que se analizan los pasos seguidos por un programa mediante el uso de un grafo del flujo de control, normalmente con el uso de una herramienta. Hay una serie de anomalías que se pueden detectar en un sistema mediante esta técnica, como bucles mal diseñados (por ejemplo, que tengan múltiples puntos de entrada o que no terminen), objetivos ambiguos de invocaciones a funciones en determinados lenguajes, secuenciación incorrecta de las operaciones, código que no se puede alcanzar, funciones no invocadas, etc.

El análisis del flujo de control puede utilizarse para determinar la complejidad ciclomática. La complejidad ciclomática es un número entero positivo que representa el número de caminos independientes en un grafo fuertemente conectado.

La complejidad ciclomática se utiliza generalmente como indicador de la complejidad de un componente. La teoría de Thomas McCabe [McCabe76] afirmaba que cuanto más complejo fuera el sistema, más difícil sería su mantenimiento y más defectos contendría. Muchos estudios han observado esta correlación entre la complejidad y el número de defectos contenidos. Cualquier componente en el que se mida una mayor complejidad debe ser revisado para una posible refactorización, por ejemplo, dividiéndolo en distintos componentes.

3.2.2 Análisis del Flujo de Datos

El análisis del flujo de datos abarca una serie de técnicas que recogen información sobre el uso de variables en un sistema. Se investiga el ciclo de vida de cada variable a lo largo de un camino del flujo de control, (es decir, dónde se declara, se define, se utiliza y se destruye), ya que se pueden identificar posibles anomalías si estas acciones se utilizan fuera de secuencia [Beizer90].

Una técnica común clasifica el uso de una variable como una de tres acciones atómicas:

- cuando se define, declara o inicializa la variable (por ejemplo, x:=3).
- cuando la variable es utilizada o leída (por ejemplo, si x > temp).
- cuando la variable es eliminada, destruida o sale del alcance (por ejemplo, text_file_1.close, variable de control del bucle (i) al salir del bucle).

Entre las secuencias de estas acciones que indican anomalías potenciales se encuentran:

- definición seguida de otra definición o eliminación sin uso intermedio.
- definición sin eliminación posterior (por ejemplo, provocando una posible fuga de memoria para las variables asignadas dinámicamente).
- uso o eliminación antes de la definición.
- uso o eliminación después de una eliminación.

Dependiendo del lenguaje de programación, algunas de estas anomalías pueden ser identificadas por el compilador, pero podría necesitarse una herramienta de análisis estático independiente para identificar las anomalías del flujo de datos. Por ejemplo, la redefinición sin uso intermedio está permitida en la mayoría de los lenguajes de programación, y puede ser programada deliberadamente, pero sería señalada por una herramienta de análisis del flujo de datos como una posible anomalía que debería ser comprobada.

El uso de los caminos del flujo de control para determinar la secuencia de acciones de una variable puede conducir a la notificación de posibles anomalías que no pueden producirse en la práctica. Por ejemplo, las herramientas de análisis estático no siempre pueden identificar si un camino del flujo de control es factible, ya que algunos caminos sólo se determinan en función de los valores asignados a las variables en tiempo de ejecución. También hay una clase de problemas de análisis del flujo de datos que son difíciles de identificar para las herramientas, cuando los datos analizados forman parte de estructuras de datos con variables asignadas dinámicamente, como registros y arreglos. Las herramientas de análisis estático también tienen dificultades para identificar posibles anomalías en el flujo de datos cuando las variables se comparten entre hilos de control concurrentes en un programa, ya que la secuencia de acciones sobre los datos se vuelve difícil de predecir.

En contraposición al análisis del flujo de datos, que es una prueba estática, la prueba del flujo de datos es una prueba dinámica en la que se generan casos de prueba para ejercitar los "pares definición-uso" en el código del programa. Las pruebas de flujo de datos utilizan algunos de los mismos conceptos que el análisis del flujo de datos, ya que estos pares definición-uso son rutas de flujo de control entre una definición y un uso posterior de una variable en un programa.

3.2.3 Uso del Análisis Estático para Mejorar la Mantenibilidad

El análisis estático puede aplicarse de varias maneras para mejorar la mantenibilidad del código, la arquitectura y los sitios web.

El código escrito de forma deficiente, sin comentarios y sin estructura tiende a ser más difícil de mantener. Puede requerir un mayor esfuerzo por parte de los desarrolladores para localizar y analizar los defectos del código, y es probable que la modificación del código para corregir un defecto o añadir una prestación dé lugar a la introducción de nuevos defectos.

El análisis estático se utiliza para verificar el cumplimiento de los estándares de codificación y las directrices; cuando se identifica un código no conforme, se puede actualizar para mejorar su mantenibilidad. Estos estándares y directrices describen las prácticas de codificación y diseño necesarias, como las convenciones de nomenclatura, comentarios, sangría y modularización. Se debe tener en cuenta que las herramientas de análisis estático suelen lanzar avisos en lugar de detectar defectos. Estas advertencias (por ejemplo, sobre el nivel de complejidad) pueden aparecer aunque el código sea sintácticamente correcto.

Los diseños modulares suelen dar como resultado un código más mantenible. Las herramientas de análisis estático apoyan el desarrollo de código modular de las siguientes maneras:

- Buscan código repetido. Estas secciones de código pueden ser candidatas a la refactorización en componentes (aunque la sobrecarga en tiempo de ejecución impuesta por las invocaciones a componentes puede ser un problema para los sistemas en tiempo real).
- Generan métricas que son valiosos indicadores de la modularidad del código. Entre ellas se encuentran las medidas de acoplamiento y cohesión. Un sistema que tiene una buena mantenibilidad es muy probable que tenga una medida baja de acoplamiento (el grado en que los componentes dependen unos de otros durante la ejecución) y una medida alta de cohesión (el grado en que un componente es autónomo y se concentra en una sola tarea).
- En el código orientado a objetos, indican en qué áreas los objetos derivados pueden tener demasiada o muy poca visibilidad en las clases padre.
- Destacan las áreas del código o de la arquitectura con un alto nivel de complejidad estructural.

El mantenimiento de un sitio web también puede apoyarse en herramientas de análisis estático. En este caso, el objetivo es comprobar si la estructura de árbol del sitio está bien equilibrada o si existe un desequilibrio que conduzca a:

- Tareas de prueba más difíciles.
- Una mayor carga de trabajo de mantenimiento.

Además de evaluar la mantenibilidad, las herramientas de análisis estático también pueden aplicarse al código utilizado en la implementación de los sitios web para comprobar la posible exposición a vulnerabilidades de seguridad como la inyección de código, la seguridad de las cookies, secuencias de comandos en sitios cruzados, manipulación de recursos e inyección de código SQL. En la sección 4.3 y en el programa de estudio sobre pruebas de seguridad [CT_SEC_SYL] se ofrecen más detalles al respecto.

3.3 Análisis Dinámico

3.3.1 Resumen

El análisis dinámico se utiliza para detectar fallos cuyos síntomas sólo son visibles cuando se ejecuta el código. Por ejemplo, la posibilidad de que haya fugas de memoria puede detectarse mediante un análisis estático (encontrando código que asigna pero nunca libera memoria), pero una fuga de memoria es fácilmente identificable con análisis dinámico.

Los fallos que no son reproducibles de forma directa (intermitentes) pueden tener consecuencias significativas en el esfuerzo de prueba y en la capacidad de entrega o uso del software de forma productiva. Estos fallos pueden ser causados por fugas de memoria o de recursos, uso incorrecto de punteros y otras anomalías (por ejemplo, de la pila del sistema) [Kaner02]. Debido a la naturaleza de estos fallos, que pueden incluir el empeoramiento gradual del rendimiento del sistema o incluso su fallo total, las estrategias de prueba deben tener en cuenta los riesgos asociados a estos defectos y, en su caso, realizar un análisis dinámico para reducirlos (normalmente mediante el uso de herramientas). Dado que estos fallos suelen ser los más costosos de encontrar y corregir, se recomienda iniciar el análisis dinámico en una etapa temprana del proyecto.

Se puede utilizar el análisis dinámico para lograr lo siguiente:

- Prevenir la ocurrencia de fallos mediante la detección de fugas de memoria (ver sección 3.3.2) y punteros sin referencia (ver sección 3.3.3).
- Analizar los fallos del sistema que no se pueden reproducir fácilmente.
- Evaluar el comportamiento de la red.
- Mejorar el rendimiento del sistema mediante el uso de perfiladores de código para proporcionar información sobre el comportamiento del sistema en tiempo de ejecución que puede utilizarse para realizar cambios informados.

Se puede realizar el análisis dinámico en cualquier nivel de prueba y requiere competencias técnicas y de sistemas para hacer lo siguiente:

- Especificar los objetivos de prueba del análisis dinámico.
- Determinar el momento adecuado para iniciar y detener el análisis.
- Analizar resultados.

Se pueden utilizar herramientas de análisis dinámico incluso si el Analista de Pruebas Técnicas tiene una competencia técnica mínima; las herramientas utilizadas suelen crear registros completos que pueden ser analizados por quienes tienen la competencia técnica y analítica necesaria.

3.3.2 Detección de Fugas de Memoria

Una fuga de memoria se produce cuando se asignan áreas de memoria (RAM) a un programa pero no se liberan con posterioridad, cuando ya no se necesitan. Esta zona de memoria no está disponible para su reutilización. Cuando esto ocurre con frecuencia o en situaciones de memoria reducida, el programa puede quedarse sin memoria utilizable. Históricamente, la manipulación de la memoria era responsabilidad del programador. Cualquier área de memoria asignada dinámicamente tenía que ser liberada por el programa que la asignaba para evitar una fuga de memoria. Muchos entornos de programación modernos incluyen una "recolección de basura" automática o semiautomática en la que la memoria asignada se libera después de su uso sin la intervención directa del programador. Aislar las fugas de memoria puede ser muy difícil en los casos en que la memoria asignada debe ser liberada por los recolectores de basura automáticos.

Las fugas de memoria suelen causar problemas al cabo de un tiempo, cuando una cantidad significativa de memoria se ha perdido y no está disponible. Cuando el software se instala por primera vez, o cuando se reinicia el sistema, la memoria se reasigna, por lo que las fugas de memoria no se notan; las pruebas son un ejemplo en el que la asignación frecuente de memoria puede impedir la detección de fugas de memoria. Por estas razones, los efectos negativos de las fugas de memoria pueden se advierten por primera vez cuando el programa está en producción.

El principal síntoma de una fuga de memoria es un empeoramiento constante del tiempo de respuesta del sistema que, en última instancia, puede resultar en un fallo del mismo. Aunque estos fallos pueden resolverse reiniciando el sistema, esto no siempre es práctico o incluso posible en algunos sistemas.

Muchas herramientas de análisis dinámico identifican las áreas del código en las que se producen fugas de memoria para poder corregirlas. También se pueden monitorizar memorias sencillas para obtener una impresión general de si la memoria disponible está disminuyendo con el tiempo, aunque seguiría siendo necesario un análisis de seguimiento para determinar la causa exacta de la disminución.

3.3.3 Detección de Punteros sin Referencia

Los punteros sin referencia dentro de un programa son punteros que ya no son precisos y no deben utilizarse. Por ejemplo, un puntero sin referencia puede haber "perdido" el objeto o la función a la que debería apuntar o no apunta al área de memoria prevista (por ejemplo, apunta a un área que está más allá de los límites asignados de un arreglo). Cuando un programa utiliza punteros sin referencia, pueden producirse diversas consecuencias, entre las que se incluyen las siguientes:

- El programa puede tener el rendimiento esperado. Este puede ser el caso en el que el puntero sin referencia acceda a una memoria que no está siendo utilizada por el programa y que está teóricamente "libre" y/o contiene un valor razonable.
- El programa puede sufrir un fallo total. En este caso, el puntero sin referencia puede haber provocado el uso incorrecto de una parte de la memoria que es crítica para el funcionamiento del programa (por ejemplo, el sistema operativo).
- El programa no funciona correctamente porque no se puede acceder a los objetos requeridos por el programa. En estas condiciones, el programa puede seguir funcionando, aunque puede emitirse un mensaje de error.
- El puntero puede corromper los datos de la ubicación de memoria y utilizar posteriormente valores incorrectos (esto también puede representar una amenaza para la seguridad).

Se debe tener en cuenta que los cambios en el uso de la memoria del programa (por ejemplo, una nueva construcción tras un cambio en el software) pueden desencadenar cualquiera de las cuatro consecuencias enumeradas anteriormente. Esto es particularmente crítico cuando inicialmente el programa rinde como se esperaba a pesar del uso de punteros sin referencia, y luego se produce un fallo total inesperado (quizás incluso en producción) tras un cambio de software. Las herramientas pueden ayudar a identificar los punteros sin referencia a medida que son utilizados por el programa, independientemente de su impacto en la ejecución del programa. Algunos sistemas operativos tienen funciones incorporadas para comprobar las violaciones de acceso a la memoria durante el tiempo de ejecución. Por ejemplo, el sistema operativo puede lanzar una excepción cuando una aplicación intenta acceder a una ubicación de memoria que está fuera del área de memoria permitida de esa aplicación.

3.3.4 Análisis de la Eficiencia de Desempeño

El análisis dinámico no sólo sirve para detectar fallos y localizar los defectos asociados. Con el análisis dinámico del rendimiento de los programas, las herramientas ayudan a identificar los cuellos

de botella de la eficiencia de desempeño y generan una amplia gama de métricas de rendimiento que pueden ser utilizadas por el desarrollador para ajustar el rendimiento del sistema. Por ejemplo, se puede proporcionar información sobre el número de veces que se llama a un componente durante la ejecución. Los componentes que son llamados con frecuencia son candidatos a mejorar su rendimiento. A menudo se cumple la regla de Pareto: un programa gasta una parte desproporcionada (80%) de su tiempo de ejecución en un pequeño número (20%) de componentes [Andrist20].

El análisis dinámico del rendimiento de los programas suele realizarse mientras se llevan a cabo pruebas de sistema, aunque también puede hacerse al probar un único subsistema en fases anteriores de las pruebas utilizando arneses de prueba. En el programa de estudio sobre pruebas de rendimiento [CT_PT_SYL] se ofrecen más detalles al respecto.

4. Características de Calidad para la Prueba Técnica - 345 minutos

Palabras Clave	
responsabilidad	**(“accountability”)**
adaptabilidad	**(“adaptability”)**
capacidad de ser analizado[5]	**(“analyzability”)**
autenticidad	**(“authenticity”)**
disponibilidad	**(“availability”)**
capacidad	**(“capacity”)**
coexistencia	**(“coexistence”)**
compatibilidad	**(“compatibility”)**
confidencialidad	**(“confidentiality”)**
tolerancia a defectos	**(“fault tolerance”)**
instalabilidad	**(“installability”)**
integridad	**(“integrity”)**
mantenibilidad	**(“maintainability”)**
madurez	**(“maturity”)**
capacidad de ser modificado[6]	**(“modifiability”)**
modularidad	**(“modularity”)**
no repudio	**(“non-repudiation”)**
perfil operativo	**(“operational profile”)**
eficiencia de desempeño	**(“performance efficiency”)**
portabilidad	**(“portability”)**
característica de calidad	**(“quality characteristic”)**
capacidad de recuperación[7]	**(“recoverability”)**
fiabilidad	**(“reliability”)**
modelo de crecimiento de la fiabilidad	**(“reliability growth model”)**
capacidad de ser reemplazado[8]	**(“replaceability”)**
uso de recursos	**(“resource utilization”)**
capacidad de reutilización	**(“reusability”)**
seguridad	**(“security”)**
capacidad de ser probado[9]	**(“testability”)**

[5] “capacidad de ser analizado” antes “analizabilidad”.

[6] “capacidad para ser modificado” antes “modificabilidad”.

[7] “capacidad de recuperación” antes “recuperabilidad”.

[8] “capacidad de ser reemplazado” antes “reemplazabilidad”.

[9] “capacidad de ser probado” antes “testabilidad”.

Palabras Clave

comportamiento temporal ("time behavior")

Objetivos de Aprendizaje para "Características de Calidad para la Prueba Técnica"

4.2 Cuestiones Generales Relativas a la Planificación

APT-4.2.1	(K4)	Dado un escenario concreto, analizar los requisitos no funcionales y redactar las secciones respectivas del plan de prueba.
APT-4.2.2	(K3)	Dado un riesgo de producto concreto, definir el tipo o tipos de pruebas no funcionales particulares más adecuados.
APT-4.2.3	(K2)	Comprender y explicar las etapas del ciclo de vida del desarrollo de software de una aplicación en las que normalmente deberían aplicarse las pruebas no funcionales.
APT-4.2.4	(K3)	Dado un escenario determinado, definir los tipos de defectos que se espera encontrar utilizando los diferentes tipos de pruebas no funcionales.

4.3 Prueba de Seguridad

APT-4.3.1	(K2)	Explicar las razones para incluir la prueba de seguridad en un enfoque de prueba.
APT-4.3.2	(K2)	Explicar los principales aspectos que se deben tener en cuenta en la planificación y especificación de la prueba de seguridad.

4.4 Prueba de Fiabilidad

APT-4.4.1	(K2)	Explicar las razones para incluir la prueba de fiabilidad en un enfoque de prueba.
APT-4.4.2	(K2)	Explicar los principales aspectos que se deben tener en cuenta en la planificación y especificación de las pruebas de fiabilidad.

4.5 Prueba de Rendimiento

APT-4.5.1	(K2)	Explicar las razones para incluir la prueba de rendimiento en un enfoque de prueba.
APT-4.5.2	(K2)	Explicar los principales aspectos que se deben tener en cuenta en la planificación y especificación de las pruebas de rendimiento.

4.6 Prueba de Mantenibilidad

APT-4.6.1	(K2)	Explicar las razones para incluir la prueba de mantenibilidad en un enfoque de prueba.

4.7 Prueba de Portabilidad

APT-4.7.1	(K2)	Explicar las razones para incluir la prueba de portabilidad en un enfoque de prueba.

4.8 Prueba de Compatibilidad

APT-4.8.1	(K2)	Explicar las razones para incluir la prueba de coexistencia en un enfoque de prueba.

4.1 Introducción

En general, el Analista de Pruebas Técnicas concentra la prueba en "cómo" funciona el producto, más que en los aspectos funcionales de "qué" hace. Estas pruebas pueden tener lugar en cualquier nivel de prueba. Por ejemplo, durante la prueba de componente para sistemas en tiempo real y embebidos, es importante realizar una evaluación comparativa de la eficiencia de rendimiento y probar el uso de recursos. Durante la prueba de aceptación operativa y la prueba de sistema, es conveniente probar los aspectos de fiabilidad, como la capacidad de recuperación. Las pruebas a este nivel tienen como objetivo probar un sistema específico, es decir, combinaciones de hardware y software. El sistema específico sujeto a prueba puede incluir varios servidores, clientes, bases de datos, redes y otros recursos. Independientemente del nivel de prueba, las pruebas deben realizarse en función de las prioridades de riesgo y los recursos disponibles.

Tanto las pruebas dinámicas como las estáticas, incluidas las revisiones (véanse los capítulos 2, 3 y 5), pueden aplicarse para probar las características de calidad no funcionales descritas en este capítulo.

La descripción de las características de calidad de producto recogida en la norma ISO 25010 [ISO 25010] sirve de guía para las características y sus subcaracterísticas. Éstas se muestran en la siguiente tabla, junto con una indicación de qué características/subcaracterísticas están cubiertas por los programas de estudio de Analista de Pruebas y Analista de Pruebas Técnicas.

Característica	Subcaracterísticas	Analista de Prueba	Analista de Prueba Técnicas
Adecuación Funcional	• Completitud funcional. • Corrección funcional. • Pertinencia funcional.	**X**	
Fiabilidad	• Disponibilidad. • Madurez. • Recuperabilidad. • Tolerancia a defectos.		**X**
Usabilidad	• Accesibilidad. • Capacidad de reconocimiento de la pertinencia. • Capacidad de ser aprendido[10]. • Capacidad de ser operado[11]. • Estética de interfaz de usuario. • Protección ante error de usuario.	**X**	
Eficiencia de Desempeño	• Capacidad. • Comportamiento temporal. • Utilización de recursos.		**X**
Mantenibilidad	• Modularidad. • Capacidad de reutilización. • Capacidad de ser analizado[12]. • Capacidad de ser modificado[13]. • Capacidad de ser probado[14].		**X**
Portabilidad	• Adaptabilidad. • Instalanbilidad[15]. • Capacidad de ser reemplazado[16].	**X**	**X**
Seguridad	• Confidencialidad. • Integridad. • No repudio. • Responsabilidad. • Autenticidad.		**X**
Compatibilidad	• Coexistencia.		**X**
	• Interoperabilidad.	**X**	

Obsérvese que en el Apéndice A se incluye una tabla que compara las características descritas en la norma ISO 9126-1, actualmente cancelada (tal y como se utilizaba en la versión 2012 de este programa de estudio), con las de la norma ISO 25010, más reciente.

[10] "capacidad de ser aprendido" o "aprendibilidad".

[11] "Capacidad de ser operado" o "operabilidad".

[12] "capacidad de ser analizado" o "analizabilidad".

[13] "capacidad para ser modificado" o "modificabilidad".

[14] "capacidad de ser probado" o "testabilidad".

[15] "instalanbilidad" o "capacidad de ser instalado".

[16] "Capacidad de ser reemplazado" o "reemplazabilidad".

Para todas las características y subcaracterísticas de calidad tratadas en esta sección, deben reconocerse los riesgos típicos para poder establecer y documentar un enfoque de prueba adecuado. Las pruebas de las características de calidad requieren una atención especial respecto a la cronología del ciclo de vida, las herramientas necesarias, los estándares requeridos, la disponibilidad del software y la documentación y los conocimientos técnicos. Si no se planifica un enfoque para hacer frente a cada característica y a sus necesidades únicas de prueba, el probador puede no disponer de un tiempo adecuado para la planificación, la preparación y la ejecución de las pruebas dentro del calendario.

Algunas de estas pruebas, por ejemplo, la prueba de rendimiento, requieren una amplia planificación, equipos dedicados, herramientas específicas, competencias de prueba especializadas y, en la mayoría de los casos, una cantidad significativa de tiempo. Las pruebas de las características y subcaracterísticas de calidad deben integrarse en el calendario general de la prueba con los recursos adecuados asignados al esfuerzo.

Mientras que el Jefe de Prueba se ocupará de recopilar y comunicar la información de las métricas resumidas relativas a las características y subcaracterísticas de calidad, el Analista de Pruebas o el Analista de Pruebas Técnicas (según la tabla anterior) reúne la información para cada métrica.

Las mediciones de las características de calidad recopiladas por el Analista de Pruebas Técnicas en las pruebas de preproducción pueden constituir la base de los Acuerdos de Nivel de Servicio (ANS) entre el proveedor y los implicados (por ejemplo, clientes, operadores) del sistema de software. En algunos casos, las pruebas pueden seguir ejecutándose después de que el software haya entrado en producción, a menudo por parte de un equipo u organización independiente. Esto suele ocurrir con las pruebas de rendimiento y de fiabilidad, que pueden mostrar resultados diferentes en el entorno de producción que en el de prueba.

4.2 Cuestiones Generales Relativas a la Planificación

No planificar las pruebas no funcionales puede poner en riesgo considerable el éxito de un proyecto. El Jefe de Prueba puede solicitar al Analista de Pruebas Técnicas que identifique los principales riesgos para las características de calidad pertinentes (véase la tabla de la sección 4.1) y que aborde cualquier problema de planificación asociado a las pruebas propuestas. Esta información puede utilizarse para crear el plan maestro de prueba.

Al realizar estas tareas se tienen en cuenta los siguientes factores generales:

- Requisitos de los implicados.
- Requisitos del entorno de prueba.
- Necesidad de adquisición de herramientas y de formación.
- Consideraciones relativas a la organización.
- Consideraciones sobre la seguridad de los datos.

4.2.1 Requisitos de los Implicados

Los requisitos no funcionales suelen estar mal especificados o incluso son inexistentes. En la fase de planificación, los analistas de pruebas técnicas deben ser capaces de obtener los niveles de expectativas relativos a las características de calidad técnica de los implicados y evaluar los riesgos que representan.

Un enfoque común es asumir que si el cliente está satisfecho con la versión existente del sistema, seguirá estándolo con las nuevas versiones, siempre que se mantengan los niveles de calidad alcanzados. Esto permite utilizar la versión existente del sistema como punto de referencia. Este enfoque puede ser especialmente útil en el caso de algunas características de calidad no funcionales,

como la eficiencia de desempeño, en las que los implicados pueden tener dificultades para especificar sus requisitos.

Es aconsejable obtener múltiples puntos de vista a la hora de capturar los requisitos no funcionales. Se deben educir[17] a partir de los implicados, como los clientes, los propietarios de producto, los usuarios, el personal de operaciones y el personal de mantenimiento. Si se excluye a los principales implicados, es probable que se pasen por alto algunos requisitos. Para más detalles sobre la captura de requisitos, consulte el programa de estudio del Jefe de Prueba Avanzado [CTAL_TM_SYL].

En el desarrollo ágil de software, los requisitos no funcionales pueden establecerse como historias de usuario o añadirse a la funcionalidad especificada en los casos de uso como restricciones no funcionales.

4.2.2 Requisitos del Entorno de Prueba

Muchas pruebas técnicas (por ejemplo, pruebas de seguridad, pruebas de fiabilidad, pruebas de eficiencia de desempeño) requieren un entorno de prueba similar al de producción para proporcionar medidas realistas. Dependiendo del tamaño y la complejidad del sistema sujeto a prueba, esto puede tener un impacto significativo en la planificación y financiación de las pruebas. Dado que el coste de estos entornos puede ser elevado, se pueden tener en cuenta las siguientes opciones:

- Uso del entorno de producción.
- Uso de una versión reducida[18] del sistema, teniendo cuidado de que los resultados de la prueba obtenidos sean suficientemente representativos del sistema de producción[19].
- Uso de recursos basados en la nube como alternativa a la adquisición directa de los recursos.
- Utilizando entornos virtualizados.

Los tiempos de ejecución de la prueba deben planificarse cuidadosamente, y es muy probable que algunas pruebas sólo puedan ejecutarse en momentos concretos (por ejemplo, en horas de poco uso).

4.2.3 Necesidad de Adquisición de Herramientas y de Formación

Las herramientas forman parte del entorno de prueba. Las herramientas comerciales o los simuladores son especialmente relevantes para la eficiencia de desempeño y ciertas pruebas de seguridad. Los Analistas de Pruebas Técnicas deben estimar los costes y los plazos de adquisición, aprendizaje e implementación de las herramientas. Cuando se vayan a utilizar herramientas especializadas, la planificación deberá tener en cuenta las curvas de aprendizaje de las nuevas herramientas y el coste de la contratación de especialistas externos en las mismas.

El desarrollo de un simulador complejo puede representar un proyecto de desarrollo en sí mismo y debe planificarse como tal. En particular se deben tener en cuenta la prueba y la documentación de la herramienta desarrollada en el calendario y el plan de recursos. Se debe planificar un presupuesto y un tiempo suficientes para actualizar y volver a probar el simulador a medida que cambia el producto simulado. La planificación de los simuladores que vayan a utilizarse en aplicaciones de seguridad física debe tener en cuenta las pruebas de aceptación y la posible certificación del simulador por un organismo independiente.

---

[17] “educción” es la traducción de “elicitation” / “educir” es la traducción de “to elicitate”.

[18] “reducida” es la traducción de “scaled-down”.

[19] “sistema de producción” es la traducción “production system”.

4.2.4 Consideraciones Relativas a la Organización

Las pruebas técnicas pueden implicar la medición del comportamiento de varios componentes de un sistema completo (por ejemplo, servidores, bases de datos, redes). Si estos componentes están distribuidos en varios sitios y organizaciones, el esfuerzo necesario para planificar y coordinar las pruebas puede ser significativo. Por ejemplo, es posible que ciertos componentes de software sólo estén disponibles para las pruebas del sistema a una hora determinada del día, o que las organizaciones sólo ofrezcan apoyo para las pruebas durante un número limitado de días. No confirmar que los componentes del sistema y el personal (es decir, la competencia técnica[20] "prestada") de otras organizaciones están disponibles "a demanda" para realizar la prueba puede dar lugar a una grave interrupción de las pruebas programadas.

4.2.5 Consideraciones sobre Seguridad y Protección de Datos

En la etapa de planificación de prueba deben tenerse en cuenta las medidas de seguridad específicas implementadas para un sistema, a fin de asegurar que todas las actividades de prueba sean posibles. Por ejemplo, el uso de la codificación de datos puede dificultar la creación de datos de prueba y la verificación de los resultados.

Las políticas y leyes de protección de datos pueden impedir la generación de cualquier dato de prueba necesario basado en datos de producción (por ejemplo, datos personales, datos de tarjetas de crédito). Hacer que los datos de prueba sean anónimos es una tarea no trivial que debe planificarse como parte de la implementación de prueba.

4.3 Prueba de Seguridad

4.3.1 Razones para Tener en Cuenta la Prueba de Seguridad

La prueba de seguridad evalúa la vulnerabilidad de un sistema a las amenazas intentando comprometer la política de seguridad del sistema. A continuación, se presenta una lista de las posibles amenazas que deberían explorarse durante la prueba de seguridad:

- Copia no autorizada de aplicaciones o datos.
- Control de acceso no autorizado (por ejemplo, capacidad de realizar tareas para las que el usuario no tiene derechos). Los derechos, el acceso y los privilegios de los usuarios son el objetivo de estas pruebas. Esta información debería estar disponible en las especificaciones del sistema.
- Software que muestra efectos secundarios no deseados cuando realiza una función prevista. Por ejemplo, un reproductor multimedia[21] que reproduzca correctamente el audio pero que lo haga escribiendo los archivos en un almacenamiento temporal no cifrado, presenta un efecto secundario que puede ser aprovechado por los piratas informáticos.
- Código insertado en una página web que puede ser practicado por siguientes usuarios (secuencia de comandos de sitios cruzados[22] o XSS). Este código puede ser malicioso.
- Desbordamiento de memoria intermedia[23] (“buffer overrun”) que puede ser causado por la introducción de cadenas en un campo de entrada de la interfaz de usuario que son más

[20] “competencia técnica” es la traducción de “expertise”.

[21] “reproductor multimedia” es la traducción de “media player”.

[22] “secuencias de comandos en sitios cruzados” es la traducción de “cross-site scripting (XSS)”.

[23] “desbordamiento de la memoria intermedia” es la traducción de “buffer overflow”.

largas de lo que el código puede tratar de forma correcta. Una vulnerabilidad de desbordamiento de memoria intermedia representa una oportunidad para ejecutar instrucciones de código malicioso.

- La denegación de servicio[24], que impide a los usuarios interactuar con una aplicación (por ejemplo, sobrecargando un servidor web con peticiones "molestas").
- La interceptación, imitación y/o alteración y posterior retransmisión de las comunicaciones (por ejemplo, las transacciones con tarjetas de crédito) por parte de un tercero, de forma que el usuario no sea consciente de la presencia de ese tercero (ataque de intermediario[25]).
- Ruptura de códigos de encriptación[26] utilizados para proteger los datos sensibles.
- Bombas lógicas (a veces llamadas Huevos de Pascua), que pueden ser insertadas maliciosamente en el código y que se activan sólo bajo ciertas condiciones (por ejemplo, en una fecha específica). Cuando las bombas lógicas se activan, pueden realizar actos maliciosos como el borrado de archivos o el formateo de discos.

4.3.2 Planificación de la Prueba de Seguridad

En general, los siguientes aspectos son de especial relevancia a la hora de planificar una prueba de seguridad:

- Dado que los problemas de seguridad pueden introducirse durante la arquitectura, el diseño y la implementación del sistema, la prueba de seguridad puede programarse para los niveles de prueba de componente, integración y sistema. Debido a la naturaleza cambiante de las amenazas a la seguridad, la prueba de seguridad también puede programarse de forma regular después de que el sistema haya entrado en producción. Esto es especialmente cierto en el caso de las arquitecturas abiertas y dinámicas, como el Internet de las cosas (IoT), en las que la fase de producción se caracteriza por numerosas actualizaciones de los elementos de software y hardware utilizados.
- Los enfoques de prueba propuestos por el Analista de Pruebas Técnicas pueden incluir revisiones de la arquitectura, el diseño y el código, y el análisis estático del código con herramientas de seguridad. Estos pueden ser efectivos para encontrar problemas de seguridad que se pasan por alto fácilmente durante las pruebas dinámicas.
- El Analista de Pruebas Técnicas puede ser llamado a diseñar y realizar ciertos "ataques" de seguridad (véase más adelante) que requieren una cuidadosa planificación y coordinación con los implicados (incluidos los especialistas en pruebas de seguridad).Otras pruebas de seguridad pueden realizarse en colaboración con los desarrolladores o con los analistas de pruebas (por ejemplo, para probar los derechos, el acceso y los privilegios de los usuarios).
- Un aspecto esencial de la planificación de la prueba de seguridad es la obtención de aprobaciones. Para el Analista de Pruebas Técnicas, esto significa asegurar que se ha obtenido el permiso explícito del Jefe de la Prueba para realizar las pruebas de seguridad planificadas. Cualquier prueba adicional no planificada que se lleve a cabo podría parecer un ataque real y la persona que lleve a cabo esas pruebas podría correr el riesgo de sufrir acciones legales. Sin nada por escrito que demuestre la intención y la autorización, la excusa

[24] “denegación del servicio” es la traducción de “service denial”.

[25] “ataque de intermediario” es la traducción de “Man in the Middle attack”). Antes “ataque de hombre interpuesto”.

[26] “ruptura de los códigos de encriptado” es la traducción de “breaking the encryption codes”.

"estábamos realizando una prueba de seguridad" puede ser difícil de explicar de forma convincente.

- Toda la planificación de la prueba de seguridad debe coordinarse con el responsable de la seguridad de la información de la organización, si ésta cuenta con dicho rol.
- Se debe tener en cuenta que las mejoras que se puedan realizar en la seguridad de un sistema pueden afectar a su eficiencia de desempeño o a su fiabilidad. Después de realizar las mejoras de seguridad es aconsejable tener en cuenta la necesidad de llevar a cabo pruebas de eficiencia de desempeño o de fiabilidad (véanse los apartados 4.5 y 4.4).

Al realizar la planificación de la prueba de seguridad pueden aplicarse estándares individuales, como la [IEC 62443-3-2], que se aplica a los sistemas de automatización y control industrial.

El programa de estudio de prueba de seguridad [CT_SEC_SYL] incluye más detalles sobre los elementos clave del plan de prueba de seguridad.

4.3.3 Especificación de la Prueba de Seguridad

Las pruebas de seguridad particulares pueden agruparse [Whittaker04] según el origen del riesgo de seguridad. Entre ellas se encuentran las siguientes:

- Relacionadas con la interfaz de usuario - acceso no autorizado y entradas maliciosas.
- Relacionadas con el sistema de archivos - acceso a datos sensibles almacenados en archivos o repositorios.
- Relacionados con el sistema operativo - almacenamiento de información sensible, como contraseñas, en forma no encriptada en la memoria, que podría quedar expuesta cuando el sistema se colapse a través de entradas maliciosas.
- Relacionados con el software externo - interacciones que pueden producirse entre los componentes externos que utiliza el sistema. Pueden darse a nivel de red (por ejemplo, paquetes o mensajes incorrectos pasados) o a nivel de componente de software (por ejemplo, fallo de un componente de software en el que se apoya el software).

Las subcaracterísticas de seguridad de la norma ISO 25010 [ISO 25010] también proporcionan una base a partir de la cual se pueden especificar las pruebas de seguridad. Éstas se concentran en los siguientes aspectos de la seguridad:

- Confidencialidad.
- Integridad.
- No repudio.
- Responsabilidad.
- Autenticidad.

Se puede utilizar el siguiente enfoque [Whittaker04] para desarrollar pruebas de seguridad:

- Recopilar información que pueda ser útil para especificar las pruebas, como los nombres de los empleados, las direcciones físicas, los detalles relativos a las redes internas, los números IP, la identidad del software o el hardware utilizado y la versión del sistema operativo.
- Llevar a cabo un escaneo de vulnerabilidad utilizando herramientas ampliamente disponibles. Dichas herramientas no se utilizan directamente para comprometer el sistema o sistemas, sino para identificar las vulnerabilidades existentes o que puedan dar lugar a una violación de la política de seguridad. También se pueden identificar las vulnerabilidades específicas

utilizando información y listas de comprobación como las proporcionadas por el “National Institute of Standards and Technology (NIST)” [Web-1] y el “Open Web Application Security Project™ (OWASP)” [Web-4].

- Desarrollar "planes de ataque" (es decir, un plan de acciones de prueba destinadas a comprometer la política de seguridad de un sistema concreto) utilizando la información recopilada. En los planes de ataque es necesario especificar varias entradas | aportación (interfaz de usuario, sistema de archivos) para detectar los defectos de seguridad más graves. Los diversos "ataques" descritos en [Whittaker04] son una valiosa fuente de técnicas desarrolladas específicamente para la prueba de seguridad.

Se debe tener en cuenta que se pueden desarrollar planes de ataque para la prueba de penetración.

La sección 3.2 (análisis estático) y el programa de estudio de prueba de seguridad [CT_SEC_SYL] incluyen más detalles sobre las pruebas de seguridad.

4.4 Prueba de Fiabilidad

4.4.1 Introducción

La clasificación ISO 25010 de las características de calidad de producto define las siguientes subcaracterísticas para la fiabilidad: madurez, disponibilidad, tolerancia a defectos y capacidad de recuperación. La prueba de fiabilidad se refiere a la capacidad de un sistema o de un software para realizar funciones específicas en condiciones específicas durante un período de tiempo determinado.

4.4.2 Prueba de Madurez

La madurez es el grado en que el sistema (o el software) cumple los requisitos de fiabilidad en condiciones normales de funcionamiento, que suelen especificarse utilizando un perfil operativo (véase el apartado 4.9). Las medidas de madurez, cuando se utilizan, suelen constituir uno de los criterios de entrega de un sistema.

Tradicionalmente, la madurez se ha especificado y medido para los sistemas de alta fiabilidad, como los asociados a funciones críticas para la seguridad (por ejemplo, un sistema de control de vuelo de una aeronave), en los que el objetivo de madurez se define como parte de una norma reglamentaria. Un requisito de madurez para un sistema de alta fiabilidad de este tipo puede ser un Tiempo Medio Entre Fallos (TMEF) de hasta 109 horas (aunque esto es prácticamente imposible de medir).

El enfoque habitual para probar la madurez de los sistemas de alta fiabilidad se conoce como modelo de crecimiento de la fiabilidad, y normalmente tiene lugar al final de la

prueba de sistema, una vez que se han completado las pruebas de otras características de calidad y se han corregido los defectos asociados a los fallos detectados. Se trata de un enfoque estadístico que suele realizarse en un entorno de pruebas lo más parecido posible al entorno de operaciones. Para medir un Tiempo Medio Entre Fallos (TMEF) especificado, se generan entradas de prueba basadas en el perfil operativo y se ejecuta el sistema y se registran los fallos (que posteriormente se corrigen). La reducción de la frecuencia de los fallos permite predecir el Tiempo Medio Entre Fallos mediante un modelo de crecimiento de la fiabilidad.

Cuando se utiliza la madurez como objetivo para los sistemas de menor fiabilidad (por ejemplo, no relacionados con la seguridad), entonces se puede utilizar el número de fallos observados durante un período definido de uso operativo previsto (por ejemplo, no más de 2 fallos de alto impacto por semana) y puede registrarse como parte del Acuerdo de Nivel de Servicio (ANS) para el sistema.

4.4.3 Prueba de Disponibilidad

La disponibilidad suele especificarse en términos de la cantidad de tiempo que un sistema (o software) está disponible para los usuarios y otros sistemas en condiciones normales de funcionamiento. Los sistemas pueden tener una madurez baja pero aún así tener una alta disponibilidad. Por ejemplo, una red telefónica puede fallar en la conexión de varias llamadas (y por tanto tener una madurez baja), pero mientras el sistema se recupere rápidamente y permita los siguientes intentos de conexión la mayoría de los usuarios estarán satisfechos. Sin embargo, un único fallo que provocara un corte de la red telefónica durante varias horas representaría un nivel de disponibilidad inaceptable. La disponibilidad suele especificarse como parte de un acuerdo de nivel de servicio y se mide en el caso de los sistemas operativos, como los sitios web y las aplicaciones de software como servicio (SaaS por sus siglas en inglés). La disponibilidad de un sistema puede describirse como del 99,999% ("cinco nueves"), en cuyo caso no debería estar indisponible más de 5 minutos al año; alternativamente, la disponibilidad del sistema puede especificarse en términos de indisponibilidad (por ejemplo, el sistema no deberá estar fuera de servicio durante más de 60 minutos al mes).

La medición de la disponibilidad antes de la operación (por ejemplo, como parte de la toma de la decisión de entrega) se realiza a menudo utilizando las mismas pruebas que se utilizan para medir la madurez; las pruebas se basan en un perfil operativo de uso previsto durante un período prolongado y se realizan en un entorno de prueba lo más parecido posible al entorno de operaciones. La disponibilidad puede medirse como MTTF/(MTTF + Tiempo Medio de Reparación (TMDR)), donde MTTF es el Tiempo Medio de Fallo y Tiempo Medio de Reparación (TMDR) es el Tiempo Medio de Reparación (Tiempo Medio de Reparación (TMDR)), que suele medirse como parte de las pruebas de mantenibilidad. Cuando un sistema es de alta fiabilidad e incorpora capacidad de recuperación (véase el apartado 4.4.5), podemos sustituir el Tiempo Medio de Reparación (Tiempo Medio de Reparación (TMDR)) en la ecuación cuando el sistema tarda en recuperarse de un fallo.

4.4.4 Prueba de Tolerancia a Defectos

Los sistemas (o el software) con requisitos de fiabilidad extremadamente altos suelen incorporar un diseño de tolerancia a defectos, que idealmente permite que el sistema siga funcionando sin que se note el tiempo de inactividad cuando se producen fallos. La principal medida de la tolerancia a fallos de un sistema es la capacidad de éste para tolerar fallos. Por lo tanto, las pruebas de tolerancia a defectos implican la simulación de fallos para determinar si el sistema puede seguir funcionando cuando se produce un fallo de este tipo. La identificación de las posibles condiciones de fallo que deben probarse es una parte importante de las pruebas de tolerancia a defectos.

Un diseño tolerante a defectos suele incluir uno o más subsistemas duplicados, lo que proporciona un nivel de redundancia en caso de fallo. En el caso del software, estos sistemas duplicados necesitan desarrollarse de forma independiente, para evitar fallos de modo común; este enfoque se conoce como programación de N-versiones. Los sistemas de control de vuelo de los aviones pueden incluir tres o cuatro niveles de redundancia, con las funciones más críticas implementadas en diversas variantes. Cuando la fiabilidad del hardware es un asunto de interés, un sistema embebido puede funcionar con varios procesadores distintos, mientras que un sitio web crítico puede funcionar con un servidor espejo (tolerancia a fallos) que realice las mismas funciones y que esté siempre disponible para tomar el relevo en caso de que falle el servidor principal. Sea cual sea el enfoque de tolerancia a defectos que se implemente, la prueba suele requerir la detección del fallo y la posterior respuesta al mismo para ser probada.

4.4.5 Prueba de la Capacidad de Recuperación

La capacidad de recuperación es una medida de la capacidad de un sistema (o de un software) para recuperarse de un fallo, ya sea en términos del tiempo necesario para recuperarse (que puede ser hasta un estado de funcionamiento disminuido) o de la cantidad de datos perdidos. Los enfoques de prueba de capacidad de recuperación incluyen la prueba de tolerancia a fallos y la prueba de copia de

seguridad y restauración; ambas suelen incluir procedimientos de prueba basados en simulacros y sólo ocasionalmente, e idealmente, sin previo aviso, pruebas prácticas en entornos operativos.

La prueba de copia de seguridad y restauración se concentra en probar los procedimientos establecidos para minimizar los efectos de un fallo en los datos del sistema. Las pruebas evalúan los procedimientos tanto para la copia de seguridad como para la restauración de los datos. Mientras que las pruebas para la copia de seguridad de los datos son relativamente fáciles, las pruebas para la restauración de un sistema a partir de los datos que cuentan con copia de seguridad pueden ser más complejas y a menudo requieren una planificación cuidadosa para asegurar que se minimiza la interrupción del sistema operativo. Las medidas incluyen el tiempo que se tarda en realizar los diferentes tipos de copia de seguridad (por ejemplo, completa e incremental), el tiempo que se tarda en restaurar los datos (el objetivo de tiempo de recuperación) y el nivel de pérdida de datos que es aceptable (objetivo de punto de recuperación[27]).

La prueba de tolerancia a fallos se lleva a cabo cuando la arquitectura del sistema comprende tanto un sistema primario como un sistema de tolerancia a fallos que tomará el relevo si el sistema primario falla. Cuando un sistema debe ser capaz de recuperarse de un fallo catastrófico (por ejemplo, una inundación, un ataque terrorista o un grave ataque de software de secuestro de datos[28]), la prueba de tolerancia a fallos suele denominarse prueba de recuperación de desastres y el sistema (o sistemas) de recuperación de fallos suele estar en otra ubicación geográfica. La realización de una prueba completa de recuperación de desastres en un sistema operativo necesita una planificación extremadamente cuidadosa debido a los riesgos y a la interrupción (a menudo del tiempo libre de los altos cargos, que probablemente se ocupará en la gestión de la recuperación). Si una prueba completa de recuperación de desastres falla, volveríamos inmediatamente al sistema primario (¡ya que no ha sido realmente destruido!). Las pruebas de tolerancia a fallos incluyen la comprobación de que el paso al sistema de recuperación, una vez asumido, proporciona el nivel de servicio requerido.

4.4.6 Planificación de la Prueba de Fiabilidad

En general, los siguientes puntos son especialmente importantes a la hora de planificar la prueba de fiabilidad:

- Cronología[29] (o sincronización) - La prueba de fiabilidad suele requerir que se pruebe el sistema completo y que se completen otros tipos de pruebas, lo que puede llevar mucho tiempo.
- Costes - Los sistemas de alta fiabilidad son notoriamente caros de probar debido a los largos periodos durante los cuales los sistemas deben ser probados sin fallos para poder predecir un Tiempo Medio Entre Fallos (TMEF) elevado requerido.
- Duración - La prueba de madurez mediante modelos de crecimiento de la fiabilidad se basa en los fallos detectados y, en el caso de niveles de fiabilidad elevados, se necesitará mucho tiempo para obtener resultados estadísticamente significativos.
- Entorno de prueba - El entorno de prueba debe ser lo más parecido posible al operativo, o bien puede utilizarse el entorno operativo. Sin embargo, si se utiliza el entorno operativo, esto puede resultar molesto para los usuarios y puede suponer un alto riesgo si, por ejemplo, una prueba de recuperación de desastres afecta negativamente al sistema operativo.
- Alcance - Pueden probarse diferentes subsistemas y componentes para diferentes tipos y niveles de fiabilidad.

[27] "objetivo del punto de recuperación" es la traducción del término "recovery point objective".

[28] "ataque de software de secuestro de datos" es la traducción del término "ransomware attack".

[29] "cronología" es la traducción del término "timing".

- Criterios de salida - Los requisitos de fiabilidad deben establecerse mediante normas reglamentarias para las aplicaciones relacionadas con la seguridad física.
- Fallo - Las medidas de fiabilidad dependen en gran medida del recuento de fallos, por lo que debe haber un acuerdo previo sobre lo que constituye un fallo.
- Desarrolladores - Para la prueba de madurez mediante modelos de crecimiento de la fiabilidad, debe llegarse a un acuerdo con los desarrolladores para que los defectos identificados se solucionen lo antes posible.
- La medición de la fiabilidad operativa es relativamente sencilla en comparación con la medición de la fiabilidad antes de la entrega, ya que sólo hay que medir los fallos; para ello puede ser necesaria la colaboración del personal de operaciones.
- Prueba temprana - Lograr una alta fiabilidad (en contraposición a la medición de la fiabilidad) requiere que las pruebas comiencen lo antes posible, con revisiones rigurosas de los primeros documentos de referencia y el análisis estático del código.

4.4.7 Especificación de la Prueba de Fiabilidad

Para probar la madurez y la disponibilidad, la prueba se basa en gran medida en probar el sistema en condiciones normales de funcionamiento. Para estas pruebas, se requiere un perfil operativo que defina cómo se espera que se utilice el sistema. Para más detalles sobre los perfiles operativos, véase el apartado 4.9.

Para probar la tolerancia a defectos y la capacidad de recuperación, suele ser necesario generar pruebas que reproduzcan fallos en el entorno y en el propio sistema, para determinar cómo reacciona éste. Para ello se suelen utilizar pruebas de inyección de defectos. Existen varias técnicas y listas de comprobación para identificar los posibles defectos y los fallos correspondientes (por ejemplo, el Análisis del Árbol de Defectos (AAD) y el Análisis del Modo de Fallo y sus Efectos (AMFE)).

4.5 Prueba de Rendimiento

4.5.1 Introducción

La clasificación ISO 25010 de las características de calidad de producto define las siguientes subcaracterísticas de eficiencia de desempeño: comportamiento temporal, uso de recursos y capacidad. La prueba de rendimiento (asociada a la característica de calidad de eficiencia de desempeño) se ocupa de medir el rendimiento de un sistema o software en condiciones especificadas en relación con la cantidad de recursos utilizados. Los recursos típicos son el tiempo transcurrido, el tiempo de la CPU, la memoria y el ancho de banda.

4.5.2 Prueba del Comportamiento Temporal

La prueba del comportamiento temporal mide los siguientes aspectos de un sistema (o software) en condiciones de funcionamiento especificadas:

- tiempo transcurrido desde la recepción de una solicitud hasta la primera respuesta (es decir, el tiempo para empezar a responder, no el tiempo para completar la actividad solicitada), también llamado tiempo de respuesta.
- tiempo de respuesta desde que se inicia una actividad hasta que se completa, también llamado tiempo de procesamiento.
- número de actividades completadas por unidad de tiempo (por ejemplo, número de operaciones de base de datos por segundo), también llamado tasa de rendimiento.

En muchos sistemas, los tiempos de respuesta máximos para las distintas funciones del sistema se especifican como requisitos. En estos casos, el tiempo de respuesta es el tiempo transcurrido más el tiempo de respuesta. Cuando un sistema tiene que realizar una serie de pasos (por ejemplo, un pipeline) para completar una actividad, puede ser útil medir el tiempo que tarda cada paso y analizar los resultados para determinar si uno o más pasos están causando un cuello de botella.

4.5.3 Prueba de Utilización de Recursos

La prueba de utilización de recursos mide los siguientes aspectos de un sistema (o software) en condiciones de funcionamiento especificadas:

- utilización de la CPU, normalmente como porcentaje del tiempo de CPU disponible.
- utilización de la memoria, normalmente como porcentaje de la memoria disponible.
- utilización de los dispositivos de entrada/salida, normalmente como porcentaje del tiempo disponible de los dispositivos de entrada/salida.
- utilización del ancho de banda, normalmente como porcentaje del ancho de banda disponible.

4.5.4 Prueba de Capacidad

La prueba de capacidad mide los límites máximos de los siguientes aspectos de un sistema (o software) en condiciones de funcionamiento especificadas:

- transacciones procesadas por unidad de tiempo (por ejemplo, un máximo de 687 palabras traducidas por minuto).
- usuarios que acceden simultáneamente al sistema (por ejemplo, un máximo de 1223 usuarios).
- nuevos usuarios añadidos para tener acceso al sistema por unidad de tiempo (por ejemplo, máximo de 400 usuarios añadidos por segundo).

4.5.5 Aspectos Comunes de la Prueba de Rendimiento

Cuando se prueba el comportamiento temporal, la utilización de recursos o la capacidad es normal que se tomen varias mediciones y se utilice la media como medida informada; esto se debe a que los valores temporales medidos pueden fluctuar en función de otras tareas de fondo que el sistema pueda estar realizando. En algunas situaciones, las mediciones se tratarán de forma más meticulosa (por ejemplo, utilizando la varianza u otras medidas estadísticas), o se investigarán los valores atípicos y se descartarán, si procede.

El análisis dinámico (véase el apartado 3.3.4) puede utilizarse para identificar los componentes que causan un cuello de botella, medir los recursos utilizados para la prueba de utilización de recursos y medir los límites máximos para la prueba de capacidad.

4.5.6 Tipos de Pruebas de Rendimiento

La prueba de rendimiento se diferencia de la mayoría de las otras formas de prueba en que puede haber dos objetivos distintos. El primero es determinar si el software bajo prueba cumple los criterios de aceptación especificados. Por ejemplo, determinar si el sistema muestra una página web solicitada en el plazo máximo especificado de 4 segundos. El segundo objetivo es proporcionar información a los desarrolladores del sistema para ayudarles a mejorar la eficiencia del mismo. Por ejemplo, detectar los cuellos de botella e identificar qué partes de la arquitectura del sistema se ven afectadas negativamente cuando un número inesperadamente elevado de usuarios accede al sistema simultáneamente.

Las pruebas de rendimiento descritas en los apartados 4.5.2, 4.5.3 y 4.5.4 pueden utilizarse para determinar si el software sometido a prueba cumple los criterios de aceptación especificados. También se utilizan para medir los valores de referencia que se utilizan para la comparación posterior cuando se modifica el sistema. Los siguientes tipos de pruebas de rendimiento se utilizan más a menudo para proporcionar información a los desarrolladores sobre cómo responde el sistema en diferentes condiciones de funcionamiento.

4.5.6.1 Prueba de Carga

La prueba de carga se concentra en la capacidad de un sistema para tratar diferentes cargas. Estas cargas suelen definirse en función del número de usuarios que acceden al sistema de forma simultánea o del número de procesos que se ejecutan de forma concurrente y pueden definirse como perfiles operativos (véase el apartado 4.9 para más detalles sobre los perfiles operativos). El tratamiento de estas cargas suele medirse en términos del comportamiento temporal del sistema y de la utilización de recursos (por ejemplo, determinando el efecto en el tiempo de respuesta de duplicar el número de usuarios). Cuando se realizan pruebas de carga, es una práctica normal comenzar con una carga baja y aumentar gradualmente la carga mientras se mide el comportamiento temporal del sistema y la utilización de recursos. La información típica de las pruebas de carga que podría ser útil para los desarrolladores incluiría cambios inesperados en los tiempos de respuesta o en el uso de los recursos del sistema cuando éste maneja una carga determinada.

4.5.6.2 Prueba de Estrés

Hay dos tipos de pruebas de estrés; la primera es similar a la prueba de carga y la segunda es una forma de prueba de robustez.

En el primero, las pruebas de carga se realizan normalmente con la carga fijada inicialmente al máximo esperado y luego se incrementa hasta que el sistema falla (por ejemplo, los tiempos de respuesta se vuelven excesivamente largos o el sistema se bloquea). A veces, en lugar de forzar el sistema a fallar, se utiliza una carga elevada para estresar el sistema y luego se reduce la carga a un nivel normal y se comprueba el sistema para asegurar que sus niveles de rendimiento se han recuperado a sus niveles anteriores al estrés.

En la segunda, las pruebas de rendimiento se ejecutan con el sistema deliberadamente comprometido reduciendo su acceso a los recursos previstos (por ejemplo, reduciendo la memoria o el ancho de banda disponibles). Los resultados de las pruebas de estrés pueden proporcionar a los desarrolladores una visión de los aspectos del sistema que son más críticos (es decir, los eslabones débiles) y que, por tanto, pueden necesitar una actualización.

4.5.6.3 Prueba de Escalabilidad

Un sistema escalable puede adaptarse a diferentes cargas. Por ejemplo, un sitio web escalable podría escalar para utilizar más servidores en el extremo posterior[30] cuando la demanda aumenta y utilizar menos cuando la demanda disminuye. La prueba de escalabilidad es similar a la prueba de carga, pero prueba la capacidad de un sistema para un escalado ascendente y descendente cuando se enfrenta a cargas cambiantes (por ejemplo, más usuarios de los que puede soportar el hardware actual).

El programa de estudio de Prueba de Rendimiento [CT_PT_SYL] incluye otros tipos de pruebas de rendimiento.

4.5.7 Planificación de la Prueba de Rendimiento

En general, los siguientes puntos son especialmente importantes a la hora de planificar la prueba de eficiencia de desempeño:

[30] “extremo posterior” es la traducción del término “back-end”

- Cronología - La prueba de rendimiento suele requerir que todo el sistema esté implementado y se ejecute en un entorno de prueba representativo, lo que significa que suele realizarse como parte de la prueba de sistema.
- Revisiones - Las revisiones del código, en particular las que se concentran en la interacción con la base de datos, la interacción con los componentes y el tratamiento de errores, pueden identificar problemas de eficiencia de desempeño (sobre todo en lo que se refiere a la lógica de "esperar y volver a intentar"[31] y a las consultas ineficientes) y deben programarse para que tengan lugar tan pronto como el código esté disponible (es decir, antes de la prueba dinámica).
- Prueba temprana - Algunas pruebas de rendimiento (por ejemplo, determinar la utilización de la CPU para un componente crítico) pueden programarse como parte de la prueba de componente. Los componentes identificados como un cuello de botella por la prueba de rendimiento pueden ser actualizados y vueltos a probar de forma aislada como parte de la prueba de componentes.
- Cambios en la arquitectura - En ocasiones, los resultados adversos de la prueba de rendimiento pueden dar lugar a un cambio en la arquitectura del sistema. Cuando los resultados de la prueba de rendimiento puedan sugerir cambios importantes en el sistema, las pruebas de rendimiento deben comenzar lo antes posible, para maximizar el tiempo disponible para abordar estos problemas.
- Costes - Las herramientas y los entornos de prueba pueden ser costosos, lo que significa que pueden alquilarse entornos de prueba temporales basados en la nube y utilizarse licencias de herramientas de "prepago"[32]. En estos casos, la planificación de la prueba suele necesitar optimizar el tiempo de ejecución de las pruebas para minimizar los costes.
- Entorno de prueba - El entorno de prueba debe ser lo más representativo posible del entorno de operaciones, ya que de lo contrario aumenta el reto de trasladar los resultados de prueba de rendimiento del entorno de prueba al entorno de operaciones previsto.
- Criterios de salida - En ocasiones, los requisitos de eficiencia de desempeño pueden ser difíciles de obtener del cliente, por lo que a menudo se obtienen a partir de líneas base de sistemas anteriores o similares. En el caso de los sistemas embebidos relacionados con la seguridad física, algunos requisitos, como la cantidad máxima de CPU y memoria utilizada, pueden estar especificados por normas reglamentarias.
- Herramientas - Las herramientas para generación de carga suelen ser necesarias para apoyar la prueba de rendimiento. Por ejemplo, verificar la escalabilidad de un sitio web que goza de gran popularidad puede requerir la simulación de cientos de miles de usuarios virtuales. Las herramientas que simulan restricciones de recursos también son especialmente útiles para las pruebas de estrés. Hay que asegurarse de que cualquier herramienta adquirida para apoyar la prueba sea compatible con los protocolos de comunicación utilizados por el sistema sujeto a prueba.

El programa de estudio de Prueba de Rendimiento [CT_PT_SYL] incluye más detalles sobre la planificación de prueba de rendimiento.

4.5.8 Especificación de la Prueba de Rendimiento

La prueba de rendimiento se basa, en gran medida, en probar el sistema en condiciones de funcionamiento especificadas. Para estas pruebas, se requiere un perfil operativo que defina cómo se

[31] "esperar y volver a intentar" es la traducción del término “wait and retry”

[32] “licencia de herramienta de "prepago"” es la traducción del término “top-up” tool license”

espera que se utilice el sistema. Consulte la sección 4.9 para obtener más detalles sobre los perfiles operativos.

Para la prueba de rendimiento, a menudo es necesario cambiar la carga del sistema modificando partes del perfil operativo para simular un cambio del uso operativo esperado del sistema. Por ejemplo, en el caso de las pruebas de capacidad, normalmente será necesario anular el perfil operativo en el área de la variable que se está probando (por ejemplo, aumentar el número de usuarios que acceden al sistema hasta que éste deje de responder para determinar la capacidad de acceso de los usuarios). Del mismo modo, se puede aumentar gradualmente el volumen de las transacciones cuando se realizan pruebas de carga.

El programa de estudio de Prueba de Rendimiento [CTSL_PT_SYL] incluye más detalles sobre el diseño de prueba de eficiencia de desempeño.

4.6 Prueba de Mantenibilidad

A menudo, el software pasa mucho más tiempo de su vida útil siendo mantenido que siendo desarrollado. Para asegurar que la tarea de realizar el mantenimiento sea lo más eficiente posible, se realiza la prueba de mantenibilidad para medir la facilidad con la que se puede analizar, cambiar, probar, modularizar y reutilizar el código. La prueba de mantenibilidad no debe confundirse con la prueba de mantenimiento, que se realiza para probar los cambios realizados en el software operativo.

Los objetivos típicos de mantenibilidad de los implicados afectados (por ejemplo, el propietario u operador del software) incluyen:

- Minimizar el coste de propiedad o de operación del software.
- Minimizar el tiempo de inactividad necesario para el mantenimiento del software.

Las pruebas de mantenibilidad deben incluirse en un enfoque de prueba cuando concurran uno o varios de los siguientes factores:

- Es probable que se produzcan cambios en el software después de su entrada en producción (por ejemplo, para corregir defectos o introducir actualizaciones planificadas).
- Los implicados consideran que los beneficios de alcanzar los objetivos de mantenibilidad a lo largo del ciclo de vida de desarrollo del software superan los costes de realizar las pruebas de mantenibilidad y de realizar los cambios necesarios.
- Los riesgos de una mala mantenibilidad del software (por ejemplo, largos tiempos de respuesta a los defectos notificados por los usuarios y/o clientes) justifican la realización de pruebas de mantenibilidad.

4.6.1 Prueba de Mantenibilidad Estática y Dinámica

Las técnicas apropiadas para la prueba de mantenibilidad estática incluyen el análisis estático y las revisiones, tal y como se ha comentado en las secciones 3.2 y 5.2. La prueba de mantenibilidad debe iniciarse tan pronto como la documentación del diseño esté disponible y debe continuar durante todo el esfuerzo de implementación del código. Dado que la mantenibilidad está integrada en el código y en la documentación de cada componente del código, la mantenibilidad puede evaluarse en las primeras fases del ciclo de vida de desarrollo del software sin tener que esperar a que el sistema esté terminado y en funcionamiento.

La prueba de mantenibilidad dinámica se concentra en los procedimientos documentados desarrollados para el mantenimiento de una aplicación concreta (por ejemplo, para realizar actualizaciones de software). Los escenarios de mantenimiento se utilizan como casos de prueba para asegurar que los niveles de servicio requeridos son alcanzables con los procedimientos documentados. Esta forma de prueba es especialmente relevante cuando la infraestructura

subyacente es compleja y los procedimientos de mantenimiento pueden implicar a múltiples departamentos/organizaciones. Esta forma de prueba puede tener lugar como parte de la prueba de aceptación operativa.

4.6.2 Subcaracterísticas de la Mantenibilidad

La mantenibilidad de un sistema puede medirse en términos de:

- Capacidad de ser analizado.
- Capacidad de ser modificado.
- Capacidad de ser probado.

Entre los factores que influyen en estas características se encuentran la aplicación de buenas prácticas de programación (por ejemplo, comentarios, nomenclatura de variables, sangría) y la disponibilidad de documentación técnica (por ejemplo, especificaciones de diseño del sistema, especificaciones de la interfaz).

Otras subcaracterísticas de calidad relevantes para la mantenibilidad [ISO 25010] son:

- Modularidad.
- Capacidad de reutilización.

La modularidad puede probarse mediante un análisis estático (véase el apartado 3.2.3). Las pruebas de reutilización pueden adoptar la forma de revisiones de arquitectura (véase el capítulo 5).

4.7 Prueba de Portabilidad

4.7.1 Introducción

En general, la prueba de portabilidad se refiere al grado en que un componente o sistema de software puede ser transferido a su entorno previsto (ya sea inicialmente o desde un entorno existente), puede ser adaptado a un nuevo entorno o puede sustituir a otra entidad.

La norma ISO 25010 [ISO 25010] incluye las siguientes subcaracterísticas de portabilidad:

- Adaptabilidad.
- Instalanbilidad.
- Capacidad de ser reemplazado.

La prueba de portabilidad puede comenzar con componentes individuales (por ejemplo, la posibilidad de sustituir un componente concreto, como el cambio de un sistema de gestión de bases de datos a otro) y luego ampliar su alcance a medida que se disponga de más código. La instalabilidad puede no ser comprobable hasta que todos los componentes del producto se encuentren funcionalmente operativos.

La portabilidad debe diseñarse e integrarse en el producto, por lo que debe tenerse en cuenta en las primeras fases de diseño y arquitectura. Las revisiones de la arquitectura y el diseño pueden ser especialmente productivas para identificar posibles requisitos y problemas de portabilidad (por ejemplo, la dependencia de un sistema operativo concreto).

4.7.2 Prueba de Instalabilidad

La prueba de instalabilidad se realiza sobre el software y sobre los procedimientos escritos utilizados para instalar el software en su entorno de destino. Esto puede incluir, por ejemplo, el software

desarrollado para instalar un sistema operativo, o un "asistente" de instalación utilizado para instalar un producto en un ordenador cliente.

Los objetivos típicos de la prueba de instalabilidad son los siguientes:

- Validar que el software puede instalarse siguiendo las instrucciones de un manual de instalación (incluida la ejecución de cualquier guion de instalación), o utilizando un asistente de instalación. Esto incluye practicar las opciones de instalación para diferentes configuraciones de hardware/software y para varios grados de instalación (por ejemplo, inicial o de actualización).
- Probar si los fallos que se producen durante la instalación (por ejemplo, la falta de carga de determinadas DLL) son resueltos por el software de instalación correctamente sin dejar el sistema en un estado indefinido (por ejemplo, software parcialmente instalado o configuraciones incorrectas del sistema).
- Probar si se puede completar una instalación/desinstalación parcial.
- Probar si un asistente de instalación puede identificar plataformas hardware o configuraciones del sistema operativo no válidas.
- Medir si el proceso de instalación puede completarse en un número determinado de minutos o en un número determinado de pasos.
- Validar que el software puede ser objeto de una reducción de prestaciones[33] o desinstalación.
- La prueba de adecuación funcional se realiza normalmente después de la prueba de instalación para detectar cualquier defecto que pueda haber introducido la instalación (por ejemplo, configuraciones incorrectas, funciones no disponibles). La prueba de usabilidad suele realizarse en paralelo a la prueba de instalabilidad (por ejemplo, para probar que los usuarios reciben instrucciones comprensibles y mensajes de retroalimentación/error durante la instalación).

4.7.3 Prueba de Adaptabilidad

La prueba de adaptabilidad comprueba si una aplicación determinada puede funcionar correctamente en todos los entornos destino previstos (hardware, software, middleware, sistema operativo, etc.). La especificación de las pruebas de adaptabilidad requiere que los entornos destino previstos estén identificados, configurados y disponibles para el equipo de prueba. A continuación, estos entornos se prueban mediante una selección de casos de prueba funcionales que practican los distintos componentes presentes en el entorno.

La adaptabilidad puede referirse a la capacidad del software para ser transferido a varios entornos específicos mediante la ejecución de un procedimiento predefinido. Las pruebas pueden evaluar este procedimiento.

4.7.4 Prueba de Capacidad de Ser Reemplazado

La prueba de la capacidad de ser reemplazado se concentra en la capacidad de un componente de software para sustituir a un componente de software existente en un sistema. Esto puede ser especialmente relevante para los sistemas que utilizan software comercial de distribución masiva (COTS) para componentes específicos del sistema o para aplicaciones IoT.

Las pruebas de capacidad de ser reemplazado pueden ejecutarse en paralelo con las pruebas de integración funcional cuando haya más de un componente alternativo disponible para su integración

[33] "reducción de prestaciones" es una traducción del término "downgrade" en este contexto.

en el sistema completo. La capacidad de ser reemplazado también puede evaluarse mediante una revisión técnica o una inspección en los niveles de arquitectura y diseño, en la que se pone énfasis en la definición clara de las interfaces del componente que puede utilizarse como reemplazo.

4.8 Prueba de Compatibilidad

4.8.1 Introducción

La prueba de compatibilidad tiene en cuenta los siguientes aspectos [ISO 25010]:

- Coexistencia.
- Interoperabilidad.

4.8.2 Prueba de Coexistencia

Se dice que los sistemas informáticos que no están relacionados entre sí coexisten cuando pueden funcionar en el mismo entorno (por ejemplo, en el mismo hardware) sin afectar al comportamiento del otro (por ejemplo, conflictos de recursos). La prueba de coexistencia debe llevarse a cabo cuando el software nuevo o actualizado se va a implantar en entornos que ya contienen aplicaciones instaladas.

Los problemas de coexistencia pueden surgir cuando la aplicación se prueba en un entorno en el que es la única aplicación instalada (en el que no se detectan problemas de incompatibilidad) y luego se despliega en otro entorno (por ejemplo, de producción) que también ejecuta otras aplicaciones.

Los objetivos típicos de la prueba de coexistencia incluyen:

- Evaluar un posible impacto adverso en la adecuación funcional cuando las aplicaciones se cargan en el mismo entorno (por ejemplo, el uso conflictivo de recursos cuando un servidor ejecuta más de una aplicación)
- Evaluar el impacto en cualquier aplicación resultante del despliegue de correcciones y actualizaciones del sistema operativo

Los problemas de coexistencia deben analizarse cuando se planifica el entorno de producción objetivo, pero las pruebas reales se realizan normalmente después de que se haya completado la prueba de sistema.

4.8.3 Perfiles Operativos

Los perfiles operativos se utilizan como parte de la especificación de una prueba para varios tipos de pruebas no funcionales, incluidas la prueba de fiabilidad y la prueba de rendimiento. Son especialmente útiles cuando el requisito que se prueba incluye la restricción de "en condiciones especificadas", ya que pueden utilizarse para definir estas condiciones.

El perfil operativo define un patrón de uso del sistema, normalmente en términos de los usuarios del sistema y de las operaciones realizadas por el mismo. Los usuarios suelen especificarse en términos de cuántos se espera que utilicen el sistema (y en qué momentos), y quizás su tipo (por ejemplo, usuario principal, usuario secundario). Se suelen especificar las diferentes operaciones que se espera que realice el sistema, con su frecuencia (y probabilidad de ocurrencia). Esta información puede obtenerse mediante el uso de herramientas de monitorización (cuando la aplicación real o una similar ya está disponible) o mediante la predicción del uso basada en algoritmos o estimaciones proporcionadas por la organización del negocio.

Pueden utilizarse herramientas para generar entradas de prueba basadas en el perfil operativo, a menudo utilizando un enfoque que genera las entradas de prueba de forma pseudoaleatoria. Dichas herramientas pueden utilizarse para crear usuarios "virtuales" o simulados en cantidades que

coincidan con el perfil operativo (por ejemplo, para pruebas de fiabilidad y disponibilidad) o que lo superen (por ejemplo, para pruebas de estrés o capacidad). Consulte la sección 6.2.3 para obtener más detalles sobre estas herramientas.

5. Revisiones - 165 minutos

Palabras Clave

revisión	**(“review”)**
revisión técnica	**(“technical review”)**

Objetivos de Aprendizaje para “Revisiones”

5.1 Tareas del Analista de Pruebas Técnicas en las Revisiones

APT-5.1.1	(K2)	Explicar por qué la preparación de la revisión es importante para el Analista de Pruebas Técnicas.

5.2 Uso de Listas de Comprobación en las Revisiones

APT-5.2.1	(K4)	Analizar un diseño arquitectónico e identificar los problemas según una lista de comprobación proporcionada en el programa de estudio.
APT-5.2.2	(K4)	Analizar una sección de código o pseudocódigo e identificar problemas según una lista de comprobación proporcionada en el programa de estudio

5.1 Tareas del Analista de Pruebas Técnicas en las Revisiones

Los Analistas de Pruebas Técnicas deben ser participantes activos en el proceso de revisión técnica, aportando sus puntos de vista específicos. Todos los participantes en la revisión deben tener una formación formal en materia de revisión para comprender mejor sus respectivos roles y deben estar comprometidos con los beneficios de una revisión técnica bien realizada. Esto incluye mantener una relación de trabajo constructiva con los autores al describir y discutir los comentarios de la revisión. Para una descripción detallada de las revisiones técnicas, que incluye numerosas listas de comprobación de revisiones, véase [Wiegers02]. Los Analistas de Pruebas Técnicas suelen participar en revisiones técnicas e inspecciones en las que aportan un punto de vista operativo (de comportamiento) que los desarrolladores pueden pasar por alto. Además, los Analistas de Pruebas Técnicas juegan un rol importante en la definición, aplicación y mantenimiento de las listas de revisión y en el suministro de información sobre la severidad de los defectos.

Independientemente del tipo de revisión que se lleve a cabo, el Analista de Pruebas Técnicas debe disponer de tiempo suficientes para prepararse. Esto incluye tiempo para revisar el producto del trabajo, tiempo para comprobar la documentación cruzada para verificar la consistencia y tiempo para determinar lo que podría faltar en el producto del trabajo. Sin el tiempo de preparación adecuado, la revisión puede convertirse en un ejercicio de edición más que en una verdadera revisión. Una buena revisión incluye la comprensión de lo que está escrito, la determinación de lo que falta, la comprobación de la corrección con respecto a los aspectos técnicos y la verificación de que el producto descrito es coherente con otros productos ya desarrollados o en desarrollo. Por ejemplo, al revisar un plan de nivel de prueba de integración, el Analista de Pruebas Técnicas también debe tener en cuenta los elementos que se están integrando. ¿Están preparados para la integración? ¿Hay dependencias que deban documentarse? ¿Hay datos disponibles para probar los puntos de integración? Una revisión no se limita al producto de trabajo que se está revisando. También debe tener en cuenta la interacción de ese elemento con otros elementos del sistema.

5.2 Uso de Listas de Comprobación en las Revisiones

Las listas de comprobación se utilizan durante las revisiones para recordar a los participantes que deben verificar puntos específicos durante la revisión. Las listas de comprobación también pueden ayudar a despersonalizar la revisión, por ejemplo, "esta es la misma lista de comprobación que utilizamos para todas las revisiones, y no nos dirigimos sólo a su producto de trabajo". Las listas de comprobación pueden ser genéricas y utilizarse para todas las revisiones o concentrarse en características de calidad o áreas específicas. Por ejemplo, una lista de comprobación genérica podría verificar el uso correcto de los términos "deberá" y "debería", verificar el formato adecuado y

otros elementos de conformidad similares. Una lista de comprobación podría concentrarse en problemas de seguridad o de eficiencia de desempeño.

Las listas de comprobación más útiles son las desarrolladas gradualmente por una organización individual porque reflejan:

- La naturaleza del producto.
- El entorno de desarrollo local.
    - El personal.
    - Las herramientas.
    - Las prioridades.
- Historial de éxitos y defectos anteriores.
- Problemas particulares (por ejemplo, eficiencia de desempeño, seguridad).

Las listas de comprobación deben adaptarse a la organización y, tal vez, al proyecto concreto. Las listas de comprobación de este capítulo son ejemplos.

5.2.1 Revisión de la Arquitectura

La arquitectura del software consiste en los conceptos o propiedades fundamentales de un sistema, plasmados en sus elementos, relaciones y en los principios de su diseño y evolución [ISO 42010].

Las listas de comprobación[34] utilizadas para las revisiones de la arquitectura del comportamiento temporal de los sitios web podrían, por ejemplo, incluir la verificación de la correcta implementación de los siguientes elementos, que se citan de [Web-2]:

- Agrupar conexiones - reducir la sobrecarga de tiempo de ejecución asociada al establecimiento de conexiones a la base de datos mediante el establecimiento de un agrupamiento compartido de conexiones.
- Equilibrado de carga: repartir la carga de manera uniforme entre un conjunto de recursos.
- Procesamiento distribuido.
- Almacenamiento en memoria intermedia: uso de una copia local de los datos para reducir el tiempo de acceso.
- Instanciación lenta.
- Concurrencia de transacciones.
- Aislamiento de procesos entre el Procesamiento Transaccional en Línea (OLTP por sus siglas en inglés) y el Procesamiento Analítico en Línea (OLAP por sus siglas en inglés).
- Replicación de datos

[34] "lista de comprobación" es la traducción del término "checklist".

5.2.2 Revisión de Código

Las listas de comprobación para las revisiones de código son necesariamente de bajo nivel y resultan más útiles cuando son específicas del lenguaje. La inclusión de antipatrones a nivel de código es útil, especialmente para los desarrolladores de software menos experimentados.

Las listas de comprobación[35] utilizadas para la revisión del código podrían incluir los siguientes elementos:

1. Estructura
    - El código, ¿implementa completa y correctamente el diseño?
    - ¿El código se ajusta a algún estándar de codificación adecuado?
    - ¿El código está bien estructurado, tiene un estilo consistente y un formato homogéneo?
    - ¿Existen procedimientos no llamados o innecesarios o algún código inaccesible?
    - ¿Existen stubs o rutinas de prueba sobrantes en el código?
    - ¿Se puede reemplazar algún fragmento de código por llamadas a componentes externos reutilizables o a funciones de librería?
    - ¿Hay algún bloque de código repetido que pueda condensarse en un único procedimiento?
    - ¿El uso del almacenamiento es eficiente?
    - ¿Se utilizan símbolos en lugar de constantes de "números mágicos" o constantes de cadena?
    - ¿Hay algún módulo excesivamente complejo que deba reestructurarse o dividirse en varios módulos?
2. Documentación
    - ¿El código está clara y adecuadamente documentado con un estilo de comentarios fácil de mantener?
    - ¿Todos los comentarios son coherentes con el código?
    - ¿La documentación se ajusta a los estándares correspondientes?
3. Variables
    - ¿Todas las variables están correctamente definidas con nombres significativos, consistentes y claros?
    - ¿Hay variables redundantes o no utilizadas?
4. Operaciones Aritméticas
    - ¿El código evita la comparación de números en coma flotante para la igualdad?
    - ¿El código evita sistemáticamente los errores de redondeo?

[35] La pregunta del examen proporcionará un subconjunto de la lista de comprobación con la que responder a la pregunta.

- ¿El código evita las sumas y restas en números con magnitudes muy diferentes?
- ¿Se prueban los divisores para ver si son cero o tienen ruido?

5. Bucles y Ramas
    - ¿Todos los bucles, ramas y construcciones lógicas están completos, son correctos y están debidamente anidados?
    - ¿Se prueban primero los casos más comunes en las cadenas IF-ELSEIF?
    - ¿Están cubiertos todos los casos en un bloque IF-ELSEIF o CASE, incluidas las cláusulas ELSE o DEFAULT?
    - ¿Todas las sentencias CASE tienen un valor por defecto?
    - ¿Las condiciones de terminación del bucle son obvias y se pueden alcanzar invariablemente?
    - ¿Los índices o subíndices se inicializan correctamente, inmediatamente antes del bucle?
    - ¿Se pueden colocar fuera de los bucles las sentencias que están dentro de los mismos?
    - ¿El código del bucle evita manipular la variable índice o utilizarla al salir del bucle?
6. Programación Defensiva
    - ¿Se prueban los índices, punteros y subíndices con respecto a las fronteras de los arreglos, registros o archivos?
    - ¿Se prueba la validez y la completitud de los datos importados y de los argumentos de entrada?
    - ¿Están asignan todas las variables de salida?
    - ¿Se opera con el elemento de dato correcto en cada sentencia?
    - ¿Se libera cada asignación de memoria?
    - ¿Se utilizan tiempos de espera o trampas de error para el acceso a dispositivos externos?
    - ¿Se comprueba la existencia de los archivos antes de intentar acceder a ellos?
    - ¿Todos los archivos y dispositivos quedan en el estado correcto al finalizar el programa?

6. Herramientas de Prueba y Automatización - 180 minutos

Palabras Clave

captura/reproducción	**(“capture/playback”)**
prueba guiada por datos	**(“data-driven testing”)**
emulador	**(“emulator”)**
inyección de defecto	**(“fault injection”)**
siembra de defecto	**(“fault seeding”)**
prueba guiada por palabra clave	**(“keyword-driven testing”)**
prueba basada en modelos (PBM)	**(“model-based testing (MBT)”)**
simulador	**(“simulator”)**
ejecución de prueba	**(“test execution”)**

Objetivos de Aprendizaje para “Herramientas de Prueba y Automatización”

6.1 Definición del Proyecto de Automatización de la Prueba

APT-6.1.1	(K2)	Resumir las actividades que realiza el Analista de Pruebas Técnicas al establecer un proyecto de automatización de la prueba.
APT-6.1.2	(K2)	Resumir las diferencias entre la automatización basada en datos y la basada en palabras clave.
APT-6.1.3	(K2)	Resumir los problemas técnicos comunes que hacen que los proyectos de automatización no logren el retorno de la inversión previsto.
APT-6.1.3	(K3)	Construir palabras clave basadas en un proceso de negocio determinado.

6.2 Herramientas de Prueba Específicas

APT-6.2.1	(K2)	Resumir el propósito de las herramientas para la siembra de defectos y la inyección de defectos.
APT-6.2.2	(K2)	Resumir las principales características y problemas de implementación de las herramientas de prueba de rendimiento.
APT-6.2.3	(K2)	Explicar el propósito general de las herramientas utilizadas para las pruebas basadas en la web.
APT-6.2.4	(K2)	Explicar cómo las herramientas apoyan la práctica de las pruebas basadas en modelos.
APT-6.2.5	(K2)	Describir la finalidad de las herramientas utilizadas para apoyar las pruebas de componentes y el proceso de construcción.
APT-6.2.6	(K2)	Describir el propósito de las herramientas utilizadas para apoyar las pruebas de aplicaciones móviles.

6.1 Definición del Proyecto de Automatización de la Prueba

Para que sean rentables, las herramientas de prueba (y en particular las que apoyan la ejecución de la prueba), deben tener una arquitectura y un diseño cuidadosos. La implementación de una estrategia de automatización de la ejecución de pruebas sin una arquitectura sólida suele dar como resultado un conjunto de herramientas que es costoso de mantener, insuficiente para el propósito e incapaz de lograr el retorno de la inversión previsto.

Un proyecto de automatización de la prueba debe tener en cuenta un proyecto de desarrollo de software. Esto incluye la necesidad de documentación de la arquitectura, documentación detallada del diseño, revisiones del diseño y del código, la prueba de integración de componentes, así como la prueba final de sistema. La prueba puede retrasarse o complicarse innecesariamente cuando se utiliza un código de automatización de la prueba inestable o impreciso.

Existen múltiples tareas que el Analista de Pruebas Técnicas puede realizar en relación con la automatización de la ejecución de prueba. Estas incluyen:

- Determinar quién será el responsable de la ejecución de prueba (posiblemente en coordinación con un Jefe de Prueba).
- Seleccionar la herramienta adecuada para la organización, el calendario, las competencias del equipo y los requisitos de mantenimiento (se debe tener en cuenta que esto podría significar la decisión de crear una herramienta para utilizarla en lugar de adquirirla).
- Definir los requisitos de la interfaz entre la herramienta de automatización y otras herramientas como la gestión de la prueba, la gestión de defectos y las herramientas utilizadas para la integración continua.
- Desarrollar adaptadores que puedan ser necesarios para crear una interfaz entre la herramienta de ejecución de la prueba y el software que se está probando.
- Seleccionar el enfoque de automatización, es decir, basado en palabras clave o en datos (véase el apartado 6.1.1).
- Trabajar con el Jefe de Prueba para estimar el coste de la implementación, incluida la formación. En el desarrollo ágil de software, este aspecto suele discutirse y acordarse en las reuniones de planificación del proyecto/esprint con todo el equipo.
- Programar[36] el proyecto de automatización y asignar el tiempo para el mantenimiento.
- Formar a los Analistas de Prueba y a los Analistas de Negocio en el uso y suministro de datos para la automatización.
- Determinar cómo y cuándo se ejecutarán pruebas automatizadas.
- Determinar cómo se combinarán los resultados de prueba automatizados con los resultados de prueba manuales.

En los proyectos con un fuerte énfasis en la automatización de la prueba, un ingeniero de automatización de la prueba puede encargarse de muchas de estas actividades (véase el programa de estudio del Ingeniero de Automatización de la Prueba [CT_TAE_SYL] para más detalles). Ciertas tareas organizativas pueden ser asumidas por un Jefe de Prueba según las necesidades y preferencias del proyecto. En el desarrollo ágil de software, la asignación de estas tareas a los roles suele ser más flexible y menos formal.

[36] "programar" es la traducción del término "scheduling" en el presente contexto.

Estas actividades y las decisiones resultantes influirán en la escalabilidad y mantenibilidad de la solución de automatización. Hay que dedicar suficiente tiempo a investigar las opciones, investigar las herramientas y tecnologías disponibles y comprender los planes futuros de la organización.

6.1.1 Selección del Enfoque de Automatización

Esta sección tiene en cuenta los siguientes factores que afectan al enfoque de automatización de la prueba:

- Automatizar a través de la IGU ("Graphic User Interface - GUI"), IPA ("Application Programming Interface - API"), ILC ("Comand Line Interface - CLI").
- Aplicar un enfoque guiado por datos.
- Aplicar un enfoque guiado por palabra clave.
- Tratamiento de los fallos del software.
- Tener en cuenta el estado del sistema.

El programa de estudio de Ingeniero de Automatización de la Prueba [CT_TAE_SYL] incluye más detalles sobre la selección de un enfoque de automatización.

6.1.1.1 Automatizar a través de la IGU ("Graphic User Interface - GUI"), IPA ("Application Programming Interface - API"), ILC ("Comand Line Interface - CLI")

La automatización de la prueba no se limita a probar a través de la IGU ("Graphic User Interface - GUI"). También existen herramientas que ayudan a automatizar las pruebas a nivel de la IPA ("Application Programming Interface - API"), a través de una interfaz de línea de comandos ILC ("Comand Line Interface - CLI") y otros puntos de interfaz en el software sujeto a prueba. Una de las primeras decisiones que debe tomar el Analista de Pruebas Técnicas es determinar la interfaz más eficaz a la que se debe acceder para automatizar la prueba. Las herramientas de ejecución de la prueba en general requieren el desarrollo de adaptadores a estas interfaces. La planificación deberá tener en cuenta el esfuerzo para el desarrollo del adaptador.

Una de las dificultades de probar a través de la IGU es la tendencia a que ésta cambie a medida que evoluciona el software. Dependiendo de la forma en que se diseñe el código de automatización de las pruebas, esto puede suponer una importante carga de mantenimiento. Por ejemplo, el uso de la capacidad de captura/reproducción de una herramienta de automatización de pruebas puede dar lugar a casos de prueba automatizados (a menudo denominados guiones de prueba) que ya no se ejecutan como se desea si la IGU cambia. Esto se debe a que el guion grabado captura las interacciones con los objetos gráficos cuando el probador ejecuta el software manualmente. Si los objetos a los que se accede cambian, los guiones grabados también pueden necesitar una actualización para reflejar esos cambios.

Las herramientas de captura/reproducción pueden utilizarse como un punto de partida conveniente para desarrollar guiones de automatización. El probador registra una sesión de prueba y el guion grabado se modifica para mejorar la mantenibilidad (por ejemplo, sustituyendo secciones del guion grabado por funciones reutilizables).

6.1.1.2 Aplicar un enfoque guiado por datos

Dependiendo del software que se esté probando, los datos utilizados para cada prueba pueden ser diferentes aunque los pasos de prueba ejecutados sean prácticamente idénticos (por ejemplo, probar el tratamiento de errores para un campo de entrada introduciendo múltiples valores no válidos y comprobando el error devuelto para cada uno). Resulta ineficaz desarrollar y mantener un guion de prueba automatizado para cada uno de estos valores a probar. Una solución técnica común a este problema es trasladar los datos de los guiones a un almacén externo, como una hoja de cálculo o una base de datos. Se escriben funciones para acceder a los datos específicos para cada ejecución del

guion de prueba, lo que permite que un único guion trabaje a través de un conjunto de datos de prueba que proporciona los valores de entrada y los valores de resultado esperados (por ejemplo, un valor mostrado en un campo de texto o un mensaje de error). Este enfoque se denomina guiado por datos.

Cuando se utiliza este enfoque, además de los guiones de prueba que procesan los datos suministrados, se necesitan un arnés y una infraestructura para apoyar la ejecución del guion o conjunto de guiones. Los datos reales de la hoja de cálculo o la base de datos son creados por Analistas de Prueba que están familiarizados con la función de negocio del software. En el Desarrollo Ágil de Software, el representante de negocio (por ejemplo, el Propietario de Producto) también puede participar en la definición de los datos, en particular para las pruebas de aceptación. Esta división del trabajo permite a los responsables del desarrollo de guiones de prueba (por ejemplo, el Analista de Pruebas Técnicas) concentrarse en la implementación de guiones de automatización mientras el Analista de Pruebas mantiene la propiedad de la prueba en sí. En la mayoría de los casos, el Analista de la Prueba será el responsable de ejecutar los guiones de prueba una vez que la automatización se haya implementado y probado.

6.1.1.3 Aplicar un enfoque guiado por palabra clave

Otro enfoque, denominado por palabras clave o por palabras de acción, va un paso más allá al separar también del guion de prueba la acción que debe realizarse sobre los datos de prueba suministrados [Buwalda01]. Para lograr esta separación adicional, se crea un lenguaje de alto nivel que es descriptivo y no directamente ejecutable. Se pueden definir palabras clave para las acciones de alto y bajo nivel. Por ejemplo, las palabras clave del proceso de negocio podrían incluir "IniciarSesión", "CrearUsuario" y "EliminarUsuario". Estas palabras clave describen las acciones de alto nivel que se realizarán en el dominio de la aplicación. Las acciones de nivel inferior denotan la interacción con la propia interfaz del software. Palabras clave como "PulsarBotón", "SeleccionarDeLaLista", o "RecorrerÁrbol" pueden utilizarse para probar las capacidades de la IGU ("Interfaz Gráfica de Usuario - GUI") que no encajan claramente en las palabras clave del proceso de negocio. Las palabras clave pueden contener parámetros, por ejemplo, la palabra clave "LogIn" podría tener dos parámetros: nombre_de_usuario y contraseña.

Una vez definidas las palabras clave y los datos que se van a utilizar, el automatizador de la prueba (por ejemplo, el analista de Pruebas Técnicas o el Ingeniero de Automatización de la Prueba) traduce las palabras clave del proceso de negocio y las acciones de nivel inferior en código de automatización de la prueba. Las palabras clave y las acciones, junto con los datos a utilizar, pueden almacenarse en hojas de cálculo o introducirse mediante herramientas específicas que admiten la automatización de la prueba por palabras clave. El marco de trabajo de automatización de la prueba implementa la palabra clave como un conjunto de una o más funciones o guiones ejecutables. Las herramientas leen los casos de prueba escritos con palabras clave y llaman a las funciones o guiones de prueba adecuados que las implementan. Los ejecutables se implementan de forma altamente modular para permitir una fácil asignación a palabras clave específicas. Se necesitan competencias de programación para implementar estos guiones modulares.

Esta separación entre el conocimiento de la lógica de negocio y la programación real necesaria para implementar los guiones de automatización de pruebas proporciona el uso más eficaz de los recursos de prueba. El Analista de Pruebas Técnicas, en su rol de automatizador de pruebas, puede aplicar eficazmente las competencias de programación sin tener que convertirse en un experto en el dominio de muchas áreas del negocio.

Separar el código de los datos modificables ayuda a aislar la automatización de los cambios, mejorando la mantenibilidad general del código y mejorando el retorno de la inversión en automatización.

6.1.1.4 Tratamiento de los fallos del software

En el diseño de automatización de la prueba, es importante anticipar y manejar los fallos del software. Si se produce un fallo, el automatizador de la prueba debe determinar qué debe hacer el software de

ejecución de la prueba. ¿Debe registrarse el fallo y continuar las pruebas? ¿Deben terminarse las pruebas? ¿Puede manejarse el fallo con una acción específica (como hacer clic en un botón de un cuadro de diálogo) o quizás añadiendo un retraso en la prueba? Los fallos de software no gestionados pueden corromper los resultados de las pruebas posteriores, así como causar un problema con la prueba que se estaba ejecutando cuando se produjo el fallo.

6.1.1.5 Tener en cuenta el estado del sistema

También es importante tener en cuenta el estado del sistema al principio y al final de cada prueba. Puede ser necesario asegurar que el sistema vuelva a un estado predefinido tras la compleción de la ejecución de prueba. Esto permitirá que un conjunto de pruebas automatizadas se ejecute repetidamente sin intervención manual para restablecer el sistema a un estado conocido. Para ello, la automatización de la prueba puede tener que, por ejemplo, borrar los datos que creó o alterar el estado de los registros en una base de datos. El marco de automatización debe asegurar que se ha llevado a cabo una terminación adecuada al final de las pruebas (es decir, el cierre de la sesión una vez completadas las pruebas).

6.1.2 Modelado de Procesos de Negocio para la Automatización

Para implementar un enfoque guiado por palabras clave para la automatización de la prueba, los procesos de negocio que se van a probar se deben modelar en el lenguaje de palabras clave de alto nivel. Es importante que el lenguaje sea intuitivo para sus usuarios, que probablemente sean los Analistas de Prueba que trabajan en el proyecto o, en el caso de desarrollo ágil de software, el representante del negocio (por ejemplo, el Propietario del Producto).

En general, las palabras clave se utilizan para representar interacciones de negocio de alto nivel con un sistema. Por ejemplo, "Cancelar_Pedido" puede requerir la comprobación de la existencia del pedido, la verificación de los derechos de acceso de la persona que solicita la cancelación, la visualización del pedido a cancelar y la solicitud de confirmación de la cancelación. El analista de pruebas utiliza las secuencias de palabras clave (por ejemplo, "IniciarSesión", "Seleccionar_Pedido", "Cancelar_Pedido") y los datos de prueba pertinentes para especificar los casos de prueba.

Entre los problemas que hay que tener en cuenta se encuentran los siguientes:

- Cuanto más detalladas sean las palabras clave, más específicos serán los escenarios que se pueden cubrir, pero el lenguaje de alto nivel puede resultar más complejo de mantener.
- Permitir a los Analistas de Prueba especificar acciones de bajo nivel ("PulsarBotón", "SeleccionarDeLista", etc.) hace que las pruebas de palabras clave sean mucho más capaces de manejar diferentes situaciones. Sin embargo, como estas acciones están ligadas directamente a la IGU, también puede hacer que las pruebas requieran más mantenimiento cuando se produzcan cambios.
- El uso de palabras clave agregadas puede simplificar el desarrollo pero complicar el mantenimiento. Por ejemplo, puede haber seis palabras clave que creen colectivamente un registro. Sin embargo, crear una palabra clave de alto nivel que llame a las seis palabras clave puede no ser el enfoque más eficiente en general.
- Por mucho que se analice el lenguaje de las palabras clave, a menudo habrá ocasiones en las que se necesitarán palabras clave nuevas y modificadas. Hay dos dominios separados en una palabra clave (es decir, la lógica de negocio que hay detrás y la funcionalidad de automatización para ejecutarla). Por lo tanto, debe crearse un proceso que se ocupe de ambos dominios.

La automatización de la prueba guiada por palabras clave puede reducir significativamente los costes de mantenimiento de la automatización de la prueba. Puede ser más costosa de establecer inicialmente, pero es probable que sea más barata en general si el proyecto dura lo suficiente.

El programa de estudio de Ingeniero de Automatización de la Prueba [CT_TAE_SYL] incluye más detalles sobre el modelado de los procesos de negocio para la automatización.

6.2 Herramientas de Prueba Específicas

Esta sección contiene información general sobre las herramientas que probablemente utilizará un Analista de Pruebas Técnicas más allá de lo que se trata en el programa de estudio de nivel básico [CTFL_SYL].

Se debe tener en cuenta que la información detallada sobre las herramientas se proporciona en los siguientes programas de estudio de ISTQB®:

- Prueba de Aplicaciones Móviles [CT_MAT_SYL]
- Prueba de Rendimiento [CT_PT_SYL]
- Prueba Basadas en Modelos [CT_MBT_SYL]
- Ingeniero de Automatización de la Prueba [CT_TAE_SYL]

6.2.1 Herramientas de Siembra de Defectos

Las herramientas de siembra de defectos modifican el código sujeto a prueba (posiblemente utilizando algoritmos predefinidos) para comprobar la cobertura alcanzada por las pruebas especificadas. Si se aplican de forma sistemática, permiten evaluar la calidad de las pruebas (es decir, su capacidad para detectar los defectos insertados) y, en su caso, mejorarla.

Las herramientas de siembra de defectos suelen ser utilizadas por el Analista de Pruebas Técnicas, pero también pueden ser utilizadas por el desarrollador al probar un código recién desarrollado.

6.2.2 Herramientas de Inyección de Defectos

Las herramientas de inyección de defectos suministran deliberadamente entradas incorrectas al software para asegurar que éste pueda hacer frente al defecto. Las entradas inyectadas provocan condiciones negativas, que deben hacer que se ejecute (y se pruebe) el tratamiento de errores. Esta interrupción del flujo de ejecución normal del código también aumenta la cobertura de código.

Las herramientas de inyección de defectos suelen ser utilizadas por el Analista de Pruebas Técnicas, pero también pueden ser utilizadas por el desarrollador al probar un código recién desarrollado.

6.2.3 Herramientas de Prueba de Rendimiento

Las herramientas de prueba de rendimiento tienen las siguientes funciones principales:

- Generar carga.
- Proporcionar la medición, monitorización, visualización y análisis de la respuesta del sistema a una carga determinada, dando una idea del comportamiento de los recursos de los componentes del sistema y de la red.

La generación de carga se realiza mediante la implementación de un perfil operativo predefinido (véase la sección 4.9) como un guion. El guion puede capturarse inicialmente para un solo usuario (posiblemente utilizando una herramienta de captura/reproducción) y luego se implementa para el perfil operativo especificado utilizando la herramienta de prueba de rendimiento. Esta implementación debe tener en cuenta la variación de datos por transacción (o conjuntos de transacciones).

Las herramientas de rendimiento generan una carga mediante la simulación de un gran número de usuarios múltiples (usuarios "virtuales") que siguen sus perfiles operativos determinados para realizar

tareas que incluyen la generación de volúmenes específicos de datos de entrada. En comparación con los guiones de automatización de la ejecución de pruebas individuales, muchos guiones de pruebas de rendimiento reproducen la interacción del usuario con el sistema a nivel de protocolo de comunicaciones y no simulando la interacción del usuario a través de una interfaz gráfica de usuario. Esto suele reducir el número de "sesiones" separadas que se necesitan durante la prueba. Algunas herramientas de generación de carga también pueden manejar la aplicación utilizando su interfaz de usuario para medir con mayor precisión los tiempos de respuesta mientras el sistema se encuentra sometido a carga.

Una herramienta de prueba de rendimiento toma una amplia gama de medidas para permitir el análisis durante o después de la ejecución de la prueba. Entre las métricas típicas que se toman y los informes que se proporcionan figuran:

- Número de usuarios simulados a lo largo de la prueba.
- Número y tipo de transacciones generadas por los usuarios simulados y la tasa de llegada de las transacciones.
- Tiempos de respuesta a determinadas solicitudes de transacciones realizadas por los usuarios.
- Informes y gráficos de la carga frente a los tiempos de respuesta
- Informes sobre el uso de los recursos (por ejemplo, uso a lo largo del tiempo con valores mínimos y máximos).

Entre los factores importantes que hay que tener en cuenta en la implementación de las herramientas de prueba de rendimiento se encuentran:

- El hardware y el ancho de banda de la red necesarios para generar la carga.
- La compatibilidad de la herramienta con el protocolo de comunicaciones utilizado por el sistema sujeto a prueba.
- La flexibilidad de la herramienta para permitir que se implementen fácilmente diferentes perfiles operativos.
- Las facilidades de monitorización, análisis y elaboración de informes requeridas.

Las herramientas de prueba de rendimiento suelen adquirirse en lugar de desarrollarse internamente debido al esfuerzo que requiere su desarrollo. Sin embargo, puede ser conveniente desarrollar una herramienta de rendimiento específica si las restricciones técnicas impiden utilizar un producto disponible, o si el perfil de carga y las facilidades que hay que proporcionar son sencillas en comparación con el perfil de carga y las facilidades que proporcionan las herramientas comerciales. En el programa de estudio sobre Prueba de Rendimiento [CT_PT_SYL] se ofrecen más detalles sobre las herramientas de prueba de rendimiento.

6.2.4 Herramientas para Probar Sitios Web

Existe una gran variedad de herramientas especializadas, tanto de código abierto como comerciales, para probar sitios web. La siguiente lista muestra la finalidad de algunas de las herramientas de prueba comunes basadas en la web:

- Las herramientas de prueba de hipervínculos se utilizan para explorar y probar que no hay hipervínculos rotos o ausentes en un sitio web.
- Los comprobadores de HTML y XML son herramientas que comprueban el cumplimiento de los estándares HTML y XML de las páginas que crea un sitio web.

- Herramientas de prueba de rendimiento para probar cómo reaccionará el servidor cuando se conecte un gran número de usuarios. Herramientas de ejecución de automatización ligeras que funcionan con diferentes navegadores.
- Herramientas para explorar el código del servidor, comprobando si hay archivos huérfanos (no enlazados) a los que haya accedido previamente el sitio web.
- Comprobadores ortográficos específicos de HTML.
- Herramientas de registro de entrada de hojas de estilo en cascada (CSS).
- Herramientas que comprueban incumplimientos de los estándares, por ejemplo, las normas de accesibilidad de la Sección 508 en EE.UU. o la M/376 en Europa.
- Herramientas que encuentran una variedad de problemas de seguridad.

Las siguientes son fuentes adecuadas de herramientas de código abierto para probar la web:

- El Consorcio World Wide Web (W3C) [Web-3] Esta organización establece los estándares de Internet y proporciona una variedad de herramientas para comprobar los errores con respecto a esos estándares.
- El Grupo de Trabajo Web Hypertext Application Technology Working Group (WHATWG) (WHATWG) [Web-5]. Esta organización establece los estándares de HTML. Disponen de una herramienta que realiza la validación de HTML [Web-6].

Algunas herramientas que incluyen un motor de araña web [37] también pueden proporcionar información sobre el tamaño de las páginas y sobre el tiempo necesario para descargarlas, y sobre si una página está presente o no (por ejemplo, el error HTTP 404). Esto proporciona información útil para el desarrollador, el administrador de sitio web[38] y el probador.

Los Analistas de Pruebas y los Analistas de Pruebas Técnicas utilizan estas herramientas principalmente durante la prueba de sistema.

6.2.5 Herramientas de Apoyo a la Prueba Basada en Modelos

La prueba basada en modelos (PBM) es una técnica que utiliza un modelo, como una máquina de estados finitos, para describir el comportamiento previsto en tiempo de ejecución de un sistema controlado por software. Las herramientas comerciales de PBM (véase [Utting07]) suelen proporcionar un motor que permite al usuario "ejecutar" el modelo. Los hilos de ejecución interesantes pueden guardarse y utilizarse como casos de prueba. Otros modelos ejecutables, como las redes de Petri y los diagramas de estado, también admiten PBM.

Los Modelos PBM (y las herramientas) pueden utilizarse para generar grandes conjuntos de hilos de ejecución distintos. Las herramientas PBM también pueden ayudar a reducir el gran número de caminos posibles que pueden generarse en un modelo. Las pruebas realizadas con estas herramientas pueden proporcionar una visión diferente del software que se va a probar. Esto puede dar lugar al descubrimiento de defectos que podrían haberse pasado por alto en las pruebas funcionales.

En el programa de estudio sobre pruebas basadas en modelos [CT_MBT_SYL] se ofrecen más detalles sobre las herramientas de prueba basadas en modelos.

[37] “araña web” es la traducción del término “web spider”.

[38] “administrador de sitio web” es la traducción del término “webmaster”.

6.2.6 Herramientas de Prueba de Componentes y de Construcción

Aunque las herramientas de prueba de componentes y de automatización de la construcción son herramientas para desarrolladores, en muchos casos las utilizan y mantienen los Analistas de Pruebas Técnicas, especialmente en el contexto del desarrollo ágil.

Las herramientas de prueba de componente suelen ser específicas del lenguaje que se utiliza para programar un componente. Por ejemplo, si se utilizara Java como lenguaje de programación, se podría utilizar JUnit para automatizar la prueba de componente. Muchos otros lenguajes tienen sus propias herramientas de prueba especiales; éstas se denominan colectivamente marcos xUnit. Un marco de este tipo genera objetos de prueba para cada clase que se crea, simplificando así las tareas que el programador necesita hacer al automatizar la prueba de componente.

Algunas herramientas de automatización de la construcción permiten activar automáticamente una nueva construcción cuando se modifica un componente. Una vez completada la construcción, otras herramientas ejecutan automáticamente las pruebas de los componentes. Este nivel de automatización en torno al proceso de construcción suele verse en un entorno de integración continua.

Cuando se configura correctamente, este conjunto de herramientas puede tener un efecto positivo en la calidad de las construcciones que se lanzan a prueba. Si un cambio realizado por un programador introduce defectos de regresión en la construcción, normalmente hará que fallen algunas de las pruebas automatizadas, lo que provocará una investigación inmediata de la causa de los fallos antes de que la construcción se entregue al entorno de prueba.

6.2.7 Herramientas de Apoyo a la Prueba de Aplicaciones Móviles

Los emuladores y simuladores son herramientas de uso frecuente para apoyar la prueba de aplicaciones móviles.

6.2.7.1 Simuladores

Un simulador móvil modela el entorno de ejecución de la plataforma móvil. Las aplicaciones que se prueban en un simulador se compilan en una versión dedicada, que funciona en el simulador pero no en un dispositivo real. Los simuladores se utilizan como sustitutos de los dispositivos reales en las pruebas, pero suelen limitarse a las pruebas funcionales iniciales y a la simulación de muchos usuarios virtuales en las pruebas de carga. Los simuladores son relativamente sencillos (en comparación con los emuladores) y pueden ejecutar las pruebas más rápidamente que un emulador. Sin embargo, la aplicación que se prueba en un simulador difiere de la que se distribuirá.

6.2.7.2 Emuladores

Un emulador móvil modela el hardware y utiliza el mismo entorno de ejecución que el hardware físico. Las aplicaciones compiladas para ser desplegadas y probadas en un emulador también podrían ser utilizadas por el dispositivo real.

Sin embargo, un emulador no puede reemplazar completamente a un dispositivo porque el emulador puede comportarse de manera diferente al dispositivo móvil que intenta imitar. Además, es posible que algunas prestaciones no sean compatibles, como el contacto múltiple[39] y el acelerómetro. Esto se debe en parte a las limitaciones de la plataforma utilizada para ejecutar el emulador.

6.2.7.3 Aspectos Comunes

Los simuladores y emuladores se utilizan a menudo para reducir el coste de los entornos de prueba sustituyendo a los dispositivos reales. Los simuladores y emuladores son útiles en la fase inicial del

[39] “contacto múltiple” es la traducción del término “(multi)touch”.

desarrollo, ya que suelen integrarse en los entornos de desarrollo y permiten una rápida implantación, prueba y monitorización de las aplicaciones. El uso de un emulador o simulador requiere ponerlo en marcha, instalar en él la aplicación necesaria y, a continuación, probar la aplicación como si estuviera en el dispositivo real. Cada entorno de desarrollo del sistema operativo móvil suele venir con su propio emulador o simulador incluido. También existen emuladores y simuladores de terceros.

Por lo general, los emuladores y simuladores permiten configurar diversos parámetros de uso. Estos ajustes pueden incluir la emulación de la red a diferentes velocidades, la intensidad de la señal y las pérdidas de paquetes, el cambio de orientación, la generación de interrupciones y los datos de localización del GPS. Algunos de estos ajustes pueden ser muy útiles porque pueden ser difíciles o costosos de reproducir con dispositivos reales, como las posiciones globales del GPS o la intensidad de la señal.

El programa de estudio de Pruebas de Aplicaciones Móviles [CT_MAT_SYL] incluye más detalles.

7. Referencias

7.1 Normas

Los siguientes estándares se mencionan en estos capítulos respectivos.

REFERENCIA	ESTÁNDAR	CAPÍTULO
[DO-178C]	DO-178C - Software Considerations in Airborne Systems and Equipment Certification, RTCA, 2011	Capítulo 2
[ISO 9126]	ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality	Capítulo 2, Apéndice A
[ISO 25010]	ISO/IEC 25010: 2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models	Capítulos 1 y 4, Apéndice A
[ISO 29119]	ISO/IEC/IEEE 29119-4:2015, Software and systems engineering - Software testing - Part 4: Test techniques	Capítulo 2
[ISO 42010]	ISO/IEC/IEEE 42010:2011, Systems and software engineering - Architecture description	Capítulo 5
[IEC 61508]	IEC 61508-5:2010, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, Part 5: Examples of methods for the determination of safety integrity levels	Capítulo 2
[ISO 26262]	ISO 26262-1:2018, Road vehicles — Functional safety, Parts 1 to 12.	Capítulo 2
[IEC 62443-3-2]	IEC 62443-3-2:2020, Security for industrial automation and control systems - Part 3-2: Security risk assessment for system design	Capítulo 4

7.2 Documentos de ISTQB®

REFERENCIA	DOCUMENTO
[CTAL_TTA_OVIEW]	ISTQB® Technical Test Analyst Advanced Level Overview v4.0
[CT_SEC_SYL]	Security Testing Syllabus, Version 2016
[CT_TAE_SYL]	Test Automation Engineer Syllabus, Version 2017
[CTFL_SYL]	Foundation Level Syllabus, Version 2018
[CT_PT_SYL]	Performance Testing Syllabus, Version 2018
[CT_MBT_SYL]	Model-Based Testing Syllabus, Version 2015
[CTAL_TM_SYL]	Test Manager Syllabus, Version 2012
[CT_MAT_SYL]	Mobile Application Testing Syllabus, 2019
[ISTQB_GLOSSARY]	Glossary of Terms used in Software Testing, Version 3.5, 2020
[CT_AuT_SYL]	Automotive Software Tester, Version 2018

7.3 Libros y Artículos

REFERENCIA	DOCUMENTO
[Andrist20]	Björn Andrist and Viktor Sehr, C++ High Performance: Master the art of optimizing the functioning of your C++ code, 2nd Edition, Packt Publishing, 2020
[Beizer90]	Boris Beizer, "Software Testing Techniques Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3
[Buwalda01]	Hans Buwalda, “Integrated Test Design and Automation”, Addison-Wesley Longman, 2001, ISBN 0-201-73725-6
[Kaner02]	Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”; Wiley, 2002, ISBN: 0-471-08112-4
[McCabe76]	Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976, pp. 308-320
[Utting07]	Mark Utting, Bruno Legeard, “Practical Model-Based Testing: A Tools Approach”, Morgan-Kaufmann, 2007, ISBN: 978-0-12-372501-1
[Whittaker04]	James Whittaker and Herbert Thompson, “How to Break Software Security”, Pearson / Addison-Wesley, 2004, ISBN 0-321-19433-0
[Wiegers02]	Karl Wiegers, "Peer Reviews in Software: A Practical Guide", Addison-Wesley, 2002, ISBN 0-201-73485-0

7.4 Otras Referencias

Las siguientes referencias apuntan a información disponible en Internet. Aunque estas referencias fueron comprobadas en el momento de la publicación de este programa de estudios de nivel avanzado, el ISTQB® no se hace responsable si las referencias dejan de estar disponibles.

REFERENCIA	ESTÁNDAR	CAPÍTULO
[Web-1]	http://www.nist.gov (NIST National Institute of Standards and Technology)	**4**
[Web-2]	http://www.codeproject.com/KB/architecture/SWArchitectureReview.aspx	**5**
[Web-3]	http://www.W3C.org	**6**
[Web-4]	https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project	**4**
[Web-5]	https://whatwg.org	**6**
[Web-6]	https://validator.w3.org/	**6**
[Web-7]	https://dl.acm.org/doi/abs/10.1145/3340433.3342822	**2**

8. Apéndice A: Resumen de las Características de Calidad

En la siguiente tabla se comparan las características de calidad descritas en la norma ISO 9126-1, ya sustituida (tal y como se utilizaba en la versión de 2012 del programa de estudios de Analista de Pruebas Técnicas), con las de la norma ISO 25010, más reciente (tal y como se utiliza en la última versión del programa).

Se debe tener en cuenta que la adecuación funcional y la usabilidad están cubiertas como parte del programa de estudios de Analista de Pruebas.

ISO/IEC 25010		ISO/IEC 9126-1		Notas
INGLÉS	**ESPAÑOL**	**INGLÉS**	**ESPAÑOL**	
Functional Suitability	Adecuación funcional	Functionality	Funcionalidad	La nueva denominación es más precisa y evita la confusión con otras acepciones de "funcionalidad".
Functional completeness	Completitud funcional			Cobertura de las necesidades planteadas.
Functional correctness	Corrección funcional	Accuracy	Exactitud	Más general que la exactitud.
Functional appropriateness	Pertinencia funcional	Suitability	Adecuación	Cobertura de las necesidades implícitas.
		Interoperability	Interoperabilidad	Se ha desplazado a Compatibilidad.
		Security	Seguridad	Ahora es una característica.
Performance efficiency	Eficiencia de desempeño	Efficiency	Eficiencia	Renombrada para no entrar en conflicto con la definición de eficiencia de la norma ISO/IEC 25062.
Time behaviour	Comportamiento temporal	Time behaviour	Comportamiento temporal	
Resource utilization	Utilización de recursos	Resource utilization	Utilización de recursos	
Capacity	Capacidad			Nueva subcaracterística.
Compatibility	Compatibilidad			Nueva característica.
Coexistence	Coexistencia	Coexistence	Coexistencia	Se ha desplazado desde Portabilidad
Interoperability	Interoperabilidad			Movido de Funcionalidad (Analista de Prueba)

ISO/IEC 25010		ISO/IEC 9126-1		Notas
INGLÉS	**ESPAÑOL**	**INGLÉS**	**ESPAÑOL**	
Usability	Usabilidad			El problema de calidad implícito se ha hecho explícito.
Appropriateness recognizability	Capacidad de reconocimiento de la pertinencia	Understandability	Capacidad de ser entendido	El nuevo nombre es más preciso.
Learnability	Capacidad de ser aprendido	Learnability	Capacidad de ser aprendido	
Operability	Capacidad de ser operado	Operability	Capacidad de ser operado	
User error protection	Protección ante error de usuario			Nueva subcaracterística.
User interface aesthetics	Estética de interfaz de usuario.	Attractiveness	Atractivo	El nuevo nombre es más preciso.
Accessibility	Accesibilidad			Nueva subcaracterística.
Reliability	Fiabilidad	Reliability	Fiabilidad	
Maturity	Madurez	Maturity	Madurez	
Availability	Disponibilidad			Nueva subcaracterística.
Fault tolerance	Tolerancia a defectos	Fault tolerance	Tolerancia a defectos	
Recoverability	Recuperabilidad	Recoverability	Capacidad de recuperación	

ISO/IEC 25010		ISO/IEC 9126-1		Notas
INGLÉS	**ESPAÑOL**	**INGLÉS**	**ESPAÑOL**	
Security	Seguridad	Security	Seguridad	No hay subcaracterísticas precedentes.
Confidentiality	Confidencialidad			No hay subcaracterísticas precedentes.
Integrity	Integridad			No hay subcaracterísticas precedentes.
Non-repudiation	No repudio			No hay subcaracterísticas precedentes.
Accountability	Responsabilidad			No hay subcaracterísticas precedentes.
Authenticity	Autenticidad			No hay subcaracterísticas precedentes.