Probador Certificado del ISTQB®

Programa de Estudio de Nivel Avanzado

Analista de Pruebas Técnicas

Traducción realizada por el

Spanish Software Testing Qualifications Board

Versión ES 001.42

Basada en el Programa de Estudio

“Certified Tester - Advanced Level Syllabus, Test Manager, Version 2012”

International Software Testing Qualifications Board

Nota sobre derechos de propiedad intelectual

Subgrupo de Trabajo de Analista de Pruebas de Nivel Avanzado: Graham Bath (Presidente), Paul Jorgensen, Jamie Mitchell; 2010-2012.

Historial de Revisiones

Versión	Fecha	Observaciones
ISEB v1.1	04SEP01	ISEB Practitioner Syllabus.
ISTQB 1.2E	SEP03	Programa de Estudio de Nivel Avanzado ISTQB de EOQ-SG.
V2007	12OCT07	Programa de Estudio de Probador Certificado de Nivel Avanzado, versión 2007.
D100626	26JUN10	Incorporación de cambios según aceptados en 2009, separación de cada capítulo para los distintos módulos.
D101227	27DIC10	Aceptación de cambios de formato y correcciones que no afectan al significado de las frases.
Borrador V1	17SEP11	Primera versión del nuevo programa de estudio de Analistas de Pruebas Técnicas basada en el documento de alcance acordado. Revisión AL WG.
Borrador V2	20NOV11	Versión revisión NB.
Alfa 2012	09MAR12	Incorporación de todos los comentarios derivados de los NB recibidos a partir de la entrega de octubre.
Beta 2012	07ABR12	Versión Beta enviada a GA.
Beta 2012	08JUN12	Versión editada entregada a NB.
Beta 2012	27JUN12	Incorporación de comentarios EWG y Glosario.
RC 2012	15AGO12	Versión candidata de entrega - ediciones NB finales incluidas.
RC 2012	02SEP12	Incorporación de comentarios de BNLTB y Stuart Reid incluyen comprobación cruzada de Paul Jorgensen.
GA 2012	19OCT12	Ediciones finales y limpieza para entrega GA.

Índice general

Agradecimientos

Este documento ha sido elaborado por un equipo principal del Subgrupo de Trabajo de Nivel Avanzado - Analista de Pruebas del International Software Testing Qualifications Board: Graham Bath (Presidente), Paul Jorgensen, Jamie Mitchell.

El equipo principal agradece las sugerencias y aportaciones del equipo de revisión y de los comités nacionales.

Cuando se finalizó el programa de estudio del Nivel Avanzado, formaban parte del Grupo de Trabajo de Nivel Avanzado los siguientes miembros (por orden alfabético):

Graham Bath, Rex Black, Maria Clara Choucair, Debra Friedenberg, Bernard Homès (Vicepresidente), Paul Jorgensen, Judy McKay, Jamie Mitchell, Thomas Mueller, Klaus Olsen, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Jan Sabak, Hans Schaefer, Mike Smith (Presidente), Geoff Thompson, Erik van Veenendaal, Tsuyoshi Yumoto.

Las siguientes personas participaron en la revisión, comentarios y votación del presente programa de estudio:

Dani Almog, Graham Bath, Franz Dijkman, Erwin Engelsma, Mats Grindal, Dr. Suhaimi Ibrahim, Skule Johansen, Paul Jorgensen, Kari Kakkonen, Eli Margolin, Rik Marselis, Judy McKay, Jamie Mitchell, Reto Mueller, Thomas Müller, Ingvar Nordstrom, Raluca Popescu, Meile Posthuma, Michael Stahl, Chris van Bael, Erik van Veenendaal, Rahul Verma, Paul Weymouth, Hans Weiberg, Wenqiang Zheng, Shaomin Zhu.

Este documento fue hecho público formalmente por la Asamblea General del ISTQB® el 19 de octubre de 2012.

Gustavo Márquez Sosa (Spanish Software Testing Qualifications Board) ha revisado la traducción al español de este programa de estudio.

0. Introducción a Este Programa de Estudio

0.1 Objetivo de este Documento

Este programa de estudio constituye la base para la Cualificación Internacional de Pruebas de Software de Nivel Avanzado para Analista de Pruebas Técnicas. El ISTQB® ofrece el presente programa de estudio:

1. a los Comités Nacionales, para que lo traduzcan a su idioma local y para acreditar a los proveedores de formación. Los Comités Nacionales podrán adaptar el programa de estudio a las necesidades específicas de su idioma y modificar las referencias para adaptarlas a sus publicaciones locales.
2. a los Comités de Exámenes, para que puedan crear preguntas de examen en su idioma local que se adapten a los objetivos de aprendizaje de cada programa de estudio.
3. a los proveedores de formación, para que elaboren los materiales didácticos y determinen las metodologías de enseñanza apropiadas.
4. a los candidatos a la certificación, para que se preparen para el examen (como parte de un curso de formación o de manera 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 Generalidades

El Nivel Avanzado consta de tres programas de estudio independientes.

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

El documento Descripción del Nivel Avanzado (“Advanced Level Syllabus (2012) Overview”) [ISTQB_AL_OVIEW] incluye la siguiente información:

- Resultados de negocio de cada programa de estudio.
- Resumen de cada programa de estudio.
- Relaciones entre los programas de estudio.
- Descripción de los niveles de conocimiento (niveles K).
- Anexos.

0.3 Objetivos de Aprendizaje Objeto de Examen

Los objetivos de aprendizaje respaldan los resultados de negocio y se utilizan para elaborar el examen para lograr la Certificación de Analista de Pruebas Técnicas de Nivel Avanzado. Por lo general, todas las partes de este programa de estudio pueden ser objeto de examen en el nivel K1. Es decir, el candidato reconocerá, memorizará y recordará un término o concepto. Los objetivos de aprendizaje correspondientes a los niveles K2, K3 y K4 se muestran al principio de cada capítulo.

0.4 Expectativas

Algunos de los objetivos de aprendizaje para el Analista de Pruebas Técnicas asumen que se dispone de experiencia básica en los siguientes ámbitos:

- Conceptos generales de programación.
- Conceptos generales de arquitecturas de sistemas.

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

Palabras clave

riesgo de producto, análisis del riesgo, evaluación del riesgo, identificación del riesgo, nivel de riesgo, mitigación del riesgo, pruebas basadas en el riesgo

Objetivos de Aprendizaje para las Tareas del Analista de Pruebas Técnicas en las Pruebas Basadas en el Riesgo

1.3 Evaluación del Riesgo

APT-1.3.1 (K2) Resumir los factores de riesgo genéricos que el Analista de Pruebas Técnicas normalmente debe tener en cuenta.

Objetivos de Aprendizaje Comunes

El siguiente objetivo de aprendizaje se refiere al contenido cubierto en más de una sección de este capítulo.

APT-1.x.1 (K2) Resumir las actividades del Analista de Pruebas Técnicas siguiendo un enfoque basado en el riesgo para planificar y ejecutar pruebas.

1.1 Introducción

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

Dada su competencia técnica específica, los Analistas de Pruebas Técnicas participan activamente en las siguientes tareas de pruebas basadas en riesgos:

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

Estas tareas se llevan a cabo de forma iterativa durante todo el proyecto para abordar los riesgos de producto emergentes y los cambios en las prioridades y para evaluar y comunicar periódicamente el estado de los riesgos.

Los Analistas de Pruebas Técnicas trabajan dentro del marco de trabajo de las pruebas basadas en riesgos establecido para el proyecto por el Jefe de Pruebas aportando su conocimiento de los riesgos técnicos que son inherentes al proyecto, tales como los riesgos asociados a la seguridad, la fiabilidad y el rendimiento del sistema.

1.2 Identificación del Riesgo

Si se recurre a una muestra lo más amplia posible de implicados, el proceso de identificación de riesgos tendrá más posibilidades de detectar el máximo número posible de riesgos significativos. Dado que los Analistas de Pruebas Técnicas poseen habilidades técnicas específicas, resultan particularmente idóneos para llevar a cabo revisiones expertas, tormenta de ideas [2] con colaboradores[3] y también para analizar las experiencias presentes y pasadas con el fin de establecer dónde se encuentran las áreas probables de riesgo de producto. En particular, los Analistas de Pruebas Técnicas trabajan en estrecha colaboración con sus pares[4] (homólogos) técnicos (por ejemplo, desarrolladores, arquitectos, ingenieros de operaciones) para establecer las áreas de riesgo técnico.

Entre los riesgos que se pueden identificar están incluidos los siguientes:

- Riesgos de rendimiento (por ejemplo, incapacidad para lograr tiempos de respuesta en condiciones de carga alta).
- Riesgos de seguridad (por ejemplo, revelación de datos sensibles a través de ataques contra la seguridad).
- Riesgos de fiabilidad (por ejemplo, aplicación incapaz de cumplir la disponibilidad prevista en el Acuerdo de Nivel de Servicio).

Las áreas de riesgo relativas a las características de la calidad específicas del software se abordan en los capítulos correspondientes de este Programa de Estudio.

1.3 Evaluación del Riesgo

Si bien la identificación del riesgo consiste en identificar la mayor cantidad de riesgos pertinentes que sea posible, la evaluación del riesgo es el estudio de estos riesgos identificados con el objetivo de clasificar cada riesgo y determinar la probabilidad e impacto asociado a cada uno de estos riesgos.

[1] “pruebas basadas en el riesgo” y “pruebas basadas en riesgos” son sinónimos.
[2] “tormenta de ideas” es la traducción de “brainstorming”.
[3] “colaborador” es la traducción de “co-worker”.
[4] “par” es la traducción de “peer”.

Determinar el nivel de riesgo normalmente conlleva evaluar, para cada elemento de riesgo, la probabilidad de ocurrencia y el impacto en caso de ocurrencia. La probabilidad de ocurrencia se interpreta, normalmente, como la probabilidad de que el problema potencial pueda existir en el sistema sujeto a prueba.

El Analista de Pruebas Técnicas contribuye a la detección y comprensión del riesgo técnico potencial asociado a cada elemento de riesgo, mientras que el Analista de Pruebas contribuye a la comprensión del posible impacto de negocio del problema en caso de producirse.

Entre los factores genéricos que normalmente deben tenerse en cuenta se encuentran:

- Complejidad tecnológica.
- Complejidad de la estructura del código.
- Conflicto entre implicados en lo que respecta a requisitos técnicos.
- Problemas de comunicación derivados de la distribución geográfica de la organización de desarrollo.
- Herramientas y tecnología.
- Tiempo, recursos y presión por parte de la dirección.
- Ausencia de aseguramiento de la calidad en una fase temprana.
- Alta tasa de cambio en requisitos técnicos.
- Detectado gran número de defectos asociados a características de calidad técnica.
- Problemas técnicos de interfaz e integración.

Dada la información disponible sobre los riesgos, el Analista de Pruebas Técnicas establece los niveles de riesgo de acuerdo con las directrices establecidas por el Jefe de Pruebas. Así por ejemplo, el Jefe de Pruebas puede establecer que el riesgo se debe categorizar utilizando valores del 1 al 10, siendo 1 el riesgo más alto.

1.4 Mitigación del Riesgo

Durante el proyecto, los Analistas de Pruebas Técnicas influyen en cómo las pruebas responden ante los riesgos identificados. Normalmente esto implica lo siguiente:

- Reducir el riesgo ejecutando las pruebas más importantes y poner en marcha actividades de mitigación y contingencia apropiadas previstas en la estrategia de prueba y en el plan de prueba.
- Evaluar los riesgos basándose en información adicional recopilada a medida que el proyecto avanza, y emplear dicha información para llevar a cabo medidas de mitigación destinadas a reducir la probabilidad o el impacto de los riesgos previamente identificados y analizados.

2. Pruebas Basadas en la Estructura - 225 minutos

Palabras clave
condición atómica, pruebas de condición, pruebas de flujo de control, pruebas de Condición/Decisión, pruebas de condición múltiple, pruebas de camino, cortocircuitado, pruebas de sentencia, técnica basada en la estructura

Objetivos de Aprendizaje para Pruebas Pasadas en la Estructura

2.2 Pruebas de Condición
APT-2.2.1 (K2) Comprender cómo lograr una cobertura de condición y por qué esta puede constituir una prueba menos rigurosa que la cobertura de decisión.

2.3 Pruebas de Condición/Decisión
APT-2.3.1 (K3) Formular casos de prueba aplicando la técnica de diseño de pruebas de Condición/Decisión para alcanzar un nivel de cobertura establecido.

2.4 Pruebas de Cobertura de Condición / Decisión Modificada (CC/DM)
APT-2.4.1 (K3) Formular casos de prueba aplicando la técnica de diseño de pruebas de cobertura de condición / decisión modificada (CC/DM) para alcanzar un nivel de cobertura establecido.

2.5 Pruebas de Condición Múltiple
APT-2.5.1 (K3) Formular casos de prueba aplicando la técnica de diseño de pruebas de condición múltiple para alcanzar un nivel de cobertura establecido.

2.6 Pruebas de Camino
APT-2.6.1 (K3) Formular casos de prueba aplicando la técnica de diseño de pruebas de camino.

2.7 Pruebas API
APT-2.7.1 (K2) Comprender la aplicabilidad de las pruebas API y los tipos de defectos que detectan.

2.8 Selección de Técnica Basada en la Estructura
APT-2.8.1 (K4) Seleccionar una técnica basada en la estructura adecuada de acuerdo a una situación de proyecto dada.

2.1 Introducción

Este capítulo describe principalmente las técnicas de diseño de pruebas basadas en la estructura, también conocidas como técnicas de prueba de caja blanca o basadas en código. Estas técnicas utilizan el código, los datos y la arquitectura y/o el flujo de sistema como base para el diseño de las pruebas. Cada técnica específica permite derivar casos de prueba de forma sistemática y se centra en un aspecto específico de la estructura a considerar. Las técnicas aportan criterios de cobertura que tienen que medirse y asociarse con un objetivo definido por cada proyecto u organización. El hecho de lograr una cobertura completa no significa que el conjunto de pruebas íntegro sea completo, sino más bien que la técnica utilizada ya no sugiere ninguna prueba útil para la estructura en cuestión.

Con la excepción de la cobertura de condición, las técnicas de diseño de pruebas basadas en la estructura que se contemplan en este programa de estudio son más rigurosas que las técnicas de cobertura de sentencia y decisión previstas en el Programa de Estudio Fundamentos [ISTQB_FL_SYL].

En este Programa de Estudio se consideran las siguientes técnicas:

- Pruebas de condición.
- Pruebas de Condición/Decisión.
- Pruebas de condición modificada/cobertura de decisión (CM/CD).
- Pruebas de condición múltiple.
- Pruebas de camino.
- Pruebas API.

Las primeras cuatro técnicas de la lista anterior se basan en predicados de decisiones y, en general, detectan los mismos tipos de defectos. Independientemente de la complejidad del predicado de decisión, este evaluará como VERDADERO o FALSO (“TRUE” o “FALSE”), adoptando uno de estos dos caminos a través del código. Se detecta un defecto cuando no se toma el camino previsto debido a que hay un predicado de decisión complejo que no realiza la evaluación según lo previsto.

En general las primeras cuatro técnicas resultan sucesivamente más profundas, requieren la definición de más pruebas para lograr su cobertura prevista y para detectar instancias más sutiles de este tipo de defecto.

Remítase a [Bath08], [Beizer90], [Beizer95], [Copeland03] y [Koomen06].

2.2 Pruebas de Condición

En comparación con las pruebas de decisión (ramas), que consideran la decisión completa como un todo y evalúa los resultados VERDADERO (“TRUE”) y FALSO (“FALSE”) en casos de prueba independientes, las pruebas de condición tienen en cuenta cómo se adopta una decisión. Cada predicado de decisión[5] consta de una o más condiciones “atómicas”, cada una de las cuales se evalúa a un valor discreto booleano. Estos valores se combinan lógicamente para determinar el resultado final de la decisión. Cada condición atómica debe evaluarse en ambos sentidos por los casos de prueba para lograr este nivel de cobertura.

Aplicabilidad
Las pruebas de condición probablemente solo resultan interesantes en teoría, a la vista de las dificultades que se indican a continuación. No obstante, es necesario comprenderlas para alcanzar mayores niveles de cobertura que se construyen a partir de ellas.

Limitaciones/Dificultades

[5] “predicado de decisión” es la traducción de “decision predicate”.

Cuando una decisión incluye dos o más condiciones atómicas, una mala elección en la selección de los datos de prueba durante el diseño de las pruebas puede resultar en la consecución de una cobertura de condición pero no de una cobertura de decisión. Así por ejemplo, asumamos el predicado de decisión “A y B”.

	A	B	A y B
Prueba 1	FALSO	VERDADERO	FALSO
Prueba 2	VERDADERO	FALSO	FALSO

Para alcanzar una cobertura de condición al 100%, ejecutamos las dos pruebas indicadas en la tabla más arriba. Si bien estas dos pruebas logran una cobertura de condición al 100%, no logran una cobertura de decisión, ya que en ambos casos el predicado evalúa como FALSO.

Cuando una decisión consta de una única condición atómica, las pruebas de condición son idénticas a las pruebas de decisión.

2.3 Pruebas de Condición/Decisión

Las pruebas de Condición/Decisión especifican que las pruebas deben alcanzar una cobertura de condición (ver más arriba) y también exigen que se cumpla la cobertura de decisión (remítase al Programa de Estudio Fundamentos [ISTQB_FL_SYL]). Una cuidada elección de los valores de los datos de prueba de las condiciones atómicas puede suponer la consecución de este nivel de cobertura sin añadir casos de prueba adicionales más allá de los necesarios para alcanzar la cobertura de condición.

En el ejemplo a continuación se prueba el mismo predicado de decisión (visto más arriba), “A y B”. La cobertura de Condición/Decisión puede lograrse con el mismo número de pruebas seleccionando distintos valores de prueba.

	A	B	A y B
Prueba 1	FALSO	VERDADERO	FALSO
Prueba 2	VERDADERO	FALSO	FALSO

Por lo tanto, esta técnica puede tener una ventaja en cuanto a la eficiencia.

Aplicabilidad
Este nivel de cobertura debería tenerse en cuenta cuando el código objeto de las pruebas es importante pero no crítico.

Limitaciones/Dificultades
Dado que pueden requerir más casos de prueba que las pruebas a nivel de decisión, estas pruebas pueden resultar problemáticas cuando el tiempo plantea un problema.

2.4 Pruebas de Cobertura de Condición / Decisión Modificada (CC/DM)

Esta técnica ofrece un nivel más alto de cobertura del flujo de control. Asumiendo N condiciones atómicas únicas, normalmente puede lograrse una CC/DM en N+1 casos de prueba únicos. Las pruebas CC/DM logran una cobertura de Condición/Decisión, pero para ello requieren que se cumpla también lo siguiente:

1. Al menos una prueba en la que el resultado de la decisión cambiaría si la condición atómica X fuera VERDADERO.
2. Al menos una prueba en la que el resultado de la decisión cambiaría si la condición atómica X fuera FALSO.

3. Cada condición atómica tiene pruebas que cumplen los requisitos 1 y 2.

	A	B	C	(A o B) y C
Prueba 1	VERDADERO	FALSO	VERDADERO	VERDADERO
Prueba 2	FALSO	VERDADERO	VERDADERO	VERDADERO
Prueba 3	FALSO	FALSO	VERDADERO	FALSO
Prueba 4	VERDADERO	FALSO	FALSO	FALSO

En el ejemplo anterior, se alcanza tanto una cobertura de decisión (el resultado del predicado de decisión es tanto VERDADERO como FALSO) como una cobertura de condición (A, B y C son evaluados como VERDADERO y FALSO).

En la Prueba 1, A es VERDADERO y la salida es VERDADERO. Si A se cambia a FALSO (como en la Prueba 3, sin cambiar ningún otro valor), el resultado será FALSO.
En la Prueba 2, B es VERDADERO y la salida es VERDADERO. Si B se cambia a FALSO (como en la Prueba 3, sin cambiar ningún otro valor), el resultado será FALSO.

En la Prueba 1, C es VERDADERO y la salida es VERDADERO. Si C se cambia a FALSO (como en la Prueba 4, sin cambiar ningún otro valor), el resultado será FALSO.

Aplicabilidad
Esta técnica se utiliza mucho en el sector aeroespacial y en muchos otros sistemas de seguridad crítica. Debería utilizarse en software de seguridad crítica en el que cualquier fallo podría provocar una catástrofe.

Limitaciones/Dificultades
Alcanzar la cobertura CC/DM puede ser complicado cuando se dan múltiples ocurrencias de un mismo término específico en una expresión, cuando esto sucede, se dice que el término está "acoplado". En función de la sentencia de decisión del código, es posible que no se pueda modificar el valor del término acoplado de manera que por sí solo altera el resultado de la decisión. Un enfoque para abordar este problema es especificar que solo deben probarse a nivel CC/DM las condiciones atómicas no acopladas. Otro enfoque consiste en analizar caso por caso las decisiones en las que se produce el acoplamiento.

Algunos lenguajes de programación y/o intérpretes han sido diseñados para exhibir un comportamiento propio de un cortocircuito (cortocircuitado[6]) cuando evalúan una sentencia de decisión compleja en el código. Es decir, el código en ejecución puede no evaluar una expresión entera si el resultado final de la evaluación se puede determinar después de evaluar solo una parte de la expresión. Por ejemplo, si estamos evaluando la decisión "A y B", no hay motivo para evaluar B si A se evalúa como FALSO. Ningún valor de B puede alterar el valor final, por lo que el código puede ahorrar tiempo de ejecución no evaluando B. Los cortocircuitos pueden afectar a la capacidad para alcanzar la cobertura CC/DM dado que es posible que algunas pruebas necesarias no sean realizables.

2.5 Pruebas de Condición Múltiple

En casos poco habituales, podría ser necesario probar todas las combinaciones posibles de valores que puede incluir una decisión. Este nivel de prueba exhaustivo se denomina cobertura de condición múltiple. El número de pruebas necesarias dependerá del número de condiciones atómicas de la sentencia de decisión y podrá establecerse calculando 2^n, donde n es el número de condiciones atómicas no acopladas. Siguiendo con el mismo ejemplo, las siguientes pruebas son necesarias para alcanzar una cobertura de condición múltiple:

	A	B	C	(A o B) y C
Prueba 1	VERDADERO	VERDADERO	VERDADERO	VERDADERO

[6] "cortocircuitado" es la traducción de "short-circuiting".

Prueba 2	VERDADERO	VERDADERO	FALSO	FALSO
Prueba 3	VERDADERO	FALSO	VERDADERO	VERDADERO
Prueba 4	VERDADERO	FALSO	FALSO	FALSO
Prueba 5	FALSO	VERDADERO	VERDADERO	VERDADERO
Prueba 6	FALSO	VERDADERO	FALSO	FALSO
Prueba 7	FALSO	FALSO	VERDADERO	FALSO
Prueba 8	FALSO	FALSO	FALSO	FALSO

Si el lenguaje utiliza el cortocircuitado, el número de casos de prueba efectivos a menudo se reducirá, dependiendo del orden y la agrupación de las operaciones lógicas que se lleven a cabo en las condiciones atómicas.

Aplicabilidad
Tradicionalmente, esta técnica se ha utilizado para probar software embebido previsto para mantenerse en ejecución, de manera fiable, sin fallos abruptos durante largos periodos de tiempo (por ejemplo, conmutadores telefónicos con una vida útil prevista de 30 años). Es probable que este tipo de prueba sea reemplazado por pruebas CC/DM en las aplicaciones más críticas.

Limitaciones/Dificultades
Dado que el número de casos de prueba puede derivarse directamente a partir de una tabla de verdad que contiene todas las condiciones atómicas, este nivel de cobertura puede determinarse fácilmente. No obstante, el elevado número de casos de prueba requerido hace que la cobertura CC/DM sea más aplicable en la mayoría de las situaciones.

2.6 Pruebas de Camino

Las pruebas de camino consisten en identificar caminos a través del código y a continuación crear pruebas para cubrirlos. Desde una perspectiva conceptual, podría ser útil probar todos los caminos únicos a través del sistema. En cualquier sistema no trivial, no obstante, el número de casos de pruebas podría ser excesivamente alto dada la naturaleza de las estructuras de bucle.

Dejando a un lado el problema de los bucles indefinidos, no obstante, es realista llevar a cabo algunas pruebas de camino. Para aplicar esta técnica [Beizer90] recomienda crear pruebas que siguen muchos caminos a través del módulo, desde la entrada hasta la salida. Para simplificar lo que podría ser una tarea compleja, recomienda que se pueda hacer automáticamente, utilizando el siguiente procedimiento:

1. Seleccione como primer camino el camino que sea más sencillo y funcionalmente apreciable desde la entrada hasta la salida.
2. Seleccione cada camino adicional como una pequeña variación del camino anterior. Intente cambiar solo una rama del camino que sea diferente para cada prueba sucesiva. Cuando sea posible, favorezca los caminos cortos frente a los caminos largos. Favorezca los caminos que tienen un sentido funcional frente a los que no lo tienen.
3. Seleccione caminos que no tienen un sentido funcional solo cuando la cobertura así lo requiera. Beizer observa en esta norma que estos caminos podrían ser superfluos y deberían cuestionarse.
4. Utilice la intuición para elegir los caminos (es decir, qué caminos tienen más probabilidad de ejecutarse).

Cabe destacar que con esta estrategia es probable que algunos segmentos de caminos se ejecuten más de una vez. El punto clave de esta estrategia es probar todas las ramas posibles a través del código por lo menos una vez y posiblemente muchas veces.

Aplicabilidad
Las pruebas parciales de camino - como se ha definido anteriormente -, a menudo, se realizan en software de seguridad crítica. Constituyen un buen complemento de los otros métodos cubiertos en este capítulo porque observa los caminos a través del software en lugar de ver solo la forma en que se toman las decisiones.

Limitaciones/Dificultades
Si bien puede utilizarse un gráfico de flujo de control para establecer los caminos, en la práctica se requiere una herramienta para calcularlos en el caso de módulos complejos.

Cobertura
Crear suficientes pruebas para cubrir todos los caminos (ignorando bucles) garantiza que se logra tanto la cobertura de sentencia como la cobertura de rama. Las pruebas de camino aportan pruebas más profundas que la cobertura de rama, con un incremento relativamente pequeño en el número de pruebas. [NIST 96]

2.7 Pruebas de Interfaz de Programación de Aplicaciones (“API”)

Una Interfaz de Programación de Aplicaciones (IPA) [7]es un código que permite la comunicación entre distintos procesos, programas y/o sistemas. Las Interfaces de Programación de Aplicaciones (“API’s”), con frecuencia, se utilizan en una relación cliente/servidor donde un proceso suministra cierto tipo de funcionalidad a otros procesos.

En ciertos aspectos las pruebas de Interfaz de Programación de Aplicaciones (“API”) son bastante parecidas a las pruebas de interfaz gráfica de usuario (GUI). Su foco está en la evaluación de los valores de entrada y datos devueltos.

Las pruebas negativas suelen ser cruciales cuando se trata con la Interfaz de Programación de Aplicaciones (“API”). Los programadores que utilizan Interfaces de Programación de Aplicaciones (API’s) para acceder a servicios externos a su propio código, pueden intentar utilizar las interfaces de las Interfaces de Programación de Aplicaciones (API’s) de formas distintas a las previstas. Esto significa que una gestión robusta de errores es fundamental para evitar una operación incorrecta. Es posible que sea necesario realizar pruebas combinatorias de varias interfaces distintas porque las Interfaces de Programación de Aplicaciones (API’s), a menudo, se utilizan de forma conjunta con otras Interfaces de Programación de Aplicaciones (API’s), y porque una única interfaz puede contener varios parámetros, donde los valores de estos se pueden combinar de muchas formas.

Con frecuencia, las Interfaces de Programación de Aplicaciones (API’s) están acopladas débilmente, lo que plantea la posibilidad real de perder transacciones o fallos imprevistos de corta duración[8]. Esto requiere realizar pruebas en profundidad de los mecanismos de recuperación y reintento[9]. Una organización que provea una Interfaz de Programación de Aplicaciones (“API”) debe asegurar que todos los servicios tienen una disponibilidad muy alta, lo cual a menudo requiere que el distribuidor de la Interfaz de Programación de Aplicaciones (“API”) realice pruebas de fiabilidad estrictas además de disponer de un soporte de infraestructura.

Aplicabilidad
Las pruebas de Interfaz de Programación de Aplicaciones (“API”) están adquiriendo cada vez más relevancia a medida que hay más sistemas distribuidos o que utilizan el procesamiento remoto como una forma de descargar de trabajo a otros procesadores. Algunos ejemplos de ello son las llamadas de sistemas operativos, arquitecturas orientadas a servicios (“Service-Oriented Architectures - SOA”), llamadas a procedimientos remotos (“remote procedure calls - RPC”), servicios Web y prácticamente cualquier otra aplicación distribuida. Las pruebas de Interfaz de Programación de Aplicaciones (“API”) son especialmente aplicables para probar sistemas de sistemas.

Limitaciones/Dificultades

[7] Interfaz de Programación de Aplicaciones (IPA) es la traducción de “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”)**.
[8] “fallo imprevisto de corta duración” es la traducción de “timing glitch”.
[9] “reintento” es la traducción de “retry”. Este término no pertenece al Diccionario de la RAE, sin embargo se utiliza debido a la extensión de su uso y a que se ha identificado en textos técnicos y académicos.

Para probar una Interfaz de Programación de Aplicaciones (“API”) de forma directa, normalmente, es necesario que el Analista de Pruebas Técnicas utilice herramientas especializadas. Dado que normalmente no hay una interfaz gráfica directa asociada a la Interfaz de Programación de Aplicaciones (“API”), es posible que se deban utilizar herramientas para configurar el entorno inicial, preparar los datos, invocar a la Interfaz de Programación de Aplicaciones (“API”) y determinar el resultado.

Cobertura
Las pruebas de Interfaz de Programación de Aplicaciones (“API”) son una descripción de un tipo de pruebas, no denotan ningún nivel de cobertura específico. Como mínimo las pruebas Interfaz de Programación de Aplicaciones (“API”) deberían incluir el ejercicio de todas las llamadas a la Interfaz de Programación de Aplicaciones (“API”) así como todos los valores válidos e inválidos razonables.

Tipos de defectos
Los tipos de defectos que pueden encontrarse con las pruebas Interfaz de Programación de Aplicaciones (“API”) son bastante dispares. Los problemas de interfaz son frecuentes, al igual que los problemas con la gestión de datos, los problemas asociados al tiempo, la pérdida de transacciones y duplicación de transacciones.

2.8 Selección de una Técnica Basada en la Estructura

El contexto del sistema a prueba determinará el nivel de cobertura de pruebas basadas en la estructura que se debe alcanzar. Cuanto más crítico sea el sistema, más alto será el nivel de cobertura necesario. En general, cuanto más alto sea el nivel de cobertura requerido, serán necesarios más tiempo y recursos para alcanzar ese nivel.

En ocasiones el nivel de cobertura requerido puede derivarse de las normas aplicables al sistema software. Por ejemplo, si el software se utilizara en un entorno aéreo, es posible que deba ajustarse a la norma DO-178B (en Europa, ED12B.) Esta norma contiene las siguientes cinco condiciones de fallo:

A. Catastrófico[10]: el fallo puede provocar la ausencia de una función crítica necesaria para volar o aterrizar la aeronave de forma segura.
B. Peligroso[11]: el fallo puede tener un gran impacto negativo en la seguridad física o en el rendimiento.
C. Importante[12]: el fallo es significativo, pero menos grave que A o B.
D. Menor[13]: el fallo es evidente, pero tiene menor impacto que C.
E. Sin efecto[14]: el fallo no afecta a la seguridad física.

Si el sistema software está clasificado como nivel A, debe probarse para la cobertura CC/DM. Si es de nivel B, debe probarse para la cobertura de nivel de decisión aunque la CC/DM es opcional. El nivel C requiere como mínimo una cobertura de sentencia.

Asimismo, la IEC-61508 es una norma internacional para la seguridad funcional de sistemas programables, electrónicos y relacionados con la seguridad física. Este estándar ha sido adaptado a muchas áreas distintas, incluyendo el sector automovilístico, ferroviario, procesos de fabricación, plantas de energía nuclear y maquinaria. La criticidad se define utilizando una escala graduada de Nivel de Integridad de la Seguridad Física (SIL) (siendo 1 el menos crítico y 4 el más crítico) y se recomiendan las siguientes coberturas:

1. Cobertura de sentencia y cobertura de rama recomendadas.
2. Cobertura de sentencia altamente recomendada, cobertura de rama recomendada.
3. Cobertura de sentencia y cobertura de rama altamente recomendadas.
4. CC/DM altamente recomendada.

[10] “catastrófico” es la traducción de “catastrophic”.
[11] “peligroso” es la traducción de “hazardous”.
[12] “importante” es la traducción de “major”.
[13] “menor” es la traducción de “minor”.
[14] “sin efecto” es la traducción de “no effect”.

En los sistemas modernos, no es habitual que todo el procesamiento se haga en un único sistema. Las pruebas Interfaz de Programación de Aplicaciones (“API”) deberían instituirse si parte del procesamiento va a ser en remoto. La criticidad del sistema debería determinar cuánto esfuerzo debería invertirse en las pruebas Interfaz de Programación de Aplicaciones (“API”).

Como siempre, el contexto del sistema de software sujeto a pruebas debería guiar al Analista de Pruebas Técnicas en lo que respecta a los métodos utilizados en las pruebas.

3. Técnicas Analíticas - 255 minutos

Palabras clave

análisis del flujo de control, complejidad ciclomática, análisis del flujo de datos, par definición-uso, análisis dinámico, fuga de memoria, pruebas de integración de a pares de elementos, pruebas de integración de proximidad, análisis estático, puntero sin referencia

Objetivos de Aprendizaje para Técnicas Analíticas

3.2 Análisis Estático

APT-3.2.1 (K3) Utilizar el análisis del flujo de control para detectar si el código presenta alguna anomalía del flujo de control.

APT-3.2.2 (K3) Utilizar análisis del flujo de datos para detectar si el código presenta alguna anomalía del flujo de datos.

APT-3.2.3 (K3) Proponer formas para mejorar la mantenibilidad del código aplicando análisis estático.

APT-3.2.4 (K2) Explicar el uso de los grafos de llamadas para establecer estrategias de pruebas de integración.

3.3 Análisis Dinámico

APT-3.3.1 (K3) Especificar los objetivos a alcanzar por el uso de análisis estáticos.

3.1 Introducción

Hay dos tipos de análisis: análisis estático y análisis dinámico.

El análisis estático (Sección 3.2) incluye las pruebas analíticas que se pueden realizar sin ejecutar el software. Dado que el software no está ejecutándose, este se examina mediante una herramienta o una persona para establecer si procesará correctamente cuando se ejecute. Esta visión estática del software permite realizar un análisis detallado sin tener que crear los datos y las precondiciones necesarias para ejecutar el escenario.

Observe que las distintas formas de revisión que son relevantes para el Analista de Pruebas Técnicas se abordan en el Capítulo 5.

El análisis dinámico (Sección 3.3) requiere la ejecución efectiva del código y sirve para encontrar defectos de codificación que se detectan más fácilmente cuando el código se está ejecutando (por ejemplo, fugas de memoria). El análisis dinámico, al igual que el análisis estático, puede basarse en herramientas o en una monitorización individual del sistema en ejecución prestando atención a indicadores tales como el crecimiento rápido de la memoria.

3.2 Análisis Estático

El objetivo del análisis estático es detectar defectos reales o potenciales en el código y la arquitectura de sistema y para mejorar su mantenibilidad. El análisis estático, normalmente, está soportado por herramientas.

3.2.1 Análisis del Flujo de Control

El análisis del flujo de control es la técnica estática en virtud de la cual el flujo de control se analiza a través de un programa, bien a través del uso de un gráfico de flujo de control, bien a través de una herramienta. Existen muchas anomalías que pueden encontrarse en un sistema empleando esta técnica, incluyendo bucles mal diseñados (por ejemplo, con múltiples puntos de entrada), objetivos ambiguos de llamadas de función en algunos lenguajes (por ejemplo, Scheme), secuenciación incorrecta de las operaciones, etc.

Uno de los usos más comunes del análisis del flujo de control es determinar la complejidad ciclomática. El valor de la complejidad ciclomática es un entero positivo que representa el número de caminos independientes en un grafo que presenta un alto nivel de conexión con bucles e iteraciones ignorados en cuanto son atravesados una vez. Cada camino independiente, desde la entrada hasta la salida, representa un camino único a través del módulo. Cada camino único debería probarse.

El valor de la complejidad ciclomática, normalmente, se utiliza para conocer la complejidad general de un módulo. La teoría de Thomas McCabe [McCabe 76] consistía en que cuanto más complejo era el sistema, más difícil sería mantenerlo y más defectos contendría. Muchos estudios a lo largo de los años han observado esta correlación entre complejidad y el número de defectos contenidos. El NIST (National Institute of Standards and Technology) recomienda un valor máximo de la complejidad de 10. Cualquier módulo que se mida con una complejidad más alta puede tener que dividirse en varios módulos.

3.2.2 Análisis del Flujo de Datos

El análisis del flujo de datos cubre una serie de técnicas que reúnen información sobre el uso de variables en un sistema Se examina el ciclo de vida de las variables, (es decir, cuando estas están declaradas, definidas, leídas, evaluadas y destruidas), debido a que se pueden producir anomalías durante cualquiera de estas operaciones.

Una técnica común es la que se denomina notación definición-uso según la cual el ciclo de vida de cada variable se divide en tres acciones atómicas distintas:

- **d**: cuando la variable es **d**eclarada, definida o inicializada.
- **u**: cuando la variable es **u**tilizada o leída en un cómputo o en un predicado de decisión.
- **k**: cuando la variable es eliminada (**k**illed), destruida o se encuentra fuera de alcance.

Estas tres acciones atómicas se combinan en pares ("pares definición-uso") para ilustrar el flujo de datos. Por ejemplo, un "camino du" representa un fragmento de código en el que se define una variable de datos y, posteriormente, es utilizada.

Las posibles anomalías de datos incluyen llevar a cabo la acción correcta sobre una variable en un momento equivocado o llevar a cabo una acción incorrecta sobre datos en una variable. Entre otras anomalías están incluidas:

- Asignar un valor inválido a una variable.
- Fallo en la asignación de un valor a una variable antes de usarla.
- Adoptar un camino incorrecto debido a un valor incorrecto en un predicado de control.
- Intentar utilizar una variable después de que haya sido destruida.
- Referenciar una variable cuando está fuera del alcance.
- Declarar y destruir una variable sin utilizarla.
- Redefinir una variable antes de que se haya utilizada.
- Fallo en la destrucción de una variable asignada de forma dinámica (provocando una posible fuga de memoria)
- Modificar una variable, que da lugar en efectos secundarios inesperados (por ejemplo, efectos dominó[15] al cambiar una variable global sin tener en cuenta todos los usos de la variable).

El lenguaje de desarrollo que se esté utilizando podrá guiar las reglas empleadas en el análisis del flujo de datos. Los lenguajes de programación pueden permitir que el programador lleve a cabo ciertas acciones con variables que no son incorrectas, pero pueden hacer que el sistema se comporte de modo diferente de lo esperado por el programador en determinadas circunstancias. Por ejemplo, una variable podría definirse dos veces sin realmente ser usada cuando se sigue un camino determinado. A menudo, el análisis del flujo de datos etiquetará estos usos como "sospechosos". Si bien esto puede constituir un uso correcto de la capacidad de asignación de la variable, puede dar lugar a problemas de mantenibilidad del código en el futuro.

Las pruebas del flujo de datos "utilizan el grafo del flujo de control para explorar las cosas poco razonables que pueden suceder a los datos" [Beizer90] y, por lo tanto, encuentran defectos distintos de los detectados por las pruebas del flujo de control. Un Analista de Pruebas Técnicas debería incluir esta técnica a la hora de planificar las pruebas ya que muchos de estos defectos provocan fallos intermitentes que son difíciles de encontrar mientras se llevan a cabo pruebas dinámicas.

Sin embargo, el análisis del flujo de datos es una técnica estática, puede obviar algunos problemas que afectan a los datos en el sistema en tiempo de ejecución. Por ejemplo, la variable de datos estática puede contener un puntero a un arreglo creado de forma dinámica que ni siquiera existe hasta el tiempo de ejecución. El uso de multiprocesadores y multitareas preventivas puede crear condiciones de carrera[16] que no se detectarán con análisis del flujo de datos o del flujo de control.

3.2.3 Uso del Análisis Estático para Mejorar la Mantenibilidad

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

Un código mal escrito, no comentado y no estructurado tiende a ser más difícil de mantener. El código puede requerir un mayor esfuerzo por parte de los desarrolladores para localizar y analizar defectos

[15] "efecto dominó" es la traducción de "ripple effect".
[16] "condición de carrera" es la traducción de "race condition".

en el código y la modificación del código para corregir un defecto o añadir una nueva prestación puede resultar en la introducción de defectos adicionales.

El análisis estático se utiliza con un soporte de herramientas para mejorar la mantenibilidad del código, comprobando el cumplimiento de los estándares y directrices de codificación. Estos estándares y directrices describen las prácticas de codificación requeridas, tales como convenciones de nombres, comentarios, indentación y modularización del código. Observe que las herramientas de análisis estático generalmente indican advertencias en lugar de errores, a pesar de que el código pueda ser sintácticamente correcto.

Normalmente, los diseños modulares dan lugar a código más mantenible. Las herramientas de análisis estático dan soporte al desarrollo de código modular de la siguiente manera:

- Buscando código repetido. Estas secciones de código pueden ser candidatas a refactorizar[17] en módulos (aunque la sobrecarga en tiempo de ejecución[18] impuesta por las llamadas de módulo pueden ser un problema para sistemas en tiempo real).
- Generando métricas que son indicadores valiosos de la modularización de código. Entre ellas están incluidas las medidas de acoplamiento y cohesión. Un sistema con buena mantenibilidad es más probable que tenga una baja medida de acoplamiento (el grado en el que los módulos dependen entre sí durante la ejecución) y una alta medida de cohesión (el grado en el que un módulo es autocontenido y centrado en una tarea única).
- Indicando, en código orientado a objeto, donde los objetos derivados pueden tener demasiada o demasiada poca visibilidad en las clases padre (superclase).
- Resaltando las áreas del código o la arquitectura con un alto nivel de complejidad estructural, que normalmente se considera un indicador de mala mantenibilidad y una mayor posibilidad de contener defectos. Los niveles aceptables de complejidad ciclomática (remítase la Sección 3.2.1.) podrán especificarse en directrices para asegurar que el código se desarrolla de una forma modular sin olvidar la mantenibilidad y la prevención de defectos. El código con altos niveles de complejidad ciclomática pueden ser candidatos a la modularización.

El mantenimiento de un sitio web también puede estar soportado mediante el uso de herramientas de análisis estático. Aquí el objetivo es comprobar si la estructura de árbol[19] del sitio está equilibrada o si hay un desequilibrio que puede dar lugar a:

- Tareas de prueba más difíciles.
- Mayor carga de trabajo de mantenimiento.
- Navegación difícil para el usuario.

3.2.4 Grafos de Llamada

Los grafos de llamada son una representación estática de la complejidad de la comunicación. Son grafos dirigidos en los que los nodos representan unidades de programa y las aristas representan la comunicación entre las unidades.

Los grafos de llamada pueden utilizarse en las pruebas unitarias donde las distintas funciones o métodos se llaman entre sí, en las pruebas de integración y de sistema cuando módulos independientes se llaman entre sí, o en las pruebas de integración de sistemas cuando sistemas independientes se llaman entre sí.

Los grafos de llamada pueden utilizarse para los siguientes fines:

- Diseñar pruebas que llaman a un módulo o sistema específico.
- Establecer el número de localizaciones dentro del software a partir del cual se llama a un módulo o sistema.
- Evaluar la estructura del código y de la arquitectura de sistemas.

[17] "refactorizar" es la traducción de "refactoring".
[18] "sobrecarga en tiempo de ejecución" es la traducción de "run-time overhead" o "runtime overhead".
[19] "estructura de árbol" es la traducción de "tree-like structure".

- Facilitar sugerencias sobre el orden de integración (por pares e integración de proximidad. Se tratan más en detalle a continuación).

En el programa de estudio de Nivel Básico [ISTQB_FL_SYL], se abordan dos categorías de integración distintas: incremental (descendente, ascendente, etc.) y no incremental ("big bang"). Se prefieren los métodos incrementales debido a que introducen código en incrementos, lo que facilita el aislamiento de defectos ya que la cantidad de código implicada es limitada.

En este programa de estudio Avanzado, se introducen tres métodos no incrementales adicionales que utilizan grafo de llamada. Estos pueden ser preferibles a los métodos incrementales, que probablemente requieran construcciones adicionales para finalizar las pruebas y escribir código ajeno al envío[20] para dar soporte a las pruebas. Estos tres métodos son:

- Las pruebas de integración de a pares de elementos (no se debe confundir con la técnica de pruebas de caja negra "pruebas de a pares de elementos"), están orientadas a pares de componentes que trabajan juntos según se observa en el grafo de llamada para las pruebas de integración. Si bien este método reduce el número de compilaciones[21] sólo en una pequeña cantidad, reduce la cantidad de código necesario para el arnés de prueba.
- Las pruebas de integración de proximidad prueban todos los nodos que se conectan a un nodo dado como base para las pruebas de integración. Todos los nodos anteriores y posteriores de un nodo específico en el grafo de llamada conforman la base para la prueba.
- Enfoque del predicado de diseño de McCabe utiliza la teoría de la complejidad ciclomática aplicada a un grafo de llamada para módulos. Esto requiere la construcción de un grafo de llamada que muestre las distintas formas en que los módulos pueden llamarse entre sí, incluyendo:
  - Llamada incondicional: la llamada de un módulo a otro siempre ocurre.
  - Llamada condicional: la llamada de un módulo a otro ocurre a veces.
  - Llamada condicional mutuamente exclusiva: un módulo llamará a uno (y sólo a uno) de entre una serie de diferentes módulos.
  - Llamada iterativa: un módulo llama a otro módulo como mínimo una vez, pero puede llamarlo más de una vez.
  - Llamada condicional iterativa: un módulo puede llamar a otro de cero a muchas veces.

  Una vez creado el grafo de llamada, se calcula la complejidad de la integración, y se crean pruebas para cubrir el grafo.

Remítase a [Jorgensen07] para más información sobre el uso de gráficos de llamada y pruebas de integración de a pares de elementos.

3.3 Análisis Dinámico

3.3.1 Resumen

El análisis dinámico se utiliza para detectar fallos cuando los síntomas pueden no ser visibles de forma inmediata. Por ejemplo, la posibilidad de fugas de memoria puede detectarse mediante un análisis estático (encontrar código que asigna pero nunca libera memoria), pero una fuga de memoria se vuelve evidente con análisis dinámico.

Los fallos que no son reproducibles de forma inmediata pueden tener consecuencias significativas sobre el esfuerzo de prueba y para la capacidad de entregar o utilizar el software de forma productiva. Estos fallos pueden ser causados por fugas de memoria, uso incorrecto de punteros y otras corrupciones (por ejemplo, de la pila del sistema[22]) [Kaner02]. Dada la naturaleza de estos fallos, que pueden incluir el empeoramiento gradual del rendimiento del sistema o incluso el colapso del sistema, las estrategias de prueba deben tener en cuenta los riesgos asociados a dichos defectos y, cuando

[20] "código ajeno al envío" es la traducción de "non-shippable code".
[21] "compilación" es la traducción de "build" en este contexto.
[22] "pila del sistema" es la traducción de "system stqck".

proceda, realizar un análisis dinámico para reducirlos (normalmente mediante el uso de herramientas). Dado que estos fallos, a menudo, son los fallos más caros de encontrar y corregir, es recomendable llevar a cabo el análisis dinámico de forma temprana en el proyecto.

El análisis dinámico puede realizarse para lograr lo siguiente:

- Evitar que ocurran fallos detectando punteros sin referencia y pérdidas de memoria del sistema.
- Analizar los fallos del sistema que no se pueden reproducir fácilmente.
- Evaluar el comportamiento de la red.
- Mejorar el rendimiento del sistema facilitando información sobre el comportamiento del sistema en tiempo de ejecución[23].

El análisis dinámico puede realizarse en cualquier nivel de prueba y requiere habilidades técnicas y de sistema para poder realizar lo siguiente:

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

Durante las pruebas de sistema, se pueden utilizar herramientas de análisis dinámico, incluso si el Analista de Pruebas Técnicas contara con unas habilidades técnicas mínimas. Las herramientas empleadas, normalmente, crean registros exhaustivos que pueden ser analizados por aquellas personas que disponen las habilidades técnicas necesarias.

3.3.2 Detección de Fugas de Memoria

Una fuga de memoria se produce cuando las áreas de memoria (RAM) disponibles para un programa son asignadas por ese programa pero posteriormente no se liberan cuando dejan de ser necesarias. Este área de memoria se deja como si estuviera asignada y no está disponible para su reutilización. Cuando ocurre esta situación a menudo o en situaciones de baja memoria, el programa puede quedarse sin memoria útil. Históricamente, la manipulación de memoria era responsabilidad del programador. Cualquier área de memoria asignada dinámicamente tenía que ser liberadas por el programa que las había asignado, dentro del alcance correcto, para evitar una fuga de memoria. Muchos entornos de programación modernos incluyen una función de "recolección de basura[24]" automática o semiautomática mediante la cual se libera la memoria asignada sin la intervención directa del programador. Aislar fugas de memoria puede resultar muy difícil en aquellos casos en los que la memoria asignada existente se libera mediante la función de recolección de basura automática.

Las fugas de memoria causan problemas que se desarrollan con el tiempo y no siempre son obvias de forma inmediata. Este puede ser el caso si, por ejemplo, el software se ha instalado recientemente o se ha reiniciado el sistema, lo que ocurre a menudo durante las pruebas. Por estos motivos, los efectos negativos de las fugas de memoria pueden detectarse primero cuando el programa está en producción.

Los síntomas de una fuga de memoria son un empeoramiento constante del tiempo de respuesta del sistema que puede resultar finalmente en un fallo del sistema. Aunque dichos fallos pueden resolverse reiniciando el sistema, esto no siempre resulta práctico y a veces incluso posible.

Muchas herramientas de análisis dinámico identifican las áreas del código donde ocurren las fugas para que puedan ser corregidas. También pueden utilizarse monitores de memoria simples para obtener una impresión general de si la memoria disponible se reduce con el tiempo, aunque aún sería necesario un análisis de seguimiento para determinar la causa exacta de esta reducción.

[23] "tiempo de ejecución" es la traducción de "run-time".
[24] "recolección de basura" es la traducción de "garbage collection".

Hay otras fuentes de fugas que también deberían tenerse en cuenta. Los ejemplos se incluyen manipuladores de archivos, semáforos y fondos de conexión para recursos.

3.3.3 Detección de Punteros sin Referencia

Los punteros sin referencia en un programa son punteros que no deben utilizarse. Por ejemplo, un "puntero sin referencia" puede haber "perdido" el objeto o función a la que debería estar apuntando o no apunta al área de memoria que se pretendía (por ejemplo, apunta a un área fuera de los límites asignados a un arreglo). Cuando un programa utiliza "puntero sin referencias", se puede producir una amplia variedad de consecuencias:

- El programa puede comportarse como estaba previsto. Este puede ser el caso cuando el "puntero sin referencia" accede a memoria que actualmente no está siendo utilizada por el programa y se encuentra teóricamente "libre" y/o contiene un valor razonable.
- El programa se puede sufrir un fallo abrupto[25]. En este caso el "puntero sin referencia" puede haber causado que se utilice incorrectamente 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 objetos requeridos por el programa. En estas condiciones el programa puede seguir funcionando, a pesar de que se puede haber emitido un mensaje de error.
- El puntero puede haber corrompido los datos en la posición de memoria y los datos utilizados, posteriormente, pueden ser incorrectos.

Observar que cualquier cambio realizado en el uso de la memoria del programa (por ejemplo, una nueva compilación después de un cambio en el software) puede desencadenar cualquiera de las cuatro consecuencias enumeradas anteriormente. Esto es particularmente crítico cuando, inicialmente, el programa funciona como estaba previsto a pesar del uso de "puntero sin referencias", y después falla inesperadamente (quizás, incluso en producción) tras un cambio en el software. Es importante destacar que dichos fallos son síntomas de un defecto subyacente (es decir, el "puntero sin referencia") (remítase a [Kaner02], "Lección 74"). Alghunas herramientas pueden ayudar a identificar "punteros sin referencia" cuando están siendo utilizados por el programa, con independencia de su impacto en la ejecución del programa. Algunos sistemas operativos cuentan con funciones integradas para comprobar violaciones de acceso a memoria durante el tiempo de ejecución. Por ejemplo, el sistema operativo puede lanzar una excepción cuando una aplicación intenta acceder a una posición de memoria que se encuentra fuera del área de memoria permitida para dicha aplicación.

3.3.4 Análisis del Rendimiento

El análisis dinámico no solo es útil para detectar fallos. Con el análisis dinámico del rendimiento del programa, las herramientas ayudan a identificar cuellos de botella del rendimiento y generan una amplia gama de métricas de rendimiento que pueden ser utilizadas por el desarrollador para "poner a punto" el rendimiento del sistema. Por ejemplo, se puede aportar información sobre el número de veces que se llama a un módulo durante la ejecución. Los módulos que son llamados con frecuencia serían candidatos probables a una mejora del rendimiento.

Al combinar la información sobre el comportamiento dinámico del software con información obtenida a partir de los grafo de llamada durante el análisis estático (remítase a la Sección 3.2.4), el probador también puede identificar los módulos que podrían ser candidatos a pruebas detalladas y extensivas (por ejemplo, módulos que son llamados con frecuencia y que tienen muchas interfaces).

El análisis dinámico del rendimiento del programa, a menudo, se realiza durante las pruebas de sistema, si bien también puede realizarse durante las pruebas de un único subsistema en etapas más tempranas de las pruebas mediante arneses de pruebas.

[25] "fallo abrupto" es la traducción de "crash".

4. Características de Calidad para las Pruebas Técnicas - 405 minutos

Palabras clave

adaptabilidad, capacidad de ser analizado[26], capacidad para ser modificado[27], coexistencia, eficiencia, instalabilidad, pruebas de mantenibilidad, madurez, pruebas de aceptación operativa, perfil operativo, pruebas de rendimiento, pruebas de portabilidad, pruebas de capacidad de recuperación[28], modelo del crecimiento de la fiabilidad, pruebas de fiabilidad, capacidad de ser reemplazado[29], pruebas de utilización de recursos, robustez, pruebas de seguridad, estabilidad, capacidad de ser probado[30]

Objetivos de Aprendizaje para Características de Calidad para las Pruebas Técnicas

4.2 Problemas Generales de la Planificación

APT-4.2.1 (K4) Para un proyecto y sistema específicos sujeto a pruebas, analizar los requisitos no funcionales y redactar las secciones correspondientes del plan de prueba.

4.3 Pruebas de Seguridad

APT-4.3.1 (K3) Definir el enfoque y diseñar casos de prueba de alto nivel para las pruebas de seguridad.

4.4 Pruebas de Fiabilidad

APT-4.4.1 (K3) Definir el enfoque y diseñar casos de prueba de alto nivel para la característica de calidad de fiabilidad y sus correspondientes subcaracterísticas según la ISO 9126.

4.5 Pruebas de Rendimiento

APT-4.5.1 (K3) Definir el enfoque y diseñar perfiles operativos de alto nivel para las pruebas de rendimiento.

Objetivos de Aprendizaje Comunes

Los siguientes objetivos de aprendizaje se refieren al contenido cubierto en varias secciones de este capítulo.

APT-4.x.1 (K2) Comprender y explicar las razones de incluir pruebas de mantenibilidad, portabilidad y utilización de recursos en una estrategia de pruebas y/o enfoque de prueba.
APT-4.x.2 (K3) Dado un riesgo de producto específico, definir los tipos de pruebas no funcionales específicos más adecuados.
APT-4.x.3 (K2) Comprender y explicar las etapas del ciclo de vida de una aplicación en las que deberían aplicarse pruebas no funcionales.
APT-4.x.4 (K3) Dado un escenario, definir los tipos de defectos que esperaría encontrar utilizando tipos de pruebas no funcionales.

[26] "capacidad de ser analizado" antes "analizabilidad".
[27] "capacidad para ser modificado" antes "modificabilidad".
[28] "pruebas de capacidad de recuperación" antes "pruebas de recuperabilidad".
[29] "capacidad de ser reemplazado" antes "reemplazabilidad".
[30] "capacidad de ser probado" antes "testabilidad".

4.1 Introducción

En general, el Analista de Pruebas Técnicas centra las pruebas en "cómo" funciona el producto, más allá de los aspectos funcionales sobre "qué" hace el producto. Estas pruebas pueden realizarse en cualquier nivel de prueba. Por ejemplo, durante las pruebas de componente de sistemas en tiempo real y sistemas embebidos, es importante establecer referencias de rendimiento y probar la utilización de recursos. Durante las pruebas de sistema y las Pruebas de Aceptación Operativa (PAO), deben probarse aspectos de la fiabilidad, tales como la capacidad de recuperación. Las pruebas en este nivel tienen por objeto probar un sistema específico, es decir, combinaciones de hardware y software. El sistema específico sometido a pruebas puede incluir varios servidores, clientes, bases de datos, redes y otros recursos. Independientemente del nivel de prueba, las pruebas deberían realizarse en función de las prioridades de riesgo y los recursos disponibles.

Para describir las características de un producto, se utiliza como guía la descripción de las características de calidad que aporta la norma ISO 9126. También se pueden utilizar otras normas, tales como la serie ISO 25000 (que reemplaza a la ISO 9126). Las características de calidad según la norma ISO 9126 se dividen en características, cada una de las cuales pueden constar de subcaracterísticas. La siguiente tabla muestra estas características e indica qué características/subcaracterísticas están cubiertas por los programas de estudio del Analista de Pruebas y del Analista de Pruebas Técnicas:

Característica	Subcaracterística	Analista de Pruebas	Analista de Pruebas Técnicas
Funcionalidad	Exactitud, adecuación, interoperabilidad, cumplimiento	X	
	Seguridad.		X
Fiabilidad	Madurez (robustez), tolerancia a faltas, capacidad de recuperación, cumplimiento		X
Usabilidad	Capacidad de ser entendido[31], capacidad de ser aprendido[32], operabilidad, atractivo, cumplimiento	X	
Eficiencia	Rendimiento (comportamiento temporal), utilización de recursos, cumplimiento		X
Mantenibilidad	capacidad de ser analizado[33], capacidad para ser modificado[34], estabilidad, capacidad de ser probado[35], cumplimiento		X
Portabilidad	Adaptabilidad, instabilidad, coexistencia, capacidad de ser reemplazado[36], cumplimiento		X

Mientras que esta asignación de trabajo puede variar según las distintas organizaciones, esta es la seguida por los programas de estudio del ISTQB.

Para cada característica de calidad se muestra la subcaracterística de cumplimiento. En el caso de algunos entornos críticos para la seguridad o entornos regulados, cada característica de calidad puede tener que cumplir estándares y normas específicos. Dado que estos estándares pueden variar mucho en función del sector, no se abordarán en detalle en esta sección. Si el Analista de Pruebas Técnicas está trabajando en un entorno afectado por requisitos de cumplimiento, es importante

[31] "capacidad de ser entendido" antes "comprensibilidad".
[32] "capacidad de ser aprendido" antes "aprendibilidad".
[33] "capacidad de ser analizado" entes "analizabilidad".
[34] "capacidad para ser modificado" antes "modificabilidad".
[35] "capacidad de ser probado" antes "testabilidad".
[36] "capacidad de ser reemplazado" antes "reemplazabilidad".

conocer dichos requisitos y asegurar que tanto las pruebas como la documentación de las pruebas se ajustan a los requisitos de cumplimiento.

Para todas las características y subcaracterísticas de la calidad estudiadas en esta sección, se deben reconocer los riesgos típicos con el objetivo de poder formular y documentar una estrategia de prueba adecuada. Las pruebas de las características de calidad requieren una especial atención en lo que respecta a los tiempos del ciclo de vida, las herramientas necesarias, la disponibilidad de software y documentación y la pericia técnica. Sin la planificación de una estrategia con la tratar cada característica y sus necesidades únicas es posible que el probador no cuente con el tiempo adecuado para planificar, preparar y ejecutar pruebas integrado en el cronograma[37] [Bath08]. Algunas de estas pruebas, por ejemplo, las pruebas de rendimiento, requieren amplia planificación, equipos específicos, habilidades de prueba especializadas y, en la mayoría de los casos, una cantidad importante de tiempo. Las pruebas de las características y subcaracterísticas de calidad deben estar integradas en el calendario general de pruebas, asignándoles los recursos adecuados al esfuerzo. Cada una de estas áreas tiene necesidades específicas, actúa sobre problemas específicos y pueden ocurrir en distintos momentos durante el ciclo de vida del software, según lo establecido en las siguientes secciones.

Mientras que el Jefe de Pruebas es el encargado de recopilar y suministrar la información de métricas resumida sobre las características y subcaracterísticas de calidad, el Analista de Pruebas o el Analista de Pruebas Técnicas (según la tabla anterior) es el que recopila la información de cada métrica.

La medición de las características de calidad recopiladas por el Analista de Pruebas Técnicas durante las pruebas en preproducción puede servir de base para establecer un Acuerdo de Nivel de Servicio[38] (ANS) entre el proveedor y los implicados[39] (por ejemplo, clientes, operadores) del sistema software. En algunos casos, se puede seguir realizando pruebas después de que el software entre en producción, con frecuencia a cargo de equipos u organizaciones independientes. Normalmente este el caso de las pruebas de eficiencia y fiabilidad que pueden presentar distintos resultados en el entorno de producción y en el entorno de prueba.

4.2 Problemas Generales de la Planificación

No realizar un plan para pruebas no funcionales puede suponer un riesgo considerable para una aplicación. El Jefe de Pruebas podría solicitar al Analista de Pruebas Técnicas que identifique los principales riesgos para las características de calidad relevantes (remítase a la tabla en la Sección 4.1) y abordar cualquier problema de planificación asociado con las pruebas propuestas. Estos pueden utilizarse durante la creación del Plan Maestro de Prueba. Se deben tener en cuenta los siguientes factores generales al realizar estas tareas:

- Requisitos de los implicados.
- Adquisición de herramientas y formación necesarias.
- Requisitos del entorno de prueba.
- Consideraciones organizativas.
- Consideraciones sobre seguridad de los datos.

4.2.1 Requisitos de los Implicados

Los requisitos no funcionales, a menudo, están especificados de forma deficiente o ser inexistentes. En la fase de planificación, los Analistas de Pruebas Técnicas deben poder obtener niveles de expectativa para las características de calidad técnicas de por parte de los implicados y evaluar los riesgos que representan.

[37] "cronograma" es la traducción de "schedule" (en este contexto).
[38] "Acuerdo de Nivel de Servicio" es la traducción de "Service Level Agreement".
[39] "implicados" es la traducción de "stakeholder".

Un enfoque común es asumir que si el cliente está satisfecho con la versión existente del sistema, seguirá estando satisfecho con las nuevas versiones, siempre que se mantengan los niveles de calidad alcanzados. Esto permite utilizar la versión existente del sistema como referencia. Este puede ser un enfoque especialmente útil para algunas características de calidad no funcionales, como el rendimiento, en las que los implicados pueden tener dificultades para especificar sus requisitos.

Es recomendable obtener múltiples puntos de vista al captar requisitos no funcionales. Estos se deben educir[40] de los implicados tales como clientes, usuarios, personal de operaciones y de mantenimiento, de lo contrario es posible que se pasen por alto algunos requisitos.

4.2.2 Adquisición de Herramientas y Formación Necesarias

Las herramientas comerciales o simuladores son particularmente relevantes para pruebas de rendimiento y determinadas pruebas de seguridad. Los Analistas de Pruebas Técnicas deberían estimar los costes y las escalas de tiempo necesarios para adquirir, aprender e implementar las herramientas. Cuando deban utilizarse herramientas especializadas, la planificación debería responder a las curvas de aprendizaje para nuevas herramientas y/o el coste de contratación de especialistas en herramientas externos.

El desarrollo de un simulador complejo puede representar un proyecto de desarrollo por derecho propio y debería planificarse como tal. En particular, las pruebas y la documentación de la herramienta desarrollada deben estar representadas en el cronograma y en el plan de recursos. Se deberían planificar presupuesto y tiempo suficientes para actualizar y volver a probar el simulador a medida que el producto simulado cambia. La planificación de simuladores que se fueran a utilizar en aplicaciones de seguridad crítica debe tener en cuenta la prueba de aceptación y posible certificación del simulador por un organismo independiente.

4.2.3 Requisitos del Entorno de Prueba

Muchas pruebas técnicas (por ejemplo, pruebas de seguridad, pruebas de rendimiento) requieren un entorno de prueba parecido al de producción con el fin de ofrecer mediciones realistas. Dependiendo del tamaño y de la complejidad del sistema sometido 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 alto, se pueden tener en cuenta las siguientes alternativas:

- Utilizar el entorno de producción.
- Utilizar una versión reducida[41] del sistema. Debe prestarse especial atención a que los resultados de las pruebas obtenidos sean suficientemente representativos del sistema de producción[42].

La cronología de las ejecuciones de dichas pruebas deba planificarse cuidadosamente y es muy probable que dichas pruebas sólo puedan realizarse en momentos específicos (por ejemplo, en períodos de uso reducido).

4.2.4 Consideraciones Organizativas

Las pruebas técnicas pueden implicar medir el comportamiento de diversos componentes en un sistema completo (por ejemplo servidores, bases de datos, redes). Si estos componentes se distribuyen entre un número de emplazamientos y organizaciones diferentes, el esfuerzo requerido para planificar y coordinar las pruebas puede ser significativo. Por ejemplo, puede que determinados componentes software sólo estén disponibles para las pruebas de sistema en momentos específicos del día o del año, o puede que las organizaciones sólo ofrezcan apoyo para pruebas en un número limitado de días. La falta de confirmación de que los componentes del sistema y el personal (es decir,

[40] "educción" es la traducción de "elicitation".
[41] "reducida" es la traducción de "scaled-down".
[42] "sistema de producción" es la traducción "production system".

competencia técnica[43] "tomada prestada") de otras organizaciones estén disponibles "a demanda" para la realización de pruebas puede resultar en una disrupción grave para el calendario de las pruebas.

4.2.5 Consideraciones sobre la Seguridad de los Datos

Se deberían tener en cuenta medidas se seguridad específicas implementadas para un sistema, al menos, en la fase de planificación para asegurar que todas las actividades de prueba son posibles. Por ejemplo, el uso de encriptado de datos puede dificultar la creación de datos de prueba y la verificación de resultados.

Las políticas de protección de datos y las leyes pueden impedir la generación de cualquier dato de prueba basado en datos de producción. Hacer que los datos de prueba sean anónimos es una tarea nada trivial que debe planificarse como parte de la implementación de las pruebas.

4.3 Pruebas de Seguridad

4.3.1 Introducción

Las pruebas de seguridad se diferencian de otras formas de pruebas funcionales en dos aspectos fundamentales:

1. Las técnicas estándar para seleccionar datos de entrada de prueba pueden obviar algunas cuestiones importantes relacionadas con la seguridad.
2. Los síntomas de los defectos relacionados con la seguridad difieren mucho de aquellos detectados en otros tipos de pruebas funcionales.

Las pruebas de seguridad evalúan la vulnerabilidad de un sistema frente a amenazas intentando comprometer la política de seguridad del sistema. A continuación se presenta una lista de posibles amenazas que se deberían explorar durante las pruebas de seguridad:

- Copias no autorizadas de aplicaciones o información.
- Control de acceso no autorizado (por ejemplo, la capacidad para desempeñar tareas para las que el usuario no tiene derechos). La problemática de los derechos, acceso y privilegios de los usuarios son el foco de estas pruebas. Esta información deberá estar disponible en las especificaciones para el sistema.
- El software que muestra efectos secundarios imprevistos durante la ejecución de su función prevista. Por ejemplo, un reproductor multimedia[44] que reproduce audio de forma correcta pero realiza esta tarea copiando archivos en un almacenamiento temporal no encriptado presentando un efecto secundario del que se pueden aprovechar los piratas de software.
- Un código insertado en una página web que puede ser utilizado por usuarios subsiguientes (secuencias de comandos en sitios cruzados [45] o XSS). Este código puede ser malintencionado.
- El desbordamiento de la memoria intermedia[46] (saturación de la memoria intermedia[47]) se puede producir al introducir cadenas en un campo de entrada de una interfaz de usuario que sean más largas de lo que el código puede gestionar correctamente. Una vulnerabilidad de desbordamiento de la memoria intermedia supone una oportunidad de ejecutar instrucciones de código maliciosas.
- La denegación del servicio[48], evita la interacción del usuario con una aplicación (por ejemplo, sobrecargando un servidor web con solicitudes superfluas).

[43] "competencia técnica" es la traducción de "expertise".
[44] "reproductor multimedia" es la traducción de "media player".
[45] "secuencias de comandos en sitios cruzados" es la traducción de "cross-site scripting (XSS)".
[46] "desbordamiento de la memoria intermedia" es la traducción de "buffer overflow".
[47] "saturación de la memoria intermedia" es la traducción de "buffer overrun".
[48] "denegación del servicio" es la traducción de "service denial".

- La intercepción, imitación y/o modificación y consiguiente transmisión de comunicaciones (por ejemplo, transacciones de tarjeta de crédito) por un tercero de tal forma tal que un usuario no advierte la presencia de dicho tercero (ataque de intermediario[49]).
- Ruptura de los códigos de encriptado[50] que sirven para proteger información confidencial.
- Bombas lógicas (también conocidas como Huevos de Pascua), que pueden insertarse de forma maliciosa en un código y ser activadas sólo en determinadas circunstancias (por ejemplo, en una fecha concreta). Cuando estas bombas lógicas se activan, pueden desempeñar actos maliciosos, como la eliminación de archivos o el formateo de discos.

4.3.2 Planificación de las Pruebas de Seguridad

Los siguientes aspectos son, en general, especialmente relevantes durante la planificación de las pruebas de seguridad:

- Dado que se pueden introducir problemas de seguridad durante la arquitectura, diseño e implementación del sistema, las pruebas de seguridad se pueden programar para pruebas unitarias, de integración y de sistema. Dada la naturaleza cambiante de las amenazas de seguridad, las pruebas de seguridad también pueden programarse periódicamente una vez que el sistema haya entrado en producción.
- Las estrategias de prueba propuestas por el Analista de Pruebas Técnicas pueden incluir revisiones de código y análisis estático con herramientas de seguridad. Estas medidas pueden ser efectivas para encontrar problemas de seguridad en la arquitectura, documentos de diseño y código que fácilmente se podrían pasar por alto durante las pruebas dinámicas.
- El Analista de Pruebas Técnicas puede recibir el encargo de diseñar y ejecutar ciertos “ataques” de seguridad (ver más abajo) que requieren planificación y coordinación meticulosas con los implicados. Se pueden llevar a cabo otras pruebas de seguridad en colaboración con desarrolladores o Analistas de Pruebas (por ejemplo, pruebas de los derechos, acceso y privilegios de usuario). La planificación de las pruebas de seguridad debe prestar contemplar problemas organizativos como éstos.
- Un aspecto fundamental de la planificación de pruebas de seguridad es la obtención de aprobación. Para el Analista de Pruebas Técnicas, esto significa obtener autorización explícita del Jefe de Pruebas para llevar a cabo las pruebas de seguridad planificadas. Cualquier prueba adicional no prevista puede parecer un ataque real y la persona que la ejecuta podría correr el riesgo de ser objeto de acciones legales. Si no hay ningún documento por escrito que demuestre el propósito y la autorización de la prueba, la excusa “Estamos llevando a cabo una prueba de seguridad” puede ser difícil de explicar de forma convincente.
- Es conveniente observar que las mejoras introducidas en la seguridad de un sistema pueden afectar a su rendimiento. Una vez realizadas las mejoras de seguridad, se recomienda considerar la necesidad de llevar a cabo pruebas de rendimiento (ver Sección 4.5).

4.3.3 Especificación de Pruebas de Seguridad

Las pruebas de seguridad específicas pueden agruparse [Whittaker04] en función del origen del riesgo de seguridad:

- Relacionado con la interfaz de usuario - acceso no autorizado y entradas malintencionadas.
- Relacionado con el sistema de archivos - acceso a la información confidencial almacenada en archivos o repositorios.
- Relacionado con el sistema operativo - almacenamiento en memoria de información sensible como contraseñas en forma no encriptada que podría quedar expuesta si el sistema presenta un fallo abrupto por entradas malintencionadas.
- Relacionado con software externo - interacciones que pueden tener lugar entre componentes externos que utiliza el sistema. Estas pueden ocurrir a nivel de red (por ejemplo, paquetes

[49] “ataque de intermediario” es la traducción de “Man in the Middle attack”). Antes “ataque de hombre interpuesto”.
[50] “Ruptura de los códigos de encriptado” es la traducción de “breaking the encryption codes”.

incorrectos o mensajes transmitidos) o a nivel de componente software (por ejemplo, un fallo de un componente software del que depende el software).

El siguiente enfoque [Whittaker04] puede ser útil para desarrollar pruebas de seguridad:

- Recopilar información que puede ser útil para especificar pruebas, como nombres de trabajadores, direcciones físicas, información concerniente a redes internas, direcciones IP, identidad del software o hardware utilizado, y versión del sistema operativo.
- Llevar a cabo una búsqueda de vulnerabilidades [51] utilizando herramientas fácilmente accesibles. Esas herramientas no se utilizan para poner en peligro el sistema o los sistemas de forma directa, sino para identificar las vulnerabilidades que tienen o pueden resultar en, una brecha en la política de seguridad [52]. También pueden identificarse vulnerabilidades específicas mediante listas de comprobación como las facilitadas por el National Institute of Standards and Technology (NIST) [Web-2].
- Desarrollar "planes de ataque" (por ejemplo, un plan de acciones de prueba orientadas a comprometer una política de seguridad de un sistema específica), empleando la información recabada. En los planes de ataque, es necesario especificar diferentes entradas por medio de diferentes interfaces (por ejemplo, interfaz de usuario, sistema de archivos) para detectar las amenazas de seguridad más severas. Los diferentes "ataques" que se describen en [Whittaker04] son una fuente muy valiosa de técnicas desarrolladas específicamente para pruebas de seguridad.

Los problemas de seguridad también se pueden identificar mediante revisiones (ver Capítulo 5) y/o el uso de herramientas de análisis estático (ver Sección 3.2). Las herramientas de análisis estático contienen un amplio conjunto de reglas que son específicas de las amenazas de seguridad y respecto de las que se comprueba el código. Por ejemplo, la herramienta puede encontrar problemas de desbordamiento de la memoria intermedia, provocados por la falta de comprobación del tamaño de la memoria intermedia antes de la asignación de datos.

Las herramientas de análisis estático pueden ser utilizadas para el código web con el objetivo de de comprobar posibles exposiciones a vulnerabilidades de seguridad, tales como la inyección de código, la seguridad de cookies, la secuencia de comandos de sitios cruzados (XSS), la alteración de recursos[53] y la inyección de código SQL.

4.4 Pruebas de Fiabilidad

La clasificación de las características de calidad de los productos según la ISO 9126 define las siguientes subcaracterísticas de fiabilidad:

- Madurez.
- Tolerancia a faltas.
- Capacidad de recuperación.

4.4.1 Medición de la Madurez del Software

Un objetivo de las pruebas de fiabilidad es monitorizar una medida estadística de madurez del software en el tiempo y compararla con un objetivo de fiabilidad deseado que puede expresarse como un Acuerdo de Nivel de Servicio (ANS)[54]. Las medidas pueden tener la forma del Tiempo Medio Entre Fallos (TMEF), Tiempo Medio de Reparación (TMDR) o cualquier otra medida de intensidad de fallos (por ejemplo, el número de fallos semanales de una gravedad concreta). Estas se pueden utilizar como criterios de salida (por ejemplo, para la entrega a producción)

[51] "búsqueda de vulnerabilidades" es la traducción de "vulnerability scan".
[52] "brecha en la política de seguridad" es la traducción de "breach of security policy".
[53] "alteración de recursos" es la traducción de "resource tampering".
[54] "Acuerdo de Nivel de Servicio (ANS)" es la traducción de "Service Level Agreement (SLA)".

4.4.2 Pruebas de Tolerancia a Faltas

Además de las pruebas funcionales que evalúan la tolerancia del software a las faltas provocadas por valores de entrada inesperados (conocidas como pruebas negativas), es necesario realizar más pruebas para evaluar la tolerancia que presenta un sistema a aquellas faltas que ocurren de forma externa a la aplicación sujeta a pruebas. Normalmente estas faltas son comunicadas por el sistema operativo (por ejemplo, disco completo, proceso o servicio no disponible, archivo no encontrado, memoria no disponible). Existen herramientas específicas para llevar a cabo pruebas de tolerancia a faltas a nivel de sistema.

Observe que los términos “robustez” y “tolerancia al error” también se suelen utilizar para hablar de tolerancia de faltas (para detalles ver [ISTQB_GLOSSARY]).

4.4.3 Pruebas de la Capacidad de Recuperación

Hay formas adicionales para las pruebas de fiabilidad que evalúan la capacidad que tiene el sistema software para recuperarse de los fallos en el hardware o el software de una forma predeterminada, lo que posibilita la reanudación de las operaciones normales. Las pruebas de capacidad de recuperación incluyen las Pruebas de Recuperación ante Fallos[55] y de Pruebas de Copia de Seguridad[56] y Pruebas de y Pruebas de Restauración[57].

Las pruebas de recuperación ante fallos se realizan allí donde las consecuencias de un fallo software son tan negativas que se han implementado medidas específicas de hardware y/o software para asegurar la operación del sistema incluso en caso de fallo. Se pueden realizar pruebas de recuperación ante fallos, por ejemplo, cuando el riesgo de pérdidas económicas es extremo o cuando existen problemas de seguridad crítica. Cuando los fallos son consecuencia de eventos catastróficos, este tipo de prueba de capacidad de recuperación también es conocido como pruebas de “recuperación ante desastres”[58].

Las medidas preventivas típicas para los fallos de hardware podrían incluir el balanceo de carga[59] entre diferentes procesadores y la agrupación de servidores, procesadores o discos para que uno pueda relevar a otros en caso de fallo (sistemas redundantes). Una medida típica en el caso de software puede ser la implementación de más de una instancia del sistema software (por ejemplo, un sistema de control de vuelo de la aeronave) en lo que se conoce como sistemas redundantes disimilares[60]. Los sistemas redundantes consisten en una combinación de medidas software y hardware que pueden denominarse sistemas dobles, triples o cuádruples, dependiendo del número de instancias (dos, tres o cuatro respectivamente). El aspecto disimilar (dispar) del software se consigue cuando se proporcionan los mismos requisitos software a dos (o más) equipos de desarrollo independientes y no conectados, con el objetivo de que se suministren los mismos servicios por software diferentes. Esto protege a los sistemas redundantes disimilares, ya que una entrada defectuosa similar presenta menos probabilidades de tener el mismo resultado. Estas medidas implementadas para mejorar la capacidad de recuperación de un sistema también deberían tener una influencia directa en la fiabilidad y deberían tenerse en cuenta cuando se realicen pruebas de fiabilidad.

Las pruebas de recuperación ante fallos están explícitamente diseñadas para probar sistemas, simulando modos de fallo o provocando fallos reales en un entorno controlado. Tras el fallo, se prueba el mecanismo de recuperación ante fallos para asegurar que los datos no se han perdido ni corrompido y que se mantienen los niveles de servicio acordados (por ejemplo, disponibilidad de las funciones o tiempos de respuesta). Para más información sobre pruebas de recuperación ante fallos, consultar [Web-1].

[55] “pruebas de recuperación ante fallos” es la traducción de “failover tests”.
[56] “Pruebas de Copia de Seguridad” es la traducción de “backup tests”.
[57] “Pruebas de Restauración” es la traducción de “restore tests”.
[58] “pruebas de recuperación ante desastres” es la traducción de “disaster recovery testing”.
[59] “balanceo de carga” es la traducción de “load balancing”.
[60] “sistemas redundantes disimilares” es la traducción de “redundant dissimilar systems”.

Las pruebas de Copias de Seguridad" y Restauración tienen como objetivo el establecimiento de medidas procedimentales para minimizar las consecuencias de un fallo. Estas pruebas evalúan los procedimientos (normalmente documentados en un manual) para llevar a cabo diferentes tipos de copias de seguridad y para restaurar datos en caso de que se produjera una pérdida o corrupción de datos. Los casos de prueba están diseñados para asegurar que se cubren los cominos críticos para cada procedimiento. Se pueden realizar revisiones técnicas para "ensayar" estos escenarios y validar los manuales con respecto a los procedimientos existentes. Las Pruebas de Aceptación Operativa (PAO) practican los escenarios en entornos de producción o en entornos similares al de producción, para validar su uso vigente.

Entre las medidas para pruebas de Copias de Seguridad y Restauración se encuentran:

- Tiempo empleado para realizar diferentes tipos de copias de seguridad (por ejemplo, completa, incremental).
- Tiempo empleado en la restauración de datos.
- Niveles de copias de seguridad de datos garantizados (por ejemplo, recuperación de todos los datos de no más de 24 horas de antigüedad, recuperación de datos de transacción específicos de no más de una hora de antigüedad).

4.4.4 Planificación de Pruebas de Fiabilidad

En general, los siguientes aspectos se consideran relevantes al planificar las pruebas de fiabilidad:

- Se puede seguir monitorizando la fiabilidad a partir de que el software haya entrado en producción. La organización y el personal responsables de la operación del software deben ser consultados para recopilar requisitos de fiabilidad a efectos de la planificación de las pruebas.
- El Analista de Pruebas Técnicas puede seleccionar un modelo de crecimiento de fiabilidad que muestre los niveles de fiabilidad previstos a lo largo del tiempo. Un modelo de crecimiento de fiabilidad puede proporcionar información útil al Jefe de Pruebas al permitir comparar los niveles de fiabilidad esperados y alcanzados.
- Las pruebas de fiabilidad deberían llevarse a cabo en un entorno similar al de producción. El entorno utilizado debería permanecer lo más estable posible para que se puedan monitorizar las tendencias de fiabilidad en función del tiempo.
- Dado las que las pruebas de fiabilidad a menudo requieren el uso de todo el sistema, las pruebas de fiabilidad normalmente se realizan como parte de las pruebas de sistema. No obstante, pueden realizarse pruebas de fiabilidad a componentes individuales además de a conjuntos integrados de componentes. Las revisiones detalladas de arquitectura, diseño y código también se pueden utilizar para eliminar algunos de los riesgos de problemas asociados a la fiabilidad ocurridos en el sistema implementado.
- Normalmente, las pruebas de fiabilidad requieren tiempos de ejecución largos para que los resultados de estas pruebas sean estadísticamente significativos. Esto puede dificultar su programación[61] dentro de otras pruebas planificadas.

4.4.5 Especificación de Pruebas de Fiabilidad

Las pruebas de fiabilidad pueden adoptar la forma de conjunto repetido de pruebas predeterminadas. Estas pueden ser pruebas seleccionadas aleatoriamente de un fondo de casos de prueba[62] o casos de prueba generados por un modelo estadístico mediante métodos aleatorios o pseudoaleatorios. Las pruebas también pueden estar basadas en patrones de uso, también conocidos como "Perfiles Operativos" (ver Sección 4.5.1).

Algunas pruebas de fiabilidad pueden especificar que se ejecuten acciones intensivas en el uso de memoria[63] de forma repetida para poder detectar posibles fugas de memoria.

[61] "programar" es la traducción de "to schedule" en este contexto.
[62] "fondo de casos de prueba" es la traducción de "pool of test cases".
[63] "acciones intensivas en el uso de memoria" es la traducción de "memory-intensive actions".

4.5 Pruebas de Rendimiento

4.5.1 Introducción

La clasificación de características de la calidad de producto de la norma ISO 9126 incluye el rendimiento (comportamiento temporal) como una subcaracterística de la eficiencia. Las pruebas de rendimiento se centran en la capacidad de un componente o sistema para responder a las entradas de un usuario o sistema en un tiempo y en unas condiciones especificados.

Las mediciones del rendimiento varían en función de cuál sea el objetivo de la prueba. Para componentes software individuales, el rendimiento se puede medir de acuerdo a los ciclos de CPU, mientras que el rendimiento de sistemas basados en clientes se puede medir de acuerdo al tiempo necesario para responder a una solicitud particular de un usuario. En aquellos sistemas cuya arquitectura consiste en diferentes componentes (por ejemplo, clientes, servidores, bases de datos), se toman mediciones del rendimiento para las transacciones que tienen lugar entre componentes individuales para que se puedan identificar los “cuellos de botella” en el rendimiento.

4.5.2 Tipos de Pruebas de Rendimiento

4.5.2.1 Pruebas de Carga

Las pruebas de carga miden la capacidad que tiene un sistema para tratar niveles crecientes de cargas realistas anticipados, resultado de las solicitudes de transacciones generadas por un número de usuarios o procesos concurrentes. Se pueden medir y analizar los tiempos de respuesta medios[64] para los usuarios en diferentes escenarios de uso típico (perfiles operativos). Ver también [Splaine01].

4.5.2.2 Pruebas de Estrés

Las pruebas de estrés se centran en la capacidad de un sistema o componente de tratar picos de carga en o más allá de los límites de las cargas de trabajo anticipados o especificados, o con disponibilidad reducida de recursos tales como la capacidad de máquina accesible y el ancho de banda disponible. Los niveles de rendimiento se deberían degradar lenta y previsiblemente sin fallo a medida que se incrementan los niveles de estrés. En particular, se debería probar la integridad funcional del sistema mientras que el sistema se encuentra bajo estrés con el fin de encontrar posibles faltas en el procesamiento funcional o inconsistencias en los datos.

Un posible objetivo de las pruebas de estrés es descubrir los límites en los cuales el sistema falla realmente de forma que se pueda determinar el “eslabón más débil de la cadena”. Las pruebas de estrés permiten añadir capacidad adicional al sistema cuando sea oportuno (por ejemplo, memoria, capacidad de CPU, almacenamiento de la base de datos).

4.5.2.3 Pruebas de Escalabilidad

Las pruebas de escalabilidad miden la capacidad que tiene un sistema para satisfacer las necesidades de eficiencia futuras, que pueden exceder a las requeridas en el momento actual. El objetivo de las pruebas es determinar la capacidad del sistema para crecer (por ejemplo, con más usuarios, mayores cantidades de datos almacenados) sin exceder los requisitos de rendimiento especificados actualmente o sin fallar. Una vez que se conocen los límites de escalabilidad, se pueden establecer y monitorizar los valores umbral en producción para proporcionar una advertencia de problemas inminentes. Asimismo, el entorno de producción puede ajustarse con cantidades de hardware adecuadas.

[64] “tiempos de respuesta medios” es la traducción de “average response times”.

4.5.3 Planificación de Pruebas de Rendimiento

Además de los problemas de planificación general previstos en la Sección 4.2, los siguientes factores pueden afectar a la planificación de las pruebas de rendimiento:

- Las pruebas de rendimiento pueden requerir que todo el sistema esté implementado antes de poder realizar las pruebas efectivas, dependiendo del entorno de pruebas utilizado y del software probado, (ver Sección 4.2.3). En este caso, las pruebas de rendimiento normalmente están programadas durante las pruebas de sistema. Otras pruebas de rendimiento que pueden llevarse a cabo de manera efectiva a nivel de componentes se pueden programar durante las pruebas unitarias.
- En general, es deseable llevar a cabo pruebas de rendimiento iniciales tan pronto como posible, incluso si aún no se dispone de un entorno similar al de producción. Estas pruebas tempranas pueden detectar problemas de rendimiento (por ejemplo, cuellos de botella) y reducir el riesgo de proyecto evitando tener que realizar correcciones que requieren mucho tiempo en etapas posteriores del desarrollo del software o en producción.
- Las revisiones de código, en particular las referidas a la interacción con base de datos, la interacción entre componentes y el tratamiento de errores, pueden identificar problemas de rendimiento (en particular con respecto a la lógica “wait and retry”[65] y a consultas ineficientes) y se deberían programar temprano en el ciclo de vida del software.
- El hardware, software y ancho de banda de red necesarios para ejecutar las pruebas de rendimiento deberían estar planificados y presupuestados. Las necesidades dependen principalmente de la carga a generar, que puede estar basada en el número de usuarios virtuales a simular y de la cantidad de tráfico de red que pueden generar. Si no se tiene esto en cuenta, podrían tomarse mediciones de rendimiento no representativas. Por ejemplo, verificar los requisitos de escalabilidad de un sitio web muy visitado puede requerir la simulación de cientos de miles de usuarios virtuales.
- Generar la carga requerida para las pruebas de rendimiento puede tener un impacto importante en los costes de adquisición de hardware y herramientas. Esto se debe tener en cuenta al planificar las pruebas de rendimiento para garantizar que se dispone de fondos adecuados.
- Los costes de generación de carga para las pruebas de rendimiento puede minimizarse alquilando la infraestructura de pruebas necesaria. Esto puede implicar, por ejemplo, el alquiler de licencias prepago[66] para herramientas de rendimiento o recurriendo a los servicios de un proveedor tercero para satisfacer las necesidades de hardware (por ejemplo, servicios de nube). Si se adopta este enfoque, el tiempo disponible para llevar a cabo las pruebas de rendimiento puede quedar limitado y, por lo tanto, debe planificarse detenidamente.
- Durante la fase de planificación se debe prestar una atención especial para asegurar que la herramienta de rendimiento que se va a utilizar cuenta con la compatibilidad necesaria con los protocolos de comunicación que utiliza el sistema sujeto a pruebas.
- Los defectos asociados al rendimiento suelen tener un impacto importante en el sistema que se está probando. Cuando los requisitos de rendimiento son imperativos, en ocasiones puede resultar útil llevar a cabo pruebas de rendimiento en componentes críticos (mediante controladores y stubs) en lugar de esperar a las pruebas de sistema.

4.5.4 Especificación de Pruebas de Rendimiento

La especificación de pruebas para los distintos tipos de pruebas de rendimiento, tales como carga y estrés, se basa en la definición de perfiles operativos. Éstos representan los distintos tipos de comportamiento del usuario cuando interactúa con una aplicación. Puede haber varios perfiles operativos para una aplicación dada.

[65] ““wait and retry” se traduciría como “esperar y volver a intentar”. Se ha considerado conveniente no traducir este texto.
[66] “licencia prepaga” es la traducción de “top-up license”.

El número de usuarios por perfil operativo se puede obtener utilizando herramientas de monitorización (cuando la aplicación real o una comparable se encuentran disponibles) o pronosticando el uso. Estos pronósticos pueden estar basados en algoritmos o ser facilitados por la empresa, y son especialmente importantes para especificar el perfil o perfiles operativos que se emplearán en las pruebas de escalabilidad.

Los perfiles operativos conforman la base para definir el número y los tipos de casos de pruebas que se van a utilizar durante las pruebas de rendimiento. Estas pruebas, a menudo, están controladas por herramientas de prueba que crean la cantidad de usuarios "virtuales" o simulados necesaria para representar el perfil sujeto a prueba (ver Sección 6.3.2).

4.6 Utilización de Recursos

La clasificación de las características de la calidad de producto según la norma ISO 9126 incluye la utilización de recursos como una subcaracterística de la eficiencia. Las pruebas relativas a la utilización de recursos evalúan el uso de los recursos de sistema (por ejemplo, uso de memoria, capacidad del disco, ancho de banda de la red, conexiones) con respecto a una referencia predefinida. Los resultados se comparan con situaciones de carga normal y de estrés, como pueden ser altos niveles de transacciones y volúmenes de datos para determinar si se está produciendo un crecimiento del uso fuera de lo normal.

Por ejemplo, para sistemas embebidos en tiempo real, el uso de la memoria (también conocido como "huella de memoria"[67]) desempeña un papel importante en las pruebas de rendimiento. Si la huella de memoria excede la medida permitida, el sistema puede no disponer de la memoria suficiente necesaria para llevar a cabo sus tareas dentro de los periodos de tiempo especificados. Esto puede ralentizar el sistema o incluso provocar un fallo abrupto[68] del sistema.

También se puede utilizar análisis dinámico en la tarea de investigar la utilización de recursos (ver Sección 3.3.4) y detectar cuellos de botella para el rendimiento.

4.7 Pruebas de Mantenibilidad

A menudo el software pasa una mayor proporción de su tiempo de vida siendo objeto de mantenimiento que siendo desarrollado. Las pruebas de mantenimiento se llevan a cabo para probar los cambios en un sistema en operación o el impacto de un cambio en el entorno para un sistema en operación. Para asegurar que la tarea de mantenimiento es tan eficiente como sea posible, se realizan pruebas de mantenibilidad para medir la facilidad con la que el código se puede ser analizado, modificado y probado.

Algunos ejemplos de objetivos de mantenibilidad típicos de implicados afectados (por ejemplo, el propietario del software o el operador del software) son:

- Minimizar el coste de propiedad[69] u de operación[70] del software.
- Minimizar el tiempo de inactividad[71] necesario para las tareas de mantenimiento del software.

Las pruebas de mantenibilidad deberían estar incluidas en una estrategia de pruebas y/o enfoque de pruebas donde uno o más de los siguientes factores correspondan:

- Es probable que se realicen cambios en el software tras su puesta en producción (por ejemplo, para corregir defectos o introducir actualizaciones planificadas).
- Los implicados afectados consideran que los beneficios de alcanzar los objetivos de mantenibilidad (ver más arriba) en el ciclo de vida del software compensan los costes de llevar a cabo las pruebas de mantenibilidad y realizar los cambios requeridos.

[67] "huella de memoria" es la traducción de "memory footprint".
[68] "fallo abrupto" es la traducción de "crash".
[69] "coste de propiedad" es la traducción de "cost of owning".
[70] "coste de operación es la traducción de "cost of operating".
[71] "tiempo de inactividad" es la traducción de down time".

- El riesgo de una baja mantenibilidad del software (por ejemplo, tiempos de respuesta prolongados a defectos informados por usuarios y/o clientes) justifica la realización de pruebas de mantenibilidad.

Algunas técnicas adecuadas para pruebas de mantenibilidad son el análisis estático y las revisiones según se lo expuesto en las Secciones 3.2 y 5.2. Se deberían iniciar las pruebas de mantenibilidad en cuanto estuvieran disponibles los documentos de diseño y deberían continuar durante el esfuerzo de implementación del código. Dado que la mantenibilidad se desarrolla en el código y a la documentación de cada componente de código individual, la mantenibilidad puede evaluarse de forma temprana en el ciclo de vida sin tener que esperar a que el sistema esté completo y en ejecución.

Las pruebas dinámicas de mantenibilidad se centran en los procedimientos documentados desarrollados para el mantenimiento de una aplicación concreta (por ejemplo, para realizar mejoras del software). Se utilizan selecciones de escenarios de mantenimiento como casos de prueba para asegurar que se alcanzan los niveles de servicio requeridos con los procedimientos documentados. Este tipo de prueba es especialmente relevante cuando la infraestructura subyacente es compleja y los procedimientos de apoyo pueden involucrar a varios departamentos/organizaciones. Este tipo de prueba puede tener lugar como parte de las Pruebas de Aceptación Operativa (PAO). [Web-1]

4.7.1 Analizabilidad, Capacidad de Ser Modificado[72], Estabilidad y Capacidad de Ser Probado[73]

La mantenibilidad de un sistema puede medirse en términos del esfuerzo necesario para diagnosticar los problemas identificados dentro de un sistema (analizabilidad), implementar los cambios de código (capacidad de ser modificado) y probar el sistema modificado (capacidad de ser probado). La estabilidad está asociada, de forma específica, con la respuesta del sistema al cambio. Los sistemas con una baja estabilidad presentan muchos problemas derivados (también conocido como el “efecto dominó”) cuando se realiza un cambio. [ISO9126] [Web-1].

El esfuerzo necesario para llevar a cabo las tareas de mantenimiento depende de una serie de factores tales como la metodología de diseño del software (por ejemplo, la orientación a objetos) y los estándares de codificación empleados.

Observe que en este contexto el término “estabilidad” no debe confundirse con los términos “robustez” y “tolerancia a faltas” que se abordan en la Sección 4.4.2.

4.8 Pruebas de Portabilidad

Las pruebas de portabilidad, en general, miden la facilidad con la que un software puede ser transferido a un entorno previsto, ya sea de forma inicial o desde otro entorno existente. Entre las pruebas de portabilidad existentes están las pruebas de instalabilidad, coexistencia/compatibilidad, adaptabilidad y capacidad de ser reemplazado. Las pruebas de portabilidad pueden comenzar por los componentes individuales (por ejemplo, capacidad de un componente específico de ser reemplazado tal como cambiar de un sistema de gestión de bases de datos a otro) y ampliar el alcance a medida que haya más código disponible. Es posible que la instalabilidad no pueda probarse hasta que todos los componentes del producto se encuentren funcionalmente operativos. La portabilidad debe diseñarse y desarrollarse en producto, por lo que debe considerarse pronto en las fases de diseño y arquitectura. Las revisiones de arquitectura y diseño pueden ser especialmente productivas para identificar posibles requisitos y problemas de portabilidad (por ejemplo, dependencia con respecto a un sistema operativo en particular).

4.8.1 Pruebas de Instalabilidad

[72] “capacidad de ser modificado” también “modificabilidad”.
[73] “capacidad de ser probado” también “testeabilidad”

Las pruebas de instalabilidad se ejecutan en el software y procedimientos escritos que se utilizan para instalar el software en su entorno objetivo. Ello puede incluir, por ejemplo, el software desarrollado para instalar un sistema operativo en un procesador, o un "asistente" de instalación[74] utilizado instalar un producto en un ordenador cliente.

Entre los objetivos típicos de las pruebas de instalabilidad se encuentran:

- Validar que el software puede ser instalado con éxito siguiendo las instrucciones de un manual de instalación (incluyendo la ejecución de cualquier "guion" de instalación), o utilizando un asistente de instalación. Esto implica la ejecución de opciones de instalación para diferentes configuraciones hardware/software y para diferentes grados de instalación (por ejemplo, inicial o actualización).
- Probar si los fallos que se producen durante la instalación (por ejemplo, fallos en la carga de librerías de enlace dinámico[75] concretas) son tratados correctamente por el software de instalación sin dejar el sistema en un estado indefinido (por ejemplo, software instalado parcialmente o configuraciones incorrectas del sistema).
- Probar si se puede completar instalación/desinstalación parcial.
- Probar si el asistente de instalación puede identificar con éxito plataformas hardware o configuraciones del sistema operativo no válidas.
- Medir si el proceso de instalación se puede completar en un número especificado de minutos o en menos de un número especificado de pasos.
- Validar que el software puede ser instalado en una versión anterior o desinstalado con éxito.

Las pruebas de funcionalidad se realizan, normalmente, con posterioridad a las pruebas de instalación para detectar cualquier defecto que pudiera haber sido introducido por la instalación (por ejemplo, configuraciones incorrectas, funciones no disponibles). Las pruebas de usabilidad se llevan a cabo, normalmente, en paralelo con las pruebas de instalabilidad (por ejemplo, para comprobar que los usuarios disponen de unas instrucciones comprensibles y obtienen realimentación[76] o mensajes de error durante la instalación).

4.8.2 Pruebas de Coexistencia/Compatibilidad

Se dice que sistemas basados en ordenador, no relacionados entre sí, son compatibles cuando pueden funcionar en un mismo entorno (por ejemplo, en el mismo hardware) sin afectar el comportamiento de los otros (por ejemplo, conflictos de recursos). La compatibilidad se debería llevar a cabo cuando se vaya a introducir software nuevo o actualizado en entornos que ya contienen aplicaciones instaladas.

Los problemas de compatibilidad pueden surgir cuando se prueba la aplicación en un entorno donde esta aplicación es la única aplicación instalada (donde los problemas de incompatibilidad no son detectables) y, posteriormente, desplegada en otro entorno (por ejemplo, producción) en el que también otras aplicaciones se encuentran en ejecución.

Entre los objetivos típicos de las pruebas de compatibilidad se encuentran:

- Evaluación del posible impacto adverso sobre la funcionalidad al cargar aplicaciones dentro de un mismo entorno (conflictos de utilización de recursos cuando un servidor ejecuta múltiples aplicaciones).
- Evaluación del impacto en cualquier aplicación como resultado del despliegue de parches y actualizaciones del sistema operativo.

Los problemas de compatibilidad deberían analizarse durante la planificación del entorno de producción objetivo, pero las pruebas efectivas normalmente se llevan a cabo después de haber completado con éxito las pruebas de sistema y de aceptación de usuario.

[74] "asistente de instalación" es la traducción de "instalation wizard".
[75] "librería de enlace dinámico" es la traducción de "dynamic-link library". También conocida por su acrónimo en inglés "DLL".
[76] "realimentación" es la traducción de "feedback".

4.8.3 Pruebas de Adaptabilidad

Las pruebas de adaptabilidad comprueban si una aplicación dada funciona correctamente en todos los entornos objetivo previstos (entornos hardware, software, “middleware”, sistemas operativos, etc.). Un sistema adaptable es por tanto un sistema abierto capaz de ajustar[77] su comportamiento de acuerdo a los cambios en su entorno o en partes del propio sistema. La especificación de pruebas de adaptabilidad requiere que las combinaciones de los entornos previstos sean identificadas, configuradas y que se encuentren disponibles para equipo de prueba. A continuación, estos entornos son probados utilizando una selección de casos de prueba funcionales que practican los distintos componentes presentes en el entorno.

La adaptabilidad se puede referir a la capacidad delel software para ser transferido a distintos entornos especificados llevando a cabo un procedimiento predefinido. Las pruebas pueden evaluar este procedimiento.

Las pruebas de adaptabilidad pueden llevarse a cabo en conjunción con las pruebas de instalabilidad y, normalmente, van seguidas de pruebas funcionales, que sirven para detectar cualquier defecto que pueda haber sido introducido en la adaptación del software a un entorno diferente.

4.8.4 Pruebas de Capacidad de Ser Reemplazado

Las pruebas de la capacidad de ser reemplazado se concentran en la capacidad que tiene un componente software dentro de un sistema para ser sustituido por otros. Esto puede ser especialmente importante para aquellos sistemas que utilizan software comercial de distribución masiva (COTS[78]) para componentes específicos del sistema.

Las pruebas de capacidad de ser reemplazado se pueden realizar en paralelo con las pruebas de integración funcionales, cuando haya más de un componente alternativo disponible para su integración en el sistema completo. La capacidad de ser reemplazado se puede evaluar con una revisión técnica o una inspección en los niveles de arquitectura y diseño, donde se hace énfasis en la definición clara de las interfaces con componentes reemplazables potenciales.

[77] “ajustar” es la traducción de “to fit”.
[78] “COT” es el acrónimo del termino en inglés “Commercial Off-The-Shelf”.

5. Revisiones - 165 minutos

Palabras clave
antipatrón

Objetivos de Aprendizaje de las Revisiones

5.1 Introducción

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

5.2 Uso de Listas para Comprobación en las Revisiones

APT 5.2.1 (K4) Analizar un diseño de arquitectura e identificar problemas de acuerdo a una lista de comprobación aportada por el programa de estudio.

APT 5.2.2 (K4) Analizar un segmento de código o pseudocódigo e identificar problemas de acuerdo a una lista de comprobación aportada por el programa de estudio.

5.1 Introducción

Los Analistas de Pruebas Técnicas debe ser un participante activo en el proceso de revisión aportando sus puntos de vista particulares. Ellos deberían contar con capacitación formal en revisiones para comprender mejor sus roles respectivos en cualquier proceso de revisión técnica. Todos los participantes del proceso de revisión deben estar comprometidos con los beneficios de una revisión técnica llevada a cabo de forma correcta. Para obtener una descripción detallada de las revisiones técnicas, con varios ejemplos de listas de comprobación de revisiones, ver [Wiegers02]. Los Analistas de Pruebas Técnicas normalmente participan en las revisiones e inspecciones técnicas donde aportan un punto de vista operativo (comportamiento) que los desarrolladores pueden pasar por alto. Además, los Analistas de Pruebas Técnicas desempeñan un papel importante en la definición, aplicación y mantenimiento de las listas de comprobación de las revisiones y de la información sobre la severidad de los defectos.

Independientemente del tipo de revisión que se realice, el Analista de Pruebas debe poder disponer de un tiempo adecuado para su preparación. Esto incluye tiempo para revisar el producto de trabajo, tiempo para comprobar documentos de referencias cruzadas para verificar la consistencia, y tiempo para establecer que podría faltar del producto de trabajo. Sin un tiempo de preparación adecuado, la revisión puede resultar un ejercicio de edición más que una verdadera revisión. Una buena revisión incluye comprender qué está escrito, determinar qué falta y verificar que el producto descrito es coherente con otros productos que ya han sido desarrollados o están en desarrollo. Por ejemplo, al revisar un plan de pruebas de nivel de integración, el Analista de Pruebas Técnicas también debe tener en cuenta los elementos que se están integrando. ¿Están listos para ser integrados? ¿Hay dependencias que deban documentarse? ¿Hay datos disponibles para probar los puntos de integración? Una revisión no es una parte aislada del producto de trabajo que se está revisando. También debe tener en cuenta la interacción de dicho elemento con los demás elementos del sistema.

No es raro que los autores de un producto objeto de una revisión se sientan criticados. El Analista de Pruebas Técnicas debería asegurarse de abordar todos los comentarios de revisión desde una perspectiva de colaboración con el autor para crear el mejor producto posible. Utilizando este enfoque, los comentarios se redactarán de una manera constructiva y estarán orientados hacia el producto de trabajo y no hacia al autor. Por ejemplo, si una afirmación resulta ambigua, es mejor decir “No entiendo qué es lo que tengo que probar para comprobar que este requisito se ha implementado correctamente. ¿Podría ayudarme a comprenderlo?” en lugar de “Este requisito es ambiguo y nadie será capaz de entenderlo.”

La tarea de un Analista de Pruebas Técnicas en una revisión es garantizar que la información facilitada en el producto de trabajo será suficiente para soportar el esfuerzo de prueba. Si la información no existe o no está clara, entonces es probable que haya un defecto que el autor deba corregir. Al mantener un enfoque positivo en lugar de un enfoque crítico, los comentarios serán mejor recibidos y la reunión será más productiva.

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 tienen que verificar puntos específicos durante la revisión. Las listas de comprobación también permiten despersonalizar la revisión, por ejemplo, “Utilizamos la misma lista de comprobación para todas las revisiones, no estamos centrándonos exclusivamente en su producto de trabajo.” Las listas de comprobación pueden ser genéricas y utilizarse para todas las revisiones o pueden centrarse en características de calidad o áreas específicas. Así por ejemplo, una lista de comprobación genérica puede comprobar el uso correcto de los términos “debe” y “debería”, comprobar el formato adecuado y otros elementos de conformidad[79]. Una lista de comprobación específica puede concentrarse en problemas de seguridad o rendimiento.

[79] “elementos de conformidad” es la traducción de “conformance item”.

Las listas de comprobación más útiles son las desarrolladas gradualmente por una organización individual, ya que en ellas se refleja:

- La naturaleza del producto.
- El entorno de desarrollo local.
  - El personal.
  - Las herramientas.
  - Las prioridades.
- La historia de éxitos y defectos anteriores.
- Problemas específicos (por ejemplo, rendimiento, seguridad).

Las listas de comprobación deberían personalizarse en función de la organización y quizás también en función del proyecto específico. Las listas de comprobación incluidas en este capítulo sólo pretenden ser utilizados como ejemplos.

Algunas organizaciones amplían la noción habitual de lista de comprobación de software para incluir "antipatrones" que se refieren a errores comunes, malas técnicas y otras prácticas inefectivas. El término procede del concepto popular de "patrones de diseño" que son soluciones reutilizables en problemas comunes que han demostrado ser efectivos en situaciones prácticas [Gamma94]. Un antipatrón es, por lo tanto, un error comúnmente cometido, a menudo implementado como un atajo conveniente.

Es importante recordar que si un requisito no cuenta con la capacidad de ser probado, es decir, no está definido de tal forma que el Analista de Pruebas Técnicas puede determinar cómo probarlo, entonces hay un defecto. Por ejemplo, un requisito que establece "El software debería ser rápido" no puede probarse. ¿Cómo puede el Analista de Pruebas Técnicas determinar si el software es rápido? Si, en lugar de eso, el requisito estableciera "El software debe ofrecer un tiempo de respuesta máximo de tres segundos en condiciones de carga específicas", entonces la capacidad de ser probado de este requisito sería sustancialmente mejor si definimos las "condiciones de carga específicas" (por ejemplo, número de usuarios concurrentes, las actividades realizadas por los usuarios). También constituye un requisito determinante[80] porque este mismo requisito podría fácilmente dar lugar a muchos casos de prueba individuales en una aplicación no trivial. La trazabilidad entre este requisito y los casos de prueba también resulta crítica ya que, si el requisito cambiara, se tendrían que revisar y actualizar todos los casos de prueba según sea necesario.

5.2.1 Revisiones de la Arquitectura

La arquitectura de software consta de la organización fundamental de un sistema, materializado en sus componentes, sus relaciones entre sí y el entorno, y los principios que rigen su diseño y evolución. [ANSI/IEEE Std 1471-2000], [Bass03].

Las listas de comprobación empleadas para las revisiones de la arquitectura podrían, por ejemplo, incluir la verificación de la correcta implementación de los siguientes elementos, que se citan en [Web-3]:

- "Fondo de conexiones para compartir[81] - reducción del tiempo de ejecución adicional asociado al establecimiento de conexiones de bases de datos estableciendo un fondo de conexiones compartidas.
- Balanceo de carga - distribuir la carga de forma uniforme entre un conjunto de recursos.
- Procesamiento distribuido.
- Uso de almacenamiento caché[82] - utilizar una copia local de datos para reducir el tiempo de acceso.
- Instanciación tardía[83].

[80] "requisito determinante" es la traducción de "overarching requirement".
[81] "fondo de conexiones para compartir" es la traducción de "connection pooling".
[82] "uso de almacenamiento caché" es la traducción de "caching".
[83] "instanciación tardía" es la traducción de "lazy instantiation".

- Concurrencia de transacciones.
- Aislamiento de procesos entre el Procesamiento Transaccional en Línea[84] (OLTP[85]) y el Procesamiento Analítico en Línea[86] (OLAP[87]).
- Replicación de datos."

Se pueden encontrar más detalles en [Web-4] (no relevantes para el examen de certificación), que se refiere a un artículo que hace un estudio de 117 listas de comprobación procedentes de 24 fuentes. Se estudian las distintas categorías de listas de comprobación y se aportan ejemplos de buenos elementos de listas de comprobación así como los que deberían evitarse.

5.2.2 Revisiones de Código

Las listas de comprobación para las revisiones de código son necesariamente muy detalladas y, al igual que sucede con las listas de comprobación para las revisiones de la arquitectura, son más útiles cuando son específicas del lenguaje, el proyecto y la empresa. La inclusión de antipatrones a nivel de código es útil, en particular para los desarrolladores de software con menos experiencia.

Las listas de comprobación empleadas para las revisiones de código podrían incluir los siguientes seis elementos: (basado en [Web-5]).

1. Estructura
   - ¿El código implementa el diseño completa y correctamente?
   - ¿El código cumple con algunos de los estándares de codificación pertinentes?
   - ¿El código está bien estructurado, es consistente con el estilo y tiene un formato consistente?
   - ¿Hay algún procedimiento no llamado o innecesario o algún código inaccesible?
   - ¿Hay stubs o rutinas de prueba remanentes[88] en el código?
   - ¿Cualquier código puede ser sustituido por llamadas a componentes reutilizables externos o funciones de librería?
   - ¿Hay algún bloque de código repetido que podría condensarse en un único procedimiento?
   - ¿El uso del almacenamiento es eficiente?
   - ¿Se utilizan constantes simbólicas en lugar de constantes en forma de "número mágico" o constantes en forma de cadena?
   - ¿Algunos de los módulos son excesivamente complejos y deberían reestructurarse o dividirse en varios módulos?

2. Documentación
   - ¿El código está clara y debidamente documentado con un estilo de comentarios fácil de mantener?
   - ¿Todos los comentarios son consistentes con el código?
   - ¿La documentación cumple con las normas aplicables?

3. Variables
   - ¿Todas las variables se encuentran debidamente definidas con nombres significativos, consistentes y claros?
   - ¿Hay alguna variable redundante o no utilizada?

4. Operaciones de aritmética
   - ¿El código evita comparar números en coma flotante para la igualdad?
   - ¿El código evita de forma sistemática errores de redondeo?
   - ¿El código evita adiciones y sustracciones de números de magnitudes muy diferentes?

[84] "Procesamiento Transaccional en Línea" es la traducción de "Online Transactional Processing".
[85] "OLTP" es el acrónimo del término en inglés "Online Transactional Processing".
[86] "Procesamiento Analítico en Línea" es la traducción de "Online Analytical Processing".
[87] "OLAP" es el acrónimo del término en inglés ""Online Analytical Processing".
[88] "stub remanente" es la traducción de "leftover stub", "rutina de prueba remanente" es la traducción de "leftover test routine".

- ¿Se prueban los divisores para los valores cero o ruido?

5. Bucles y ramas
- ¿Todos los bucles, ramas y construcciones lógicas están completos, correctos y debidamente anidados?
- ¿Se prueban primero los casos más comunes en cadenas IF-ELSEIF?
- ¿Todos los casos están cubiertos en un bloque IF-ELSE o CASE, incluyendo cláusulas ELSE o DEFAULT?
- ¿Todas las sentencias "case" tienen un valor por defecto?
- ¿Las condiciones de terminación de los bucles son obvias e invariablemente alcanzables?
- ¿Los índices o subíndices se encuentran debidamente inicializados, inmediatamente antes del bucle?
- ¿Algunas sentencias incluidas dentro de los bucles pueden situarse fuera de los bucles?
- ¿El código del bucle evita manipular la variable correspondiente al índice o utilizarla a la salida del bucle?

6. Programación Defensiva
- ¿Se prueban los índices, punteros y subíndices con respecto a los límites de las matrices, registros o archivos?
- ¿Se prueban los datos importados y los argumentos de entrada para comprobar su validez y compleción?
- ¿Están asignadas todas las variables de salida?
- ¿Se opera el elemento de dato correcto en cada sentencia?
- ¿Se liberan todas las asignaciones de memoria?
- ¿Se utilizan tiempos de espera[89] o trampas de errores[90] para el acceso de dispositivos externos?
- ¿Se comprueba la existencia de archivos antes de intentar el acceso a ellos?
- ¿Todos los archivos y dispositivos quedan en el estado correcto en el momento de la terminación del programa?

Para ejemplos adicionales de listas de comprobación utilizadas en revisiones de código en distintos niveles de prueba ver [Web-6].

[89] "tiempo de espera" es la traducción de "timeout".
[90] "trampa de error" es la traducción de "error trap".

6. Herramientas de Prueba y Automatización - 195 minutos

Palabras clave

pruebas guiadas por datos, herramienta de depuración, herramienta de siembra de faltas, herramienta de prueba de hipervínculos, pruebas guiadas por palabras clave, herramienta de pruebas de rendimiento, herramienta de registro/reproducción, analizador estático, herramienta de ejecución de pruebas, herramienta de gestión de pruebas

Objetivos de Aprendizaje para Herramientas de Prueba y Automatización

6.1 Integración e Intercambio de Información Entre Herramientas

APT-6.1.1 (K2) Explicar los aspectos técnicos a tener en cuenta cuando se utilizan múltiples herramientas de forma conjunta.

6.2 Definición del Proyecto de Automatización de Pruebas

APT-6.2.1 (K2) Resumir las actividades que el Analista de Pruebas Técnicas lleva a cabo cuando se organiza un proyecto de automatización.
APT-6.2.2 (K2) Resumir las diferencias entre la automatización guiada por datos y por palabras clave.
APT-6.2.3 (K2) Resumir los aspectos técnicos comunes que provocan que la automatización de los proyectos no logre alcanzar el retorno de la inversión esperado.
APT-6.2.4 (K3) Crear una tabla de palabras clave tomando como base a un proceso de negocio dado.

6.3 Herramientas de Prueba Específicas

APT-6.3.1 (K2) Resumir la finalidad de las herramientas de siembra de faltas e inyección de faltas.
APT-6.3.2 (K2) Resumir las principales características y problemas de implementación de las pruebas de rendimiento y las herramientas de monitorización.
APT-6.3.3 (K2) Explicar el objeto general de las herramientas utilizadas para las pruebas basadas en la Web.
APT-6.3.4 (K2) Explicar cómo las herramientas respaldan el concepto de pruebas basadas en modelos.
APT-6.3.5 (K2) Perfilar el objeto de las herramientas utilizadas para dar soporte a las pruebas de componente y el proceso de construcción.

6.1 Integración e Intercambio de Información Entre Herramientas

Si bien la responsabilidad de seleccionar e integrar las herramientas compete al Jefe de Pruebas, el Analista de Pruebas Técnicas puede tener que revisar la integración de una herramienta o conjunto de herramientas para garantizar un seguimiento preciso de los datos derivados de las distintas áreas de pruebas, tales como análisis estáticos, automatización de la ejecución de las pruebas y gestión de la configuración. Asimismo, en función de las aptitudes de programación del Analista de Pruebas Técnicas, también podrá participar en la creación del código que integrará las herramientas que no se integren "directamente"[91].

Un conjunto de herramientas[92] ideal debería eliminar la duplicación de información en el conjunto de todas las herramientas. Almacenar los guiones de ejecución de prueba tanto en una base de datos de gestión de pruebas como en el sistema de gestión de la configuración, es más costoso y propenso a errores. Sería mejor contar con un sistema de gestión de pruebas que incluyera un componente de gestión de la configuración o que sea capaz de integrarse con una herramienta de gestión de la configuración ya instaurada en la organización. Unas herramientas de seguimiento de defectos y gestión de pruebas bien integradas permitirán a un probador lanzar un informe de defectos durante la ejecución de los casos de prueba sin tener que salir de la herramienta de gestión de pruebas. Unas herramientas de análisis estático bien integradas deberían ser capaces de informar al sistema de gestión de defectos sobre cualquier incidencia y advertencia[93] descubiertas (si bien esto debería ser configurable debido a la de advertencias que podrían generarse).

La adquisición de un paquete de herramientas de prueba de un mismo fabricante no significa que, automáticamente, las herramientas podrán trabajar conjuntamente de forma correcta. Al considerar el enfoque para integrar herramientas de forma conjunta, es preferible un enfoque centrado en datos. Los datos deben intercambiarse sin una intervención manual, oportunamente y con una exactitud garantizada, incluyendo la recuperación ante faltas. Si bien es útil contar con una experiencia de usuario consistente, la captura, el almacenamiento, la protección y la presentación de datos deberían ser el foco principal de la integración de herramientas.

Una organización debería evaluar el coste de automatizar el intercambio de información frente al riesgo de perder información o permitir que los datos dejen de estar sincronizados debido a una necesaria intervención manual. Dado que la integración puede ser cara o difícil, debería ser una consideración primordial en la estrategia general para las herramientas.

Algunos entornos de desarrollo integrado (IDE) pueden simplificar la integración entre herramientas que son capaces de trabajar en dicho entorno. Esto ayuda a unificar el aspecto y la sensación de las herramientas que comparten el mismo marco de trabajo. Sin embargo, una interfaz de usuario similar no garantizará un intercambio fluido de información entre los componentes. Es posible que sea necesario codificar (o programar) para completar la integración.

6.2 Definición del Proyecto de Automatización de la Prueba

Para ser rentables, las herramientas de pruebas y en particular las herramientas de automatización de pruebas, deben contar con una arquitectura y diseño concebidos cuidadosamente. Implementar una estrategia de automatización de pruebas sin una arquitectura sólida, normalmente, tiene como resultado un conjunto de herramientas costoso de mantener, insuficiente para el objetivo e incapaz de alcanzar el retorno sobre la inversión objetivo.

Un proyecto de automatización de pruebas debería considerarse un proyecto de desarrollo de software. Esto incluye la necesidad de documentos de arquitectura, documentos de diseño detallados, revisiones de diseño y código, pruebas de componente y de integración de componentes, así como las pruebas de sistema finales. Las pruebas se pueden demorar de forma innecesaria o

[91] "directamente" es la traducción de "out of the box" en este contexto.
[92] "conjunto de herramientas" es la traducción de toolset".
[93] "advertencia es la traducción de "warning".

complicarse cuando se utiliza código de automatización de pruebas inestable o inexacto. Hay varias actividades que el Analista de Pruebas Técnicas lleva a cabo con respecto a la automatización de pruebas. Éstas incluyen:

- Determinar quién será el responsable de la ejecución de las pruebas.
- Seleccionar la herramienta adecuada para la organización, el progresión temporal, las capacidades del equipo, los requisitos de mantenimiento (tener en cuenta que esto podría implicar tomar la decisión de crear una herramienta en lugar de adquirirla).
- Definir los requisitos de interfaz entre la herramienta de automatización y otras herramientas tales como las herramientas de gestión de pruebas y las herramientas de gestión de defectos.
- Seleccionar el enfoque de automatización, es decir, las pruebas guiadas por palabras clave o por datos (ver Sección 6.2.1 a continuación).
- Trabajar con el Jefe de Pruebas para calcular el coste de la implementación, incluyendo la formación.
- Desarrollar el calendario del proyecto de automatización y asignar el tiempo de mantenimiento correspondiente.
- Formar a los Analistas de Pruebas y a los Analistas de Negocio en el uso y suministro de datos para la automatización.
- Determinar cómo se ejecutarán las pruebas automatizadas.
- Determinar cómo se combinarán los resultados de las pruebas automatizadas con los resultados de las pruebas manuales.

Estas actividades y las decisiones resultantes afectarán a la escalabilidad y mantenibilidad de la solución de automatización. Se debe dedicar tiempo suficiente a buscar las opciones, investigar las herramientas y tecnologías disponibles y comprender los planes de futuro de la organización. Algunas de estas actividades requieren más atención que otras, especialmente durante el proceso de decisión. Estas se estudian en más detalle en las secciones a continuación.

6.2.1 Selección del Enfoque de Automatización

La automatización de las pruebas no se limita a las pruebas a través de la Interfaz Gráfica de Usuario[94] (“GUI”)[95]. Existen herramientas para automatizas a nivel de la API, a través de una Interfaz de Línea de Comando[96] (“CLI”)[97] y otros puntos de interfaz en el software sujeto a pruebas. Una de las primeras decisiones que Analista de Pruebas Técnicas debe hacer se refiere a la interfaz a la que acceder más efectiva para automatizar las pruebas.

Una de las dificultades de las pruebas a través de la Interfaz Gráfica de Usuario es la tendencia de la Interfaz Gráfica de Usuario a cambiar a medida que el software evoluciona. En función de la forma en que se haya diseñado el código de automatización de las pruebas, esto puede suponer una sobrecarga de mantenimiento importante. Por ejemplo, utilizar la capacidad de grabación/reproducción de una herramienta de automatización de pruebas puede dar lugar a casos de prueba automatizados (a menudo denominados guiones de prueba) que han dejan de ejecutarse de la forma esperada si la Interfaz Gráfica de Usuario cambia. Esto es así porque el guion grabado captura las interacciones con los objetos gráficos cuando el probador ejecuta el software manualmente. Si el objeto al que se ha accedido cambia, es posible que los guiones grabados también requieran actualización para reflejar dichos cambios.

Las herramientas de captura y reproducción se pueden utilizar como un punto de partida cómodo para desarrollar guiones de automatización. El probador graba una sesión de pruebas y, a continuación, se modifica el guion grabado para mejorar la mantenibilidad (por ejemplo, sustituyendo secciones en el guion grabado con funciones reutilizables).

[94] “Interfaz Gráfica de Usuario” es la traducción de “Graphic User Interface”.
[95] “GUI” es al acrónimo del término en inglés “Graphic User Interface”.
[96] “Interfaz de Línea de Comando” es la traducción de “Command Line Interface”.
[97] “CLI” es al acrónimo del término en inglés “Command Line Interface”.

Dependiendo del software sujeto a pruebas, los datos utilizados para cada prueba pueden diferir aunque los pasos de prueba ejecutados sean prácticamente idénticos (por ejemplo, las pruebas del tratamiento de errores para un campo de entrada introduciendo múltiples valores inválidos y comprobando el error que se devuelve para cada uno de ellos). No es eficiente desarrollar y mantener un guion de prueba automatizado para cada uno de los valores que se va a probar. Una solución técnica habitual para este problema es mover los datos de los guiones a un almacén externo, como una hoja de datos o una base de datos. Las funciones se formulan para acceder a los datos específicos en cada ejecución del guion de prueba, lo que permite que un único guion trabaje sobre un conjunto de datos de prueba que proporciona los valores de entrada y los valores de resultado esperados (por ejemplo, un valor presentado en un campo de texto o un mensaje de error). Este enfoque se denomina guiado por datos. Cuando se utiliza este enfoque, se desarrolla un guion de prueba que procesará los datos suministrados, así como un arnés y la infraestructura necesaria para soportar la ejecución del guion o conjunto de guiones. Los Analistas de Prueba, que conocen la función de negocio del software, crean los datos efectivos incluidos en la hoja de cálculo o base de datos. La división de tareas permite a los responsables de desarrollar guiones de pruebas (por ejemplo, el Analista de Pruebas Técnicas) concentrarse en la implementación de los guiones de automatización inteligentes mientras que el Analista de Pruebas mantiene la propiedad de la prueba efectiva. En la mayoría de los casos, el Analista de Pruebas será responsable de ejecutar los guiones de prueba una vez que se ha implementado y probado la automatización.

Otro enfoque, conocido como pruebas guiadas por palabras clave o palabras de acción, va un paso más allá al separar también la acción a realizar sobre los datos facilitados del guion de prueba [Buwalda01]. Para llevar a cabo esta separación adicional, los expertos del dominio (por ejemplo, los Analistas de Pruebas) crean un meta lenguaje de alto nivel que es descriptivo más que directamente ejecutable. Cada sentencia de este lenguaje describe un proceso de negocio, completo o parcial, del dominio que puede requerir pruebas. Por ejemplo, las palabras clave de un proceso de negocio podrían incluir “IniciarSesion”[98], “CrearUsuario”[99] y “BorrarUsuario”[100]. Una palabra clave describe una acción de alto nivel que se llevará a cabo en el dominio de la aplicación. Las acciones de nivel más bajo que denotan una interacción con la propia interfaz del software, tales como: “PulsarBoton”[101] , “SeleccionarDeLaLista”[102], o “RecorrerArbol”[103] también se pueden definir y se pueden utilizar para probar las capacidades Interfaz Gráfica de Usuario que no encajan perfectamente con las palabras clave del proceso.

Una vez definidas las palabras clave y los datos que se van a utilizar, la persona que llevará a cabo la automatización de las pruebas (por ejemplo, el Analista de Pruebas Técnicas) traduce las palabras clave del proceso de negocio y las acciones más bajo nivel a código de automatización de pruebas. Las palabras y acciones clave junto con los datos a utilizar, se pueden almacenar en hojas de cálculo o introducirse mediante herramientas específicas que soportan la automatización de pruebas guiadas por palabras clave. El marco de trabajo de la automatización de pruebas 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 claves y llaman a las funciones o a los guiones de prueba adecuados que los implementa. Los ejecutables se implementan de una forma altamente modular para permitir mapear fácilmente palabras claves específicas. Se requieren capacidades de programación para poder implementar estos guiones modulares.

Esta separación del conocimiento de la lógica del negocio de la programación real necesaria para implementar los guiones de automatización de pruebas aporta el uso más efectivo de los recursos de prueba. El Analista de Pruebas Técnicas, en el rol de automatizador[104] de pruebas, puede aplicar de manera efectiva sus capacidades de programación sin tener que ser un experto del dominio en muchas áreas del negocio.

[98] “IniciarSesion” es la traducción de “Login”.
[99] “CrearUsuario” es la traducción de “CreateUser”.
[100] BorrarUsuario” es la traducción de “DeleteUser”.
[101] “PulsarBoton” es la traducción de“ClickButton”.
[102] SeleccionarDeLaLista” es la traducción de “SelectFromList”.
[103] “RecorrerArbol” es la traducción de“TraverseTree”.
[104] “automatizador” es la traducción de “automator”.

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 sobre la inversión de la automatización.

En cualquier diseño de automatización, es importante anticiparse y gestionar fallos software. Si se produce un fallo, el automatizador debe establecer qué debe hacer el software. ¿Debería grabarse el fallo y continuar las pruebas? ¿Deberían finalizar las pruebas? ¿Se puede tratar el fallo con una acción específica (como pulsar un botón en un cuadro de diálogo) o quizás añadiendo una demora en la prueba? Los fallos software no tratados pueden corromper resultados de pruebas subsiguientes así como provocar un problema con la prueba que se estaba ejecutando cuando se produjo el fallo.

También es importante tener en cuenta el estado del sistema al inicio y al final de las pruebas. Puede ser necesario asegurar que el sistema vuelve a un estado predefinido después de finalizar la ejecución de las pruebas. Esto permitirá ejecutar repetidamente un juego de pruebas automatizadas sin intervención manual para restablecer el estado del sistema a un estado conocido. Para hacer esto, la automatización de pruebas podría tener, por ejemplo, que borrar los datos que ha creado o modificar el estado de los registros de una base de datos. El marco de trabajo de automatización debería garantizar que se ha alcanzado una terminación adecuada al finalizar las pruebas (es decir, finalizar la sesión una vez que las pruebas hayan sido completadas).

6.2.2 Modelado de Procesos de Negocio para la Automatización

Para implementar un enfoque de pruebas guiadas por palabras clave para la automatización de las pruebas, los procesos de negocio a probar deben estar modelados 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 Pruebas que estén trabajando en el proyecto.

En general, las palabras clave se utilizan para representar interacciones de negocio de alto nivel con un sistema. Por ejemplo, “Cancelar_Pedido”[105] puede requerir comprobar la existencia del pedido, verificar los derechos de acceso de la persona que está solicitando la cancelación, mostrar el pedido a cancelar y solicitar confirmación de la cancelación. El Analista de Pruebas utiliza las secuencias de palabras clave (tales como, “Inciar_Sesion”[106], “Seleccionar Pedido”[107] , “Cancelar_Pedido”) y los datos de prueba relevantes para especificar casos de prueba. A continuación se incluye una tabla simple, de entradas guiadas por palabra clave que podrían utilizarse para probar la capacidad del software a añadir, restablecer y borrar cuentas de usuario:

Palabra clave	Usuario	Contraseña	Resultado
Agregar_Usuario[108]	Usuario1	**Pass1**	Mensaje de usuario añadido.
Agregar_Usuario	@Rec34	**@Rec35**	Mensaje de usuario añadido.
Reiniciar_Contraseña[109]	Usuario1	**Bienvenido**	Mensaje de confirmación de reinicio de la contraseña.
Borrar_Usuario	Usuario1		Mensaje de nombre de usuario/contraseña inválido.
Agregar_Usuario	Usuario3	**Pass3**	Mensaje de usuario añadido.
Borrar_Usuario	Usuario2		Mensaje de usuario no encontrado.

El guion de automatización que utiliza esta tabla buscará los valores de entrada que utilizará el guion de automatización. Por ejemplo, cuando llega a la fila con la palabra clave “Borrar_Usuario”, sólo se requiere el nombre de usuario. Para añadir un nuevo usuario se requieren el nombre de usuario y la contraseña. Los valores de entrada también se pueden referenciar desde un almacén de datos según se muestra en la segunda palabra clave “Agregar_Usuario”, en la que se introduce una referencia al dato en lugar del propio dato, proporcionando más flexibilidad para acceder al dato que puede estar

[105] “Cancelar_Pedido” es la traducción de “Cancel_Order”.
[106] “Inciar_Sesion” es la traducción de“Login”.
[107] “Seleccionar Pedido” es la traducción de“Select_Order”.
[108] “Agregar_Usuario” es la traducción de “Add_User”.
[109] “Reiniciar_Contraseña” es la traducción de “Reset_Password”.

cambiando a medida que se ejecutan las pruebas. Esto permite combinar técnicas guiadas por datos con el esquema de palabras clave.

Algunos problemas que se deben tener en cuenta:

- Cuanto más granulares sean las palabras clave, más específicos serán los escenarios que se pueden cubrir, pero el mantenimiento del lenguaje de alto nivel puede resultar más complejo.
- Permitir que los Analistas de Pruebas especifiquen acciones de bajo nivel ("PulsarBoton"[110] "ClickButton", "SeleccionarDeLaLista", etc.) hace que las pruebas por palabras clave cuenten con mayor capacidad para tratar distintas situaciones. No obstante, dado que estas acciones están directamente vinculadas a la Interfaz Gráfica de Usuario, también pueden hacer que las pruebas requieran más mantenimiento cuando se produzcan los cambios.
- El uso de palabras clave agregadas puede simplificar el desarrollo pero complicar el mantenimiento. Por ejemplo, puede haber seis palabras clave distintas que en conjunto crean un registro. ¿Debería crearse una palabra clave que llame a las seis palabras clave consecutivamente para simplificar esa acción?
- Independientemente de la cantidad de análisis que se dedique al lenguaje de palabras clave, muchas veces tendrán que crearse palabras clave nuevas y diferentes. Hay dos dominios independientes para una palabra clave (es decir, la lógica de negocio detrás de ella y la funcionalidad de automatización para ejecutarla). Por lo tanto, debe crearse un proceso para abordar su tratamiento en ambos dominios.

La automatización de pruebas basadas en palabras clave puede reducir significativamente los costes de mantenimiento de la automatización de pruebas, pero es más cara, más difícil de desarrollar y su diseño requiere más tiempo si se desea lograr el retorno sobre la inversión deseado.

6.3 Herramientas de Prueba Específicas

Esta sección contiene información sobre herramientas que los Analistas de Pruebas Técnicas suelen utilizar, más allá de lo estudiado en los programas de estudio de Nivel Básico [ISTQB_FL_SYL] y Analista de Pruebas de Nivel Avanzado [ISTQB_ALTA_SYL].

6.3.1 Herramientas de Siembra e Inyección de Faltas

La siembra de faltas[111] utilizará una herramienta similar a un compilador para crear tipos únicos o limitados de faltas de código de manera sistemática. Estas herramientas introducen defectos, de forma deliberada, en el objeto de prueba con el objetivo de evaluar la calidad de los juegos de pruebas (es decir, su capacidad para detectar defectos).

La inyección de faltas se centra en probar cualquier mecanismo de tratamiento de faltas creado en el objeto de la prueba sometiéndolo a condiciones anormales. Las herramientas de inyección de faltas proporcionan entradas incorrectas al software para asegurar que el software puede hacer frente a la falta.

Los Analistas de Pruebas Técnicas utilizan normalmente los dos tipos de herramientas, si bien también pueden utilizarlas el desarrollador para probar código recién desarrollado.

6.3.2 Herramientas de Pruebas de Rendimiento

Las herramientas de pruebas de rendimiento presentan dos funciones principales:

- Generación de carga.
- Medición y análisis de la respuesta del sistema ante una carga determinada.

La generación de carga se lleva a cabo implementando un perfil operativo predefinido (ver Sección 4.5.4) como un guion. El guion puede ser capturado inicialmente por un único usuario (que utilice,

[110] "PulsarBoton" es la traducción de ClickButton",
[111] "falta" es la traducción de "fault" y es un sinónimo de "defecto".

posiblemente, una herramienta de grabación/reproducción) y, posteriormente, implementado para el perfil operativo específico utilizando la herramienta de pruebas de rendimiento. Esta implementación debe tener en cuenta la variación de datos por transacción (o conjunto de transacciones).

Las herramientas de rendimiento generan una carga simulando un gran número de múltiples usuarios (usuarios "virtuales") que siguen sus perfiles operativos designados para generar volúmenes específicos de datos de entrada. En comparación con los guiones de automatización de ejecución de pruebas individuales, muchos guiones de pruebas de rendimiento reproducen la interacción de un usuario con un sistema en el nivel de protocolo de comunicaciones y no simulando la interacción del usuario mediante una interfaz gráfica de usuario. Esto, normalmente, reduce el número de "sesiones" independientes necesarias durante las pruebas. Algunas herramientas de generación de carga también pueden activar [112] la aplicación utilizando su interfaz de usuario para medir más estrechamente el tiempo de respuesta cuando el sistema se encuentra bajo carga.

La herramienta de pruebas de rendimiento lleva a cabo una serie de mediciones que posibilitan el análisis durante o después de la ejecución de la prueba. Las métricas extraídas y los informes obtenidos incluyen:

- Número de usuarios simulados durante la prueba.
- Número y tipo de transacciones generadas por los usuarios simulados y la tasa de llegada[113] de las transacciones.
- Tiempos de respuesta a las solicitudes de transacciones específicas realizadas por los usuarios.
- Informes y gráficos de carga con respecto a tiempos de respuesta
- Informes sobre el uso de recursos (por ejemplo, uso en función del tiempo con valores mínimos y máximos).

Entre los factores significativos que se deben tener en cuenta en la implementación de herramientas de pruebas de rendimiento, se encuentran:

- El hardware y el ancho de banda de la red que se necesitan para generar la carga.
- La compatibilidad de la herramienta con el protocolo de comunicaciones empleado por el sistema sujeto a prueba.
- La flexibilidad de la herramienta para permitir la implementación de distintos perfiles operativos de forma sencilla.
- Las facilidades de monitorización, análisis y suministro de información[114] requeridas.

Normalmente, las herramientas de pruebas de rendimiento se adquieren más que desarrollarse internamente debido al esfuerzo que supone desarrollarlas. Sin embargo, puede ser útil desarrollar una herramienta de rendimiento específica si las restricciones técnicas impiden que se utilice un producto disponible o si el perfil de la carga y las facilidades que ofrece son relativamente simples.

6.3.3 Herramientas Para Pruebas Basadas en la Web

Hay toda una variedad de herramientas especializadas de código abierto y comerciales disponibles para realizar pruebas web. La siguiente lista muestra el objetivo de algunas herramientas de pruebas basadas en la web más comunes:

- Las herramientas de pruebas de hipervínculo se utilizan para explorar y comprobar que no hay hipervínculos rotos o ausentes en un sitio web.
- Los comprobadores HTML y XML son herramientas que comprueban el cumplimiento de las normas HTML y XML de las páginas creadas por un sitio web.
- Los simuladores de carga para probar cómo reaccionará el servidor cuando se conecte un gran número de usuarios.
- Herramientas de ejecución de automatización ligera que funcionan con varios navegadores.
- Herramientas para explorar[115] el servidor en busca de ficheros huérfanos (sin enlace[116]).

[112] "activar" es la traducción de "drive" en este contexto.
[113] "tasa de llegada" es la traducción de "arrival rate".
[114] "suministro de información" es la traducción de "reporting".

- Correctores ortográficos específicos de HTML.
- Herramientas de comprobación de Hojas de Estilo en Cascada[117] (CSS[118]).
- Herramientas para comprobar posibles infracciones de las normas, por ejemplo, Sección 508 de las normas de accesibilidad en EE.UU. o M/376 en Europa.
- Herramientas que detectan una serie de problemas de seguridad.

[Web-7] es una buena fuente de herramientas de pruebas web de código abierto. La organización detrás de este sitio web establece los estándares para Internet y ofrece una serie de herramientas que permiten comprobar errores contra esos estándares.

Algunas herramientas que incluyen un motor araña de la web también pueden facilitar, información sobre el tamaño de las páginas, el tiempo necesario para descargarlas y si una página está presente o no (por ejemplo, HTTP error 404). Esto ofrece información útil para el desarrollador, el administrador de la web[119] y el probador.

Los Analistas de Pruebas y los Analistas de Pruebas Técnicas utilizan estas herramientas, principalmente, durante las pruebas de sistema.

6.3.4 Herramientas de Soporte Para Pruebas Basadas en Modelos

Las Pruebas Basadas en Modelos (PBM) constituyen una técnica según la cual se utiliza un modelo formal, como una máquina de estados finitos, para describir el comportamiento previsto en tiempo de ejecución[120] de un sistema controlado por software. Las herramientas PBM comerciales (ver [Utting 07)], a menudo, facilitan un motor que permite a un usuario “ejecutar” el modelo. Los hilos de ejecución interesantes se pueden guardar y utilizar como casos de prueba. Otros modelos ejecutables, tales como las Redes de Petri y Gráficos de Estado también soprtan PBM. Los modelos PBM (y las herramientas) se pueden utilizar para generar amplios conjuntos de hilos de ejecución distintos.

Las herramientas PBM pueden ayudar a reducir el gran número de caminos posibles que se pueden generar en un modelo.

Realizar las pruebas empleando estas herramientas puede ofrecer un punto de vista distinto del software sujeto a pruebas. Esto puede dar lugar al descubrimiento de defectos que las pruebas funcionales podrían haber omitido.

6.3.5 Pruebas de Componente y Herramientas de Construcción[121]

Mientras que las herramientas de automatización de pruebas de componente y construcción son herramientas propias de los desarrolladores, en muchos casos, los Analistas de Pruebas Técnicas tienen que utilizarlas y mantenerlas, especialmente en el contexto de desarrollos ágiles.

Las herramientas de pruebas de componente, a menudo, son específicas del lenguaje empleado para programar un módulo. Así por ejemplo, si se ha utilizado Java como lenguaje de programación, JUnit se podría utilizar para automatizar las pruebas unitarias. Muchos otros lenguajes tienen sus propias herramientas de prueba especiales, que se conocen conjuntamente como marcos de trabajo xUnit. Este tipo de marco de trabajo genera objetos de prueba para cada clase creada, simplificando así las tareas que el programador debe realizar al automatizar las pruebas de componente.

---

[115] “explorar” es la traducción de “scan”.
[116] “enlace” es la traducción de “link”.
[117] “Hojas de Estilo en Cascada” es la traducción de “Cascading Style Sheets”.
[118] “CSS” es el acrónimo del término en inglés de “Cascading Style Sheets”.
[119] “administrador de la web” es la traducción de “webmaster”.
[120] “tiempo de ejecución” es la traducción de “execution-time”.
[121] “construcción” es la traducción de “build”. La construcción incluye, entre otras funciones: (control de versión + calidad del código + compilación + …).

Las herramientas de depuración facilitan las pruebas de componente manuales a muy bajo nivel, permitiendo a los desarrolladores y Analistas de Pruebas Técnicas modificar los valores de las variables durante la ejecución y recorrer el código línea por línea durante las pruebas. Las herramientas de depuración también se utilizan para ayudar al desarrollador a aislar e identificar problemas en el código cuando el equipo de pruebas informa un fallo.

Las herramientas de automatización de la construcción, a menudo, permiten desencadenar automáticamente una nueva construcción en el momento en que se modifica un componente. Una vez finalizada la construcción, otras herramientas ejecutan automáticamente las pruebas de componente. Este nivel de automatización en torno al proceso de construcción es habitual en entornos de integración continua.

Si se configuran correctamente, este conjunto de herramientas tienen un efecto muy positivo en la calidad de las construcciones que pasan a pruebas. En caso de que un cambio realizado por un programador introduzca defectos de regresión en la construcción, normalmente esto provocará que alguna de las pruebas automatizadas falle, desencadenando inmediatamente la investigación de la causa de los fallos antes de que la construcción pase al entorno de pruebas.

7. Referencias

7.1 Normas

Las siguientes normas aparecen en los capítulos siguientes:

- ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems.
  Capítulo 5
- IEC-61508
  Capítulo 2
- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE).
  Capítulo 4
- [ISO9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality.
  Capítulo 4
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12B.1992.
  Capítulo 2

7.2 Documentos ISTQB

- [ISTQB_AL_OVIEW] Resumen del Nivel Avanzado ISTQB, Versión 2012.
- [ISTQB_ALTA_SYL Programa de Estudio Analista de Pruebas de Nivel Avanzado ISTQB, Versión 2012.
- [ISTQB_FL_SYL] Programa de Estudio de Nivel Básico ISTQB, Versión 2011.
- [ISTQB_GLOSSARY] ISTQB Glosario de Términos utilizados en Pruebas de Software, Versión 2.2, 2012.

7.3 Libros

[Bass03] Len Bass, Paul Clements, Rick Kazman “Software Architecture in Practice (2nd edition)”, Addison-Wesley 2003] ISBN 0-321-15495-9
[Bath08] Graham Bath, Judy McKay, “The Software Test Engineer’s Handbook”, Rocky Nook, 2008, ISBN 978-1-933952-24-6
[Beizer90] Boris Beizer, "Software Testing Techniques Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3
[Beizer95] Boris Beizer, “Black-box Testing”, John Wiley & Sons, 1995, ISBN 0-471-12094-4
[Buwalda01]: Hans Buwalda, “Integrated Test Design and Automation” Addison-Wesley Longman, 2001, ISBN 0-201-73725-6
[Copeland03]: Lee Copeland, “A Practitioner's Guide to Software Test Design”, Artech House, 2003, ISBN 1-58053-791-X
[Gamma94] Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994, ISBN 0-201-63361-2
[Jorgensen07]: Paul C. Jorgensen, “Software Testing, a Craftsman’s Approach third edition”, CRC press, 2007, ISBN-13:978-0-8493-7475-3
[Kaner02]: Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”; Wiley, 2002, ISBN: 0-471-08112-4

[Koomen06]: Tim Koomen, Leo van der Aalst, Bart Broekman, Michael Vroon, "TMap Next for result-driven testing"; UTN Publishers, 2006, ISBN: 90-72194-79-9
[McCabe76] Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976. PP 308-320
[NIST96] Arthur H. Watson and Thomas J. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric", NIST Special Publication 500-235, Prepared under NIST Contract 43NANB517266, September 1996.
[Splaine01]: Steven Splaine, Stefan P. Jaskiel, “The Web-Testing Handbook”, STQE Publishing, 2001, ISBN 0-970-43630-0
[Utting 07] 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. A pesar de que estas referencias fueron consultadas en el momento de la publicación de este Programa de Estudio de Nivel Avanzado, el ISTQB no se responsabiliza de la disponibilidad de tales referencias.

[Web-1] www.testingstandards.co.uk
[Web-2] http://www.nist.gov NIST National Institute of Standards and Technology,
[Web-3] http://www.codeproject.com/KB/architecture/SWArchitectureReview.aspx
[Web-4] http://portal.acm.org/citation.cfm?id=308798
[Web-5] http://www.processimpact.com/pr_goodies.shtml
[Web-6] http://www.ifsq.org
[Web-7] http://www.W3C.org
Chapter 4: [Web-1], [Web-2]
Chapter 5: [Web-3], [Web-4], [Web-5], [Web-6]
Chapter 6: [Web-7]