Capítulo 1: Fundamentos de la Prueba (180 minutos)

- El estudiante aprende los principios básicos relacionados con las pruebas, las razones por las que se requiere la prueba y cuáles son los objetivos de la prueba.
- El estudiante entiende el proceso de la prueba, las principales actividades de la prueba y el testware.
- El estudiante entiende las habilidades esenciales para la prueba.

• K1: Recordar
• K2: Comprender
• K3: Aplicar

Términos

cobertura, depuración, defecto, error, falla, calidad, aseguramiento de la calidad, causa raíz, análisis de prueba, base de prueba, caso de prueba, finalización de prueba, condición de prueba, control de la prueba, datos de prueba, diseño de la prueba, ejecución de prueba, implementación de prueba, monitoreo de prueba, objeto de prueba, objetivo de prueba, planificación de prueba, procedimiento de prueba, resultado de prueba, prueba, testware, validación, verificación

Objetivos de Aprendizaje para el Capítulo 1:

1.1 ¿Qué es Probar?
FL-1.1.1 (K1) Identificar los objetivos típicos de la prueba
FL-1.1.2 (K2) Diferenciar pruebas de depuración
1.2 ¿Por qué es Necesario Probar?
FL-1.2.1 (K2) Ejemplificar por qué las pruebas son necesarias
FL-1.2.2 (K1) Recordar la relación entre las pruebas y el aseguramiento de la calidad
FL-1.2.3 (K2) Distinguir entre causa raíz, error, defecto y falla
1.3 Principios de la Prueba
FL-1.3.1 (K2) Explicar los siete principios de prueba
1.4 Actividades de la Prueba, Testware y Roles de la Prueba
FL-1.4.1 (K2) Resumir las diferentes actividades y tareas de prueba
FL-1.4.2 (K2) Explicar el impacto del contexto en el proceso de prueba
FL-1.4.3 (K2) Diferenciar el testware que respalda las actividades de prueba
FL-1.4.4 (K2) Explicar el valor de mantener la trazabilidad
FL-1.4.5 (K2) Comparar los diferentes roles en las pruebas
1.5 Habilidades Esenciales y Buenas Prácticas en las Pruebas
FL-1.5.1 (K2) Proporcionar ejemplos de las habilidades genéricas requeridas para probar
FL-1.5.2 (K1) Recordad las ventajas del enfoque de equipo completo
FL-1.5.3 (K2) Distinguir los beneficios y los inconvenientes de la independencia de las pruebas

1.1 ¿Qué es Probar?

Las **pruebas** son un conjunto de actividades que implican verificación, es decir, verificar si el sistema cumple con los requisitos especificados,y también implican validación, lo que significa comprobar si el sistema cumple con las necesidades de los usuarios y otras partes interesadas en su entorno operativo.

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

Las pruebas pueden desencadenar fallas causadas por defectos en el software (pruebas dinámicas) o pueden encontrar directamente defectos en el objeto de prueba (pruebas estáticas).
Cuando las pruebas dinámicas desencadenan una falla, la depuración se ocupa de encontrar las causas de esta falla (defectos), analizar estas causas y eliminarlas.

La **depuración** es un proceso que implica
• Reproducción de una falla
• Diagnóstico (encontrar la causa raíz)
• Corregir la causa
Las pruebas de confirmación posteriores comprueban si las correcciones resolvieron el problema.

1.2 ¿Por qué es Necesario Probar?

Las pruebas, como una forma de control de calidad,

- Ayudan a lograr los objetivos acordados dentro del alcance, el tiempo, la calidad y las limitaciones presupuestarias establecidas. Las pruebas de componentes, sistemas y documentación asociada ayudan a identificar defectos en el software. Las pruebas proporcionan un medio rentable para detectar defectos.
- Proporcionan un medio para evaluar directamente la calidad de un objeto de prueba en varias etapas en el Ciclo de Vida de Desarrollo de Software. Estas medidas se utilizan como parte de una actividad de gestión de proyectos más amplia, lo que contribuye a las decisiones de pasar a la siguiente etapa del Ciclo de Vida de Desarrollo de Software, como la decisión de entrega.
- Pueden ser necesarias para cumplir con los requisitos contractuales o legales, o para cumplir con los estándares reglamentarios.

Las pruebas son una forma de control de calidad (QC).
El control de calidad es un enfoque correctivo orientado al producto que se centra en aquellas actividades que apoyan el logro de niveles apropiados de calidad. Las pruebas son una forma importante de control de calidad, mientras que otras incluyen métodos formales (comprobación del modelo y prueba de que esté correcto), simulación y creación de prototipos.

QA es un enfoque preventivo orientado a procesos que se centra en la implementación y mejora de

procesos. Funciona sobre la base de que, si un buen proceso se sigue correctamente, entonces generará un buen producto. El aseguramiento de calidad se aplica tanto a los procesos de desarrollo como a los de prueba, y es responsabilidad de todos en un proyecto.

Los resultados de la prueba son utilizados por QA y QC. En QC se utilizan para corregir defectos, mientras que en QA proporcionan retroalimentación sobre qué tan bien se están desempeñando los procesos de desarrollo y prueba.

Los seres humanos cometen **errores** (equivocaciones), que producen **defectos** (faltas, bugs), que a su vez pueden resultar en **fallas**.

Los **defectos** se pueden encontrar en la documentación, como una especificación de requisitos o un script de prueba, en el código fuente o en un artefacto de soporte como un archivo de compilación.

Los defectos en los artefactos producidos anteriormente en el Ciclo de Vida de Desarrollo de Software, si no se detectan, a menudo conducen a artefactos defectuosos más adelante en el ciclo de vida. Si se ejecuta un defecto en el código, el sistema puede no hacer lo que debería hacer, o hacer algo que no debería, causando una **falla**.

Algunos defectos siempre resultarán en una falla si se ejecutan, mientras que otros solo resultarán en una falla en circunstancias específicas, y algunos nunca pueden resultar en una falla.

Los errores y defectos no son la única causa de las fallas. Las **fallas** también pueden ser causadas por condiciones ambientales, como cuando la radiación o el campo electromagnético causan defectos en el firmware.

Una causa raíz es una razón fundamental para la ocurrencia de un problema (por ejemplo, una situación que conduce a un error). Las causas raíz se identifican a través del análisis de causa raíz, que generalmente se realiza cuando se produce una falla o se identifica un defecto. Se cree que se pueden prevenir otras fallas o defectos similares o reducir su frecuencia abordando la causa raíz, tal como eliminándola.

1.3 Principios de la Prueba

A lo largo de los años se han sugerido una serie de principios de prueba que ofrecen pautas generales aplicables a todas las pruebas. Este Programa de Estudio describe siete de estos principios.

1. Las pruebas muestran la presencia, no la ausencia de defectos. Las pruebas pueden mostrar que hay defectos presentes en el objeto de prueba, pero no pueden probar que no hay defectos (Buxton 1970). Las pruebas reducen la probabilidad de que los defectos permanezcan sin descubrir en el objeto de prueba, pero **incluso si no se encuentran defectos, las pruebas no pueden probar que el objeto de prueba esté correcto.**
2. Las pruebas exhaustivas son imposibles. La prueba de todo no es factible excepto en casos triviales (Manna 1978). En lugar de intentar realizar pruebas exhaustivas, se deben utilizar técnicas

de prueba (consulte el capítulo 4), priorización de casos de prueba (consulte la sección 5.1.5) y pruebas basadas en el riesgo (consulte la sección 5.2) para enfocar los esfuerzos de prueba.
3. Las pruebas tempranas ahorran tiempo y dinero. Los defectos que se eliminan al principio del proceso no causarán defectos posteriores en los productos de trabajo derivados. El costo de la calidad se reducirá ya que se producirán menos fallas más adelante en el SDLC (Boehm 1981). Para encontrar defectos temprano, tanto las pruebas estáticas (ver capítulo 3) como las pruebas dinámicas (ver capítulo 4) deben iniciarse lo antes posible.
4. Los defectos se agrupan. Un pequeño número de componentes del sistema generalmente contienen la mayoría de los defectos descubiertos o son responsables de la mayoría de las fallas operativas (Enders 1975). Este fenómeno es una ilustración del principio de Pareto. Los grupos de defectos pronosticados y los grupos de defectos reales observados durante las pruebas o en funcionamiento son un dato importante para las pruebas basadas en el riesgo (ver sección 5.2).
5. Las pruebas se desgastan. Si las mismas pruebas se repiten muchas veces, se vuelven cada vez más ineficaces para detectar nuevos defectos (Beizer 1990). Para superar este efecto, es posible que sea necesario modificar las pruebas existentes y los datos de las pruebas, y es posible que sea necesario escribir nuevas pruebas. Sin embargo, en algunos casos, repetir las mismas pruebas puede tener un resultado beneficioso, por ejemplo, en las pruebas de regresión automatizadas (ver sección 2.2.3).
6. Las pruebas dependen del contexto. No existe un enfoque único universalmente aplicable a las pruebas. Las pruebas se realizan de manera diferente en diferentes contextos (Kaner 2011).
7. Falacia de la ausencia de defectos. Es una falacia (es decir, un concepto erróneo) esperar que la verificación del software garantice el éxito de un sistema. Probar minuciosamente todos los requisitos especificados y corregir todos los defectos encontrados aún podría producir un sistema que no satisfaga las necesidades y expectativas de los usuarios, que no ayude a lograr los objetivos comerciales del cliente y que sea inferior en comparación con otros sistemas de la competencia.
Además de la verificación, también se debe llevar a cabo la validación (Boehm 1981).

1.4 Actividades de la Prueba, Testware y Roles de la Prueba

Las pruebas dependen del contexto, pero, a un alto nivel, hay conjuntos comunes de actividades de prueba sin las cuales es menos probable que las pruebas logren los objetivos de la prueba. Estos conjuntos de actividades de prueba forman un proceso de prueba.

La **planificación** de la prueba consiste en definir los objetivos de la prueba y luego seleccionar el enfoque que mejor logre los objetivos dentro de las limitaciones impuestas por el contexto general.
El **monitoreo y el control de prueba**. El monitoreo de pruebas implica la comprobación continua de todas las actividades de prueba y la comparación del progreso real con el plan. El control de pruebas implica tomar las acciones necesarias para cumplir con los objetivos de las pruebas.
El **análisis de prueba** incluye el análisis de la base de prueba para identificar características comprobables y para definir, y priorizar las condiciones de prueba asociadas, junto con los riesgos y niveles de riesgo relacionados (ver sección 5.2). La base de prueba y los objetos de prueba también se evalúan para identificar defectos que pueden contener y para evaluar su comprobabilidad. El

análisis de prueba a menudo se apoya en el uso de técnicas de prueba (consulte el capítulo 4). El análisis de prueba responde a la pregunta "¿qué probar?" en términos de criterios de cobertura medibles.
El **diseño de la prueba** incluye la elaboración de las condiciones de prueba en casos de prueba y otro testware (por ejemplo, cartas de prueba). Esta actividad a menudo implica la identificación de ítems de cobertura, que sirven como guía para especificar entradas de los casos de prueba. Las técnicas de prueba (véase el capítulo 4) se pueden utilizar para apoyar esta actividad. El diseño de prueba también incluye la definición de los requisitos de datos de prueba, el diseño del entorno de prueba y la identificación de cualquier otra infraestructura y herramientas requeridas. El diseño de la prueba responde a la pregunta "¿cómo probar?".
La **implementación de prueba** incluye crear o adquirir el testware necesario para la ejecución de la prueba (por ejemplo, datos de prueba). Los casos de prueba se pueden organizar en procedimientos de prueba y, a menudo, se ensamblan en juegos de pruebas. Se crean scripts de prueba manuales y automatizados. Los procedimientos de prueba se priorizan y organizan dentro de un cronograma de ejecución de prueba para una ejecución eficiente de la prueba (consulte la sección 5.1.5). El entorno de prueba es construido y verificado para configurarse correctamente.
La **ejecución de pruebas** incluye la ejecución de las pruebas de acuerdo con el cronograma de ejecución de pruebas (ejecuciones de pruebas). La ejecución de la prueba puede ser manual o automatizada. La ejecución de pruebas puede tomar muchas formas, incluyendo pruebas continuas o sesiones de prueba por pares. Los resultados reales de las pruebas se comparan con los resultados esperados. Los resultados de la prueba se registran. Las anomalías se analizan para identificar sus causas probables. Este análisis nos permite reportar las anomalías en base a los fallos observados (ver apartado 5.5).
Las **actividades de la finalización de pruebas** generalmente ocurren en los hitos del proyecto (por ejemplo, en la entrega, en el final de la iteración, en la finalización del nivel de prueba) para cualquier defecto no resuelto, solicitudes de cambio o ítems de la pila (backlog) del producto creados. Cualquier testware que pueda ser útil en el futuro se identifica y archiva o se entrega a los equipos apropiados. El entorno de prueba se cierra a un estado acordado. Las actividades de prueba se analizan para identificar lecciones aprendidas y mejoras para futuras iteraciones, entregas o proyectos (consulte la sección 2.1.6). Se crea un informe de finalización de la prueba y se comunica a las partes interesadas.

La forma en que se lleva a cabo la prueba dependerá de una serie de factores contextuales que incluyen:
• Partes interesadas (necesidades, expectativas, requisitos, voluntad de cooperar, etc.)
• Miembros del equipo (habilidades, conocimientos, nivel de experiencia, disponibilidad, necesidades de capacitación, etc.)
• Dominio del negocio (criticidad del objeto de prueba, riesgos identificados, necesidades del mercado, regulaciones legales específicas, etc.)
• Factores técnicos (tipo de software, arquitectura del producto, tecnología utilizada, etc.)
• Limitaciones del proyecto (alcance, tiempo, presupuesto, recursos, etc.)
• Factores organizativos (estructura organizativa, políticas existentes, prácticas utilizadas, etc.)
• Ciclo de vida del desarrollo de software (prácticas de ingeniería, métodos de desarrollo, etc.)
• Herramientas (disponibilidad, usabilidad, cumplimiento, etc.)

Estos factores tendrán un impacto en muchos temas relacionados con la prueba, incluyendo: estrategia de prueba, técnicas de prueba utilizadas, grado de automatización de la prueba, nivel requerido de cobertura, nivel de detalle de la documentación de la prueba, informes, etc.

El **Testware** es un conjunto de productos de trabajo

• Los **productos de trabajo de planificación de pruebas** incluyen: plan de pruebas, cronograma de pruebas, registro de riesgos, y criterios de entrada y salida (consulte la sección 5.1). El registro de riesgos es una lista de riesgos junto con la probabilidad del riesgo, el impacto del riesgo y la información sobre la mitigación de riesgos (ver sección 5.2). El cronograma de pruebas, el registro de riesgos, y los criterios de entrada y salida a menudo forman parte del plan de pruebas.
• Los **productos de trabajo de monitoreo y control de pruebas** incluyen: informes de progreso de la prueba (ver sección 5.3.2), documentación de directivas de control (ver sección 5.3) e información de los riesgos (ver sección 5.2).
• Los **productos de trabajo de análisis de prueba** incluyen: condiciones de prueba (priorizadas) (por ejemplo, criterios de aceptación, consulte la sección 4.5.2) e informes de defectos con respecto a los defectos en la base de prueba (si no son corregidos directamente).
• Los **productos de trabajo de diseño de prueba** incluyen: casos de prueba (priorizados), cartas de prueba, ítems de cobertura, requisitos de datos de prueba y requisitos de entorno de prueba.
• Los **productos de trabajo de implementación de prueba** incluyen: procedimientos de prueba, scripts de prueba automatizados, conjuntos de pruebas, datos de prueba, cronograma de ejecución de prueba y elementos del entorno de prueba. Los ejemplos de elementos del entorno de prueba incluyen: stubs, drivers (controladores), simuladores y virtualizaciones de servicios.
• Los **productos de trabajo de ejecución de prueba** incluyen: registros de prueba e informes de defecto (consulte la sección 5.5).
• Los **productos de trabajo de finalización de la prueba** incluyen: informe de finalización de la prueba (consulte la sección 5.3.2), elementos de acción para la mejora de proyectos o iteraciones posteriores, lecciones aprendidas documentadas y solicitudes de cambio (por ejemplo, ítems de la pila(backlog) del producto).

La **trazabilidad** es importante para:

- Implementar un monitoreo y control de prueba efectivos,
- Respaldar la evaluación de la cobertura, por lo que es muy útil si se definen criterios de cobertura medibles en la base de la prueba. Los criterios de cobertura pueden funcionar como indicadores clave de rendimiento para impulsar las actividades que muestran en qué medida se han alcanzado los objetivos de la prueba (véase la sección 1.1.1). Por ejemplo:
    - • La trazabilidad de los casos de prueba a los requisitos puede verificar que los requisitos están cubiertos por los casos de prueba.
    - • La trazabilidad de los resultados de la prueba a los riesgos se puede utilizar para evaluar el nivel de riesgo residual en un objeto de prueba.

- determinar el impacto de los cambios, facilita las auditorías de prueba y ayuda a cumplir con los criterios de gobierno de TI.
- Permitir que el progreso de la prueba y los informes de finalización sean más fáciles de entender al incluir el estado de los elementos básicos de la prueba.
- comunicar los aspectos técnicos de las pruebas a las partes interesadas de una manera comprensible.
- Proporcionar información para evaluar la calidad del producto, la capacidad del proceso y el progreso del proyecto en relación con los objetivos comerciales.

En este Programa de Estudio, se cubren **dos roles principales en las pruebas: un rol de gestión de pruebas y un rol de la prueba.**

El **rol de gestión de pruebas** asume la responsabilidad general del proceso de prueba, el equipo de prueba y el liderazgo de las actividades de prueba.
Se centra principalmente en las actividades de planificación de pruebas, monitoreo y control de pruebas, y finalización de pruebas.

El **rol de prueba** asume la responsabilidad general del aspecto de ingeniería (técnico) de las pruebas.
Se centra principalmente en las actividades de análisis de pruebas, diseño de pruebas, implementación de pruebas y ejecución de pruebas.
Diferentes personas pueden asumir estos roles en diferentes momentos. También es posible que una persona asuma los roles de prueba y gestión de pruebas al mismo tiempo.

1.5 Habilidades Esenciales y Buenas Prácticas en las Pruebas

Las siguientes habilidades son particularmente relevantes para los probadores:
• Conocimiento en pruebas (uso de técnicas de prueba)
• Rigurosidad, cuidado, curiosidad, atención a los detalles, ser metódico (para identificar defectos, especialmente los que son difíciles de encontrar)
• Buenas habilidades de comunicación, escucha activa, ser un jugador de equipo (para interactuar de manera efectiva con todas las partes interesadas, para transmitir información a los demás, para ser entendido, y para informar y discutir defectos)
• Pensamiento analítico, pensamiento crítico, creatividad (para aumentar la efectividad de las pruebas)
• Conocimientos técnicos (para aumentar la eficiencia de las pruebas, por ejemplo, mediante el uso de herramientas de prueba apropiadas)
• Conocimiento del dominio (para poder comprender y comunicarse con los usuarios finales/representantes comerciales)

En el **enfoque de equipo completo**, cualquier miembro del equipo con los conocimientos y habilidades necesarios puede realizar cualquier tarea, y todos son responsables de la calidad.

Mejora:
- la dinámica del equipo,
- la comunicación
- la colaboración dentro del equipo,

Crea sinergia al permitir que los diversos conjuntos de habilidades dentro del equipo se aprovechen en beneficio del proyecto.

Los probadores trabajan en estrecha colaboración con otros miembros del equipo para garantizar que se alcancen los niveles de calidad deseados.

Un cierto grado de **independencia** hace que el probador sea más efectivo para encontrar defectos debido a las diferencias entre los sesgos cognitivos del autor y del probador (cf. Salman 1995).

Los productos de trabajo pueden ser probados por su autor (sin independencia), por los compañeros del autor del mismo equipo (cierta independencia), por probadores de fuera del equipo del autor pero dentro de la organización (alta independencia) o por probadores externos de la organización (muy alta independencia). Para la mayoría de los proyectos, generalmente es mejor llevar a cabo pruebas con múltiples niveles de independencia (por ejemplo, desarrolladores que realizan pruebas de componente e integración de componentes, equipos de prueba que realizan pruebas de sistema e integración de sistemas, y representantes comerciales que realizan pruebas de aceptación).

El principal beneficio de la independencia de las pruebas es que es probable que los **probadores independientes reconozcan diferentes tipos de fallas y defectos en comparación con los desarrolladores debido a sus diferentes antecedentes, perspectivas técnicas y sesgos.** Además, un probador independiente puede verificar, desafiar o refutar las suposiciones hechas por las partes interesadas durante la especificación e implementación del sistema.

La desventaja es que **los probadores independientes pueden estar aislados del equipo de desarrollo, lo que puede conducir a una falta de colaboración, problemas de comunicación o una relación adversa con el equipo de desarrollo.** Los desarrolladores pueden perder el sentido de responsabilidad por la calidad. Los probadores independientes pueden ser vistos como un cuello de botella o ser culpados por retrasos en la liberación.

• Capítulo 2: Pruebas a lo Largo del Ciclo de Vida del Desarrollo de Software (130 minutos)

- El estudiante aprende cómo las pruebas se incorporan a diferentes enfoques de desarrollo.
- El estudiante aprende los conceptos de enfoques de prueba primero, así como DevOps.
- El estudiante aprende sobre los diferentes niveles de prueba, tipos de prueba y pruebas de mantenimiento.

• K1: Recordar
• K2: Comprender
• K3: Aplicar

Términos

pruebas de aceptación, pruebas de caja negra, pruebas de integración de componentes, pruebas de componentes, pruebas de confirmación, pruebas funcionales, pruebas de integración, pruebas de mantenimiento, pruebas no funcionales, pruebas de regresión, desplazamiento hacia la izquierda, pruebas de integración de sistemas, pruebas del sistema, nivel de prueba, objeto de prueba, tipo de prueba, pruebas de caja blanca

Objetivos de Aprendizaje para el Capítulo 2:

2.1 Pruebas en el Contexto de un Ciclo de Vida de Desarrollo de Software
FL-2.1.1 (K2) Explicar el impacto del ciclo de vida de desarrollo de software elegido en las pruebas
FL-2.1.2 (K1) Recordar las buenas prácticas de prueba que se aplican a todos los ciclos de vida de desarrollo de software
FL-2.1.3 (K1) Recordar los ejemplos de enfoques de desarrollo de prueba primero
FL-2.1.4 (K2) Resumir cómo DevOps podría tener un impacto en las pruebas
FL-2.1.5 (K2) Explicar el enfoque de desplazamiento hacia la izquierda
FL-2.1.6 (K2) Explicar cómo se pueden utilizar las retrospectivas como mecanismo para la mejora de procesos
2.2 Niveles de Prueba y Tipos de Prueba
FL-2.2.1 (K2) Distinguir los diferentes niveles de prueba
FL-2.2.2 (K2) Distinguir los diferentes tipos de prueba
FL-2.2.3 (K2) Distinguir la prueba de confirmación de la prueba de regresión
2.3 Pruebas de Mantenimiento
FL-2.3.1 (K2) Resumir las pruebas de mantenimiento y sus activadores

2.1 Pruebas en el Contexto de un Ciclo de Vida de Desarrollo de Software

Un modelo de ciclo de vida de desarrollo de software (SDLC) es una representación abstracta y de alto nivel del proceso de desarrollo de software. Un modelo SDLC define cómo las diferentes fases de desarrollo y los tipos de actividades realizadas dentro de este proceso se relacionan entre sí, tanto lógica como cronológicamente. Los ejemplos de modelos SDLC incluyen: modelos de desarrollo secuencial (por ejemplo, el modelo cascada, el modelo V), modelos de desarrollo iterativo (por ejemplo, el modelo

en espiral, la creación de prototipos) y modelos de desarrollo incremental (por ejemplo, el proceso unificado).
Algunas actividades dentro de los procesos de desarrollo de software también pueden describirse mediante métodos de desarrollo de software más detallados y prácticas Ágiles.

Los ejemplos incluyen:
desarrollo guiado por pruebas de aceptación (ATDD), desarrollo guiado por el comportamiento (BDD), diseño guiado por dominios (DDD), programación extrema (XP), desarrollo guiado por características (FDD), Kanban, Lean IT, Scrum y desarrollo guiado por pruebas (TDD).

Las pruebas deben adaptarse al SDLC para tener éxito. La elección del SDLC tiene un impacto en:
• Alcance, momento o secuencia de las actividades de prueba (por ejemplo, niveles de prueba y tipos de prueba)
• Nivel de detalle de la documentación de la prueba
• Elección de técnicas de prueba y enfoque de prueba
• Alcance de la automatización de pruebas
• Rol y responsabilidades de un probador

En los modelos de desarrollo secuencial, en las fases iniciales, los probadores suelen participar en revisiones de requisitos, análisis de pruebas y diseño de pruebas. El código ejecutable generalmente se crea en las fases posteriores, por lo que normalmente las pruebas dinámicas no se pueden realizar al principio del SDLC.

En algunos modelos de desarrollo iterativos e incrementales, se supone que cada iteración entrega un prototipo de trabajo o incremento de producto. Esto implica que en cada iteración se pueden realizar pruebas estáticas y dinámicas en todos los niveles de prueba. La entrega frecuente de incrementos requiere retroalimentación rápida y pruebas de regresión extensas.

El desarrollo ágil de software asume que el cambio puede ocurrir a lo largo del proyecto. Por lo tanto, la documentación ligera del producto de trabajo y la extensa automatización de pruebas para facilitar las pruebas de regresión se ven favorecidas en proyectos ágiles. Además, la mayoría de las pruebas manuales tienden a realizarse utilizando técnicas de prueba basadas en la experiencia que no requieren un extenso análisis y diseño de pruebas previas.

TDD, ATDD y BDD son enfoques de desarrollo similares, donde las pruebas se definen como un medio para dirigir el desarrollo. Cada uno de estos enfoques implementa el principio de pruebas tempranas y sigue un enfoque de desplazamiento hacia la izquierda , ya que las pruebas se definen antes de escribir el código. Apoyan un modelo de desarrollo iterativo. Estos enfoques se caracterizan de la siguiente manera:

Desarrollo Guiado por Pruebas (TDD)
• Dirige la codificación a través de casos de prueba (en lugar de un extenso diseño de software) (Beck 2003)
• Las pruebas se escriben primero, luego el código se escribe para satisfacer las pruebas, y luego las pruebas y el código se refactorizan.

Desarrollo Guiado por Pruebas de Aceptación (ATDD)
• Deriva las pruebas de los criterios de aceptación como parte del proceso de diseño del sistema (Gärtner 2011)
• Las pruebas se escriben antes de que se desarrolle parte de la aplicación, para satisfacer las pruebas

Desarrollo Guiado por el Comportamiento (BDD)
• Expresa el comportamiento deseado de una aplicación con casos de prueba escritos en una forma simple de lenguaje natural, que es fácil de entender por las partes interesadas, generalmente utilizando el formato Dado/Cuándo/Entonces. (Given/When/Then) (Chelimsky 2010)
• Luego los casos de prueba son traducidos automáticamente en pruebas ejecutables
Para todos los enfoques anteriores, las pruebas pueden persistir como pruebas automatizadas para garantizar la calidad del código en futuras adaptaciones / refactorizaciones.

DevOps es un enfoque organizacional que tiene como objetivo crear sinergia al lograr que el desarrollo (incluidas las pruebas) y las operaciones trabajen juntos para lograr un conjunto de objetivos comunes. DevOps requiere un cambio cultural dentro de una organización para cerrar las brechas entre el desarrollo (incluidas las pruebas) y las operaciones mientras tratan sus funciones con el mismo valor. DevOps promueve la autonomía del equipo, la retroalimentación rápida, las cadenas de herramientas integradas y las prácticas técnicas como la integración continua (CI) y la entrega continua (CD). Esto permite a los equipos construir, probar y entregar código de alta calidad más rápido a través de un canal (pipeline) de entrega de DevOps (Kim 2016).

Desde la perspectiva de las pruebas, algunos de los **beneficios de DevOps** son:
• Retroalimentación rápida sobre la calidad del código y si los cambios afectan negativamente al código existente
• La Integración Continua (CI) promueve un enfoque de desplazamiento hacia la izquierda en las pruebas (consulte la sección 2.1.5) alentando a los desarrolladores a presentar código de alta calidad acompañado de pruebas de componentes y análisis estático
• Promueve procesos automatizados como Integración Continua y Entrega Continua (CI/CD) que facilitan el establecimiento de entornos de prueba estables
• Aumenta la visión sobre las características de calidad no funcionales (por ejemplo, rendimiento, fiabilidad)
• La automatización a través de un canal (pipeline) de entrega reduce la necesidad de pruebas manuales repetitivas
• El riesgo en la regresión se minimiza debido a la escala y el rango de las pruebas de regresión automatizadas
DevOps no está exento de **riesgos** y desafíos, que incluyen:
• El canal (pipeline) de entrega de DevOps debe estar definido y establecido
• Las herramientas de Integración Continua y Entrega Continua (CI / CD) deben ser introducidas y mantenidas

• La automatización de pruebas requiere recursos adicionales y puede ser difícil de establecer y mantener.

Aunque DevOps viene con un alto nivel de pruebas automatizadas, las pruebas manuales, especialmente desde la perspectiva del usuario, seguirán siendo necesarias.

El principio de las pruebas tempranas (ver sección 1.3) a veces se denomina desplazamiento hacia la izquierda porque es un enfoque en el que las pruebas se realizan antes en el Ciclo de Vida del Desarrollo de Software (SDLC). El desplazamiento hacia la izquierda normalmente sugiere que las pruebas deben hacerse antes (por ejemplo, no esperar a que se implemente el código o que se integren los componentes), pero no significa que las pruebas posteriores en el SDLC deban descuidarse.

Hay algunas buenas prácticas que ilustran cómo lograr un "desplazamiento hacia la izquierda" en las pruebas, que incluyen:
• Revisar la especificación desde la perspectiva de las pruebas. Estas actividades de revisión de las especificaciones a menudo encuentran defectos potenciales, como ambigüedades, incompletitud e inconsistencias.
• Escribir casos de prueba antes de escribir el código y ejecutar el código en un arnés de prueba durante la implementación del código
• Utilizar Integración Continua (CI) e incluso mejor Entrega Continua (CD), ya que viene con retroalimentación rápida y pruebas automatizadas de componentes para acompañar el código fuente cuando se envía al repositorio de código
• Completar el análisis estático del código fuente antes de las pruebas dinámicas, o como parte de un proceso automatizado
• Realizar pruebas no funcionales comenzando en el nivel de prueba de componentes, cuando sea posible. Esta es una forma de desplazamiento a la izquierda, ya que estos tipos de prueba no funcionales tienden a realizarse más adelante en el Ciclo de Vida de Desarrollo de Software (SDLC) cuando están disponibles un sistema completo y un entorno de prueba representativo.

Un enfoque de desplazamiento hacia la izquierda podría resultar en capacitación, esfuerzo y/o costos adicionales más temprano en el proceso, pero se espera que ahorre esfuerzos y/o costos más adelante en el proceso.

Para el enfoque de desplazamiento hacia la izquierda es importante que las partes interesadas estén convencidas y se involucren en este concepto.

Las retrospectivas (también conocidas como “reuniones posteriores al proyecto” y retrospectivas del proyecto) a menudo se llevan a cabo al final de un proyecto o una iteración, en un hito de lanzamiento, o se pueden realizar cuando sea necesario. El momento y la organización de las retrospectivas dependen del modelo del Ciclo de Vida del Desarrollo de Software (SDLC) particular que se siga.

En estas reuniones, los participantes (no solo los probadores, sino también, por ejemplo, desarrolladores, arquitectos, propietarios de productos, analistas de negocios) discuten:

• ¿Qué fue exitoso y qué debe mantenerse?
• ¿Qué es lo que no ha tenido éxito y se puede mejorar?
• ¿Cómo incorporar las mejoras y retener los éxitos en el futuro?

Los resultados deben registrarse y normalmente forman parte del informe de finalización de la prueba (ver sección 5.3.2). Las retrospectivas son fundamentales para la implementación exitosa de la mejora continua y es importante que se haga un seguimiento de las mejoras recomendadas.

Los beneficios típicos de las pruebas incluyen:
• Aumento de la eficacia / eficiencia de las pruebas (por ejemplo, mediante la implementación de sugerencias para la mejora del proceso)
• Mayor calidad del testware (por ejemplo, revisando conjuntamente los procesos de prueba)
• Vinculación y aprendizaje en equipo (por ejemplo, como resultado de la oportunidad de plantear problemas y proponer puntos de mejora)
• Mejora de la calidad de la base de prueba (por ejemplo, ya que las deficiencias en el alcance y la calidad de los requisitos podrían abordarse y resolverse)
• Mejor cooperación entre el desarrollo y las pruebas (por ejemplo, a medida que la colaboración se revisa y optimiza regularmente)

2.2 Niveles de Prueba y Tipos de Prueba

Los **niveles de prueba** son grupos de actividades de prueba que se organizan y administran juntas. Cada nivel de prueba es una instancia del proceso de prueba, realizado en relación con el software en una etapa dada de desarrollo, desde componentes individuales hasta sistemas completos o, cuando corresponda, sistemas de sistemas.

• Las **pruebas de componentes** (también conocidas como pruebas unitarias) se centran en probar componentes de forma aislada. A menudo requiere soporte específico, como arneses de prueba o marcos de trabajo de prueba de unidades. Las pruebas de componentes normalmente son realizadas por desarrolladores en sus entornos de desarrollo.
• Las **pruebas de integración de componentes** (también conocidas como pruebas de integración de unidades) se centran en probar las interfaces y las interacciones entre los componentes. Las pruebas de integración de componentes dependen en gran medida de los enfoques de estrategia de integración como de abajo hacia arriba, de arriba hacia abajo o big-bang.
• Las **pruebas del sistema** se centran en el comportamiento general y las capacidades de todo un sistema o producto, a menudo incluyen pruebas funcionales de tareas de extremo a extremo y pruebas no funcionales de características de calidad. Para algunas características de calidad no funcionales, es preferible probarlas en un sistema completo en un entorno de prueba representativo (por ejemplo, usabilidad). Usando simulaciones de subsistemas también son posibles. Las pruebas del sistema pueden ser realizadas por un equipo de prueba independiente y están relacionadas con las especificaciones del sistema.
• Las **pruebas de integración de sistemas** se centran en probar las interfaces del sistema bajo prueba y otros sistemas y servicios externos. La prueba de integración de sistemas requiere entornos de prueba adecuados preferiblemente similares al entorno operativo.

• Las **pruebas de aceptación** se centran en la validación y en demostrar la preparación para el despliegue, lo que significa que el sistema satisface las necesidades comerciales del usuario.

Idealmente, las pruebas de aceptación deben ser realizadas por los usuarios previstos. Las principales formas de pruebas de aceptación son: pruebas de aceptación del usuario (UAT), pruebas de aceptación operativa, pruebas de aceptación contractual y regulatoria, pruebas alfa y pruebas beta.
Los niveles de prueba se distinguen por la siguiente lista no exhaustiva de atributos, para evitar la superposición de actividades de prueba:
• Objeto de prueba
• Objetivos de la prueba
• Base de prueba
• Defectos y fallas
• Enfoque y responsabilidades

Los **tipos de prueba** son grupos de actividades de prueba relacionadas con características de calidad específicas y la mayoría de esas actividades de prueba se pueden realizar en todos los niveles de prueba.

Las **pruebas funcionales** evalúan las funciones que un componente o sistema debe realizar. Las funciones son “lo que” debe hacer el objeto de prueba. El objetivo principal de las pruebas funcionales es comprobar la completitud funcional, la corrección funcional y la pertinencia funcional.

Las **pruebas no funcionales** evalúan atributos distintos de las características funcionales de un componente o sistema. Las pruebas no funcionales son las pruebas de "qué tan bien se comporta el sistema". El objetivo principal de las pruebas no funcionales es comprobar las características de calidad del software no funcional. La norma ISO/IEC 25010 proporciona la siguiente clasificación de las características de calidad del software no funcional:
• Eficiencia del rendimiento
• Compatibilidad
• Usabilidad
• Fiabilidad
• Seguridad
• Mantenibilidad
• Portabilidad

La **prueba de caja negra** (ver sección 4.2) se basa en especificaciones y deriva pruebas de documentación que es externa al objeto de prueba. El objetivo principal de las pruebas de caja negra es comprobar el comportamiento del sistema contra sus especificaciones.

Las **pruebas de caja blanca** (ver sección 4.3) se basan en la estructura y derivan pruebas de la implementación del sistema o la estructura interna (por ejemplo, código, arquitectura, flujos de trabajo y flujos de datos). El objetivo principal de las pruebas de caja blanca es cubrir la estructura subyacente mediante las pruebas al nivel aceptable.

Los cuatro tipos de prueba mencionados anteriormente se pueden aplicar a todos los niveles de prueba, aunque el enfoque será diferente en cada nivel. Se pueden usar diferentes técnicas de prueba para derivar las condiciones de prueba y los casos de prueba para todos los tipos de prueba mencionados.

Las **pruebas de confirmación** confirman que un defecto original se ha corregido con éxito. Dependiendo del riesgo, se puede probar la versión fija del software de varias maneras, incluyendo:
• ejecutando todos los casos de prueba que previamente hayan fallado debido al defecto, o, también
• agregando nuevas pruebas para cubrir cualquier cambio que sea necesario para corregir el defecto.
Sin embargo, cuando el tiempo o el dinero es corto al corregir defectos, las pruebas de confirmación pueden limitarse a simplemente ejercer los pasos que deberían reproducir la falla causada por el defecto y verificar que la falla no ocurra.
Las pruebas de regresión confirman que no se han causado consecuencias adversas por un cambio, incluida una corrección que ya ha sido probada. Estas consecuencias adversas podrían afectar al mismo componente donde se realizó el cambio, a otros componentes en el mismo sistema o incluso a otros sistemas conectados. Las pruebas de regresión pueden no estar restringidas al objeto de prueba en sí, sino que también pueden estar relacionadas con el entorno. Es aconsejable primero realizar un análisis de impacto para optimizar el alcance de las pruebas de regresión. El análisis de impacto muestra qué partes del software podrían verse afectadas.

Los conjuntos de pruebas de regresión se ejecutan muchas veces y, en general, el número de casos de prueba de regresión aumentará con cada iteración o entrega, por lo que las pruebas de regresión son un fuerte candidato para la automatización. La automatización de estas pruebas debe comenzar temprano en el proyecto. Cuando se utiliza Integración Continua (IC), como en DevOps (ver sección 2.1.4), es una buena práctica incluir también pruebas de regresión automatizadas. Dependiendo de la situación, esto puede incluir pruebas de regresión en diferentes niveles.

Se necesitan pruebas de confirmación y/o pruebas de regresión para el objeto de prueba en todos los niveles de prueba si se corrigen los defectos y/o se realizan cambios en estos niveles de prueba.

2.3 Pruebas de Mantenimiento

Existen diferentes categorías de mantenimiento, puede ser correctivo, adaptativo a los cambios en el entorno o mejorar el rendimiento o la capacidad de mantenimiento (consulte ISO/IEC 14764 para obtener más detalles), por lo que el mantenimiento puede implicar entregas/despliegues planificados y entregas/despliegues no planificados (correcciones en caliente).

El alcance de las pruebas de mantenimiento generalmente depende de:
• El grado de riesgo del cambio
• El tamaño del sistema existente
• El tamaño del cambio

Los activadores para el mantenimiento y las pruebas de mantenimiento se pueden clasificar de la siguiente manera:
• Modificaciones, como mejoras planificadas (es decir, basadas en entregas), cambios correctivos o correcciones en caliente.
• Actualizaciones o migraciones del entorno operativo, como de una plataforma a otra, que pueden requerir pruebas asociadas con el nuevo entorno, así como del software cambiado, o pruebas de conversión de datos cuando se migran datos de otra aplicación al sistema que se mantiene.
• Retiro, como cuando una aplicación llega al final de su vida útil. Cuando se retira un sistema,esto puede requerir pruebas de archivado de datos si se requieren largos períodos de retención de datos. La prueba de los procedimientos de restauración y recuperación después del archivo también puede ser necesaria en el caso de que se requieran ciertos datos durante el período de archivo.

• Capítulo 3: Pruebas Estáticas (80 minutos)

- El estudiante aprende sobre los conceptos básicos de las pruebas estáticas, el proceso de retroalimentación y revisión.
• K1: Recordar
• K2: Comprender
• K3: Aplicar

Términos
anomalía, pruebas dinámicas, revisión formal, revisión informal, inspección, revisión, análisis estático, pruebas estáticas, revisión técnica, revisión guiada

Objetivos de Aprendizaje para el Capítulo 3:

3.1 Conceptos Básicos de las Pruebas Estáticas
FL-3.1.1 (K1) Reconocer los tipos de productos que pueden ser examinados por las diferentes técnicas de prueba estática
FL-3.1.2 (K2) Explicar el valor de las pruebas estáticas
FL-3.1.3 (K2) Comparar y contrastar pruebas estáticas y dinámicas
3.2 Proceso de Retroalimentación y Revisión
FL-3.2.1 (K1) Identificar los beneficios de la retroalimentación temprana y frecuente de las partes interesadas
FL-3.2.2 (K2) Resumir las actividades del proceso de revisión
FL-3.2.3 (K1) Recordar qué responsabilidades se asignan a los roles principales al realizar revisiones
FL-3.2.4 (K2) Comparar y contrastar los diferentes tipos de revisión
FL-3.2.5 (K1) Recordar los factores que contribuyen a una revisión exitosa

3.1 Conceptos Básicos de las Pruebas Estáticas

En las **pruebas estáticas** no es necesario ejecutar el software bajo prueba. El código, la especificación del proceso, la especificación de la arquitectura del sistema u otros productos de trabajo se evalúan

mediante un examen manual (por ejemplo, revisiones) o con la ayuda de una herramienta (por ejemplo, análisis estático). Los objetivos de la prueba incluyen mejorar la calidad, detectar defectos y evaluar características como legibilidad, completitud, corrección, comprobabilidad y consistencia. Las pruebas estáticas se pueden aplicar tanto para la verificación como para la validación.

Los probadores, representantes comerciales y desarrolladores trabajan juntos durante el mapeo de ejemplos (backlogs) para garantizar que las historias de usuario y los productos de trabajo relacionados cumplan con los criterios definidos, por ejemplo, la Definición de Preparado (consulte la sección 5.1.3).

El análisis estático puede identificar problemas antes de las pruebas dinámicas y, a menudo, requiere menos esfuerzo, ya que no se requieren casos de prueba y, por lo general, se utilizan herramientas (consulte el capítulo 6). El análisis estático se incorpora a menudo en los marcos de trabajo de Integración Continua (IC) (véase la sección 2.1.4). Si bien se utiliza en gran medida para detectar defectos de código específicos, el análisis estático también se utiliza para evaluar la capacidad de mantenimiento y la seguridad. Los correctores ortográficos y las herramientas de legibilidad son otros ejemplos de herramientas de análisis estático.

Casi cualquier producto de trabajo puede ser examinado mediante pruebas estáticas. Los ejemplos incluyen documentos de especificación de requisitos, código fuente, planes de prueba, casos de prueba, ítems de la pila (backlog) del producto, cartas de prueba, documentación de proyectos, contratos y modelos.
Cualquier producto de trabajo que se pueda leer y comprender puede ser objeto de una revisión. Sin embargo, para el análisis estático, los productos de trabajo necesitan una estructura con la que puedan ser comprobados (por ejemplo, modelos, código o texto con una sintaxis formal).

Los productos de trabajo que no son apropiados para las pruebas estáticas se incluyen aquellos que son difíciles de interpretar por los seres humanos y que no deben ser analizados por herramientas (por ejemplo, código ejecutable de terceros debido a razones legales).

Las pruebas estáticas pueden:

- detectar defectos en las primeras fases del Ciclo de Vida Del Desarrollo de Software (SDLC), cumpliendo con el principio de pruebas tempranas, lo que generalmente resulta en menos defectos de código y un menor esfuerzo de desarrollo general.
- identificar defectos que no se pueden detectar mediante pruebas dinámicas.
- proporcionar la capacidad de evaluar la calidad y generar confianza en los productos de trabajo. Al verificar los requisitos documentados, las partes interesadas también pueden asegurarse de que estos requisitos describan sus necesidades reales.

Las pruebas estáticas y pruebas dinámicas se complementan entre sí. Tienen objetivos similares, como apoyar la detección de defectos en los productos de trabajo, pero también hay algunas diferencias, como:

• Las pruebas estáticas y dinámicas (con análisis de fallas) pueden conducir a la detección de defectos, sin embargo, hay algunos tipos de defectos que solo se pueden encontrar ya sea mediante pruebas estáticas o dinámicas.
• Las pruebas estáticas encuentran defectos directamente, mientras que las pruebas dinámicas producen fallas a partir de las cuales los defectos asociados se determinan a través de análisis posteriores.
• Las pruebas estáticas pueden detectar más fácilmente defectos que se encuentran en las rutas a través del código que rara vez se ejecutan o son difíciles de alcanzar mediante pruebas dinámicas.
• Las pruebas estáticas se pueden aplicar a productos de trabajo no ejecutables, mientras que las pruebas dinámicas solo se pueden aplicar a productos de trabajo ejecutables.
• Las pruebas estáticas se pueden usar para medir características de calidad que no dependen de la ejecución del código (por ejemplo, capacidad de mantenimiento), mientras que las pruebas dinámicas se pueden usar para medir características de calidad que dependen de la ejecución del código (por ejemplo, eficiencia de rendimiento).

Los defectos típicos que son más fáciles y/o más económicos de encontrar a través de pruebas estáticas incluyen:
• Defectos en los requisitos (por ejemplo, inconsistencias, ambigüedades, contradicciones, omisiones, inexactitudes, duplicaciones)
• Defectos de diseño (por ejemplo, estructuras de base de datos ineficientes, mala modularización)
• Ciertos tipos de defectos de codificación (por ejemplo, variables con valores indefinidos, variables no declaradas, código inalcanzable o duplicado, complejidad excesiva del código)
• Desviaciones de los estándares (por ejemplo, falta de adherencia a las convenciones de nomenclatura en los estándares de codificación)
• Especificaciones de interfaz incorrectas (por ejemplo, número, tipo u orden de parámetros no coincidentes)
• Tipos específicos de vulnerabilidades de seguridad (por ejemplo, desbordamientos del búfer)
• Brechas o inexactitudes en la cobertura de la base de pruebas (por ejemplo, pruebas faltantes para un criterio de aceptación)

3.2 Proceso de Retroalimentación y Revisión

La retroalimentación temprana y frecuente permite la comunicación temprana de posibles problemas de calidad. Los comentarios frecuentes de las partes interesadas en todo el Ciclo de Vida de Desarrollo de Software (SDLC) pueden prevenir malentendidos sobre los requisitos y garantizar que los cambios en los requisitos se entiendan e implementen antes. Esto ayuda al equipo de desarrollo a mejorar su comprensión de lo que están construyendo. Les permite centrarse en aquellas características que ofrecen el mayor valor a las partes interesadas y que tienen el impacto más positivo en los riesgos identificados.

La norma ISO/IEC 20246 define un proceso de revisión genérico. El tamaño de muchos productos de trabajo los hace demasiado grandes para ser cubiertos por una sola revisión. El proceso de revisión puede ser invocado un par de veces para completar la revisión de todo el producto de trabajo.
Las actividades en el proceso de revisión son:
• Planificación. Se define el alcance de la revisión, que comprende el propósito, el producto de trabajo que se revisará, las características de calidad que se evaluarán, las áreas donde enfocarse, los criterios de salida, la información de apoyo, como los estándares, el esfuerzo y los plazos para la revisión.
• Inicio de la Revisión. El objetivo es asegurarse de que todos y cada uno de los involucrados estén preparados para comenzar la revisión. Esto incluye asegurarse de que cada participante tenga acceso al producto de trabajo bajo revisión, entienda su rol y responsabilidades y reciba todo lo necesario para realizar la revisión.
• Revisión Individual. Cada revisor realiza una revisión individual para evaluar la calidad del producto de trabajo bajo revisión e identificar anomalías, recomendaciones y preguntas mediante la aplicación de una o más técnicas de revisión (por ejemplo, revisión basada en listas de comprobación, revisión basada en escenarios). La norma ISO/IEC 20246 proporciona más profundidad sobre diferentes técnicas de revisión. Los revisores registran todas sus anomalías, recomendaciones y preguntas identificadas.
• Comunicación y Análisis. Dado que las anomalías identificadas durante una revisión no son necesariamente defectos, todas estas anomalías deben analizarse y discutirse. Para cada anomalía, la decisión debe tomarse sobre su estado, propiedad y acciones requeridas. Esto se hace típicamente en una reunión de revisión, durante la cual los participantes también deciden cuál es el nivel de calidad del producto de trabajo revisado y qué acciones de seguimiento se requieren. Es posible que se requiera una revisión de seguimiento para completar las acciones.
• Reparación e Informes. Para cada defecto, se debe crear un informe de defecto para que se puedan seguir las acciones correctivas. Una vez que se alcanzan los criterios de salida, se puede aceptar el producto de trabajo. Se informan los resultados de la revisión.

Las revisiones involucran a varias partes interesadas, que pueden asumir varios roles. Las funciones principales y sus responsabilidades son:
• Director - decide lo que debe ser revisado y proporciona recursos, tales como el personal y tiempo para la revisión
• Autor – crea y corrige el producto de trabajo bajo revisión
• Moderador - (también conocido como el facilitador): asegura el funcionamiento efectivo de las reuniones de revisión, incluida la mediación, la gestión del tiempo y un entorno de revisión seguro en el que todos puedan hablar libremente
• Escriba - (también conocido como grabador): recopila anomalías de los revisores y registra la información de revisión, como las decisiones y las nuevas anomalías encontradas durante la reunión de revisión
• Revisor – realiza las revisiones. Un revisor puede ser alguien que trabaja en el proyecto, un experto en la materia o cualquier otra parte interesada.
• Líder de revisión - asume la responsabilidad general de la revisión, como decidir quién participará y organizar cuándo y dónde se llevará a cabo la revisión.

Otros roles más detallados son posibles, como se describe en el estándar ISO/IEC 20246.

Algunos tipos de revisión comúnmente utilizados son:
• **Revisión informal.** Las revisiones informales no siguen un proceso definido y no requieren un resultado documentado formal. El objetivo principal es detectar anomalías.
• **Revisión guiada.** Una revisión guiada, dirigido por el autor, puede servir para muchos objetivos, como evaluar la calidad y generar confianza en el producto de trabajo, educar a los revisores, obtener consenso, generar nuevas ideas, motivar y permitir a los autores mejorar y detectar anomalías. Los revisores puede que realicen una revisión individual antes de la revisión guiada, pero esto no es necesario.
• **Revisión Técnica.** Una revisión técnica es realizada por revisores técnicamente calificados y dirigida por un moderador. Los objetivos de una revisión técnica son obtener consenso y tomar decisiones con respecto a un problema técnico, pero también detectar anomalías, evaluar la calidad y generar confianza en el producto de trabajo, generar nuevas ideas y motivar y permitir a los autores mejorar.
• **Inspección.** Como las inspecciones son el tipo de revisión más formal, siguen el proceso genérico completo (ver sección 3.2.2). El objetivo principal es encontrar el número máximo de anomalías. Otros objetivos son evaluar la calidad, generar confianza en el producto del trabajo y motivar y capacitar a los autores para mejorar. Las métricas se recopilan y utilizan para mejorar el SDLC, incluido el proceso de inspección. En las inspecciones, el autor no puede actuar como el líder de revisión o escriba.

Hay varios factores que determinan el éxito de las revisiones, que incluyen:

• Definir objetivos claros y criterios de salida medibles. La evaluación de los participantes nunca debe ser un objetivo
• Elegir el tipo de revisión apropiado para lograr los objetivos dados y para adaptarse al tipo de producto de trabajo, los participantes de la revisión, las necesidades del proyecto y el contexto
• Realización de revisiones en trozos pequeños, para que los revisores no pierdan la concentración durante una revisión individual y/o la reunión de revisión (cuando se lleva a cabo)
• Proporcionar comentarios de las revisiones a las partes interesadas y los autores para que puedan mejorar el producto y sus actividades (consulte la sección 3.2.1)
• Proporcionar tiempo suficiente a los participantes para prepararse para la revisión
• Apoyo de la gerencia para el proceso de revisión
• Hacer que las revisiones formen parte de la cultura de la organización, para promover el aprendizaje y la mejora de los procesos
• Proporcionar una formación adecuada a todos los participantes para que sepan cómo cumplir su función
• Facilitación de reuniones

• Capítulo 4: Análisis y Diseño de Pruebas (390 minutos)

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

• K1: Recordar
• K2: Comprender
• K3: Aplicar

Términos

criterios de aceptación, desarrollo guiado por pruebas de aceptación, técnica de prueba de caja negra, análisis de valores límite, cobertura de rama, pruebas basadas en listas de comprobación, enfoque de prueba basado en la colaboración, cobertura, ítem de cobertura, pruebas de tabla de decisiones, partición de equivalencia, predicción de errores, técnica de prueba basada en la experiencia, pruebas exploratorias, pruebas de transición de estado, cobertura de sentencia, técnica de prueba, técnica de prueba de caja blanca

Objetivos de Aprendizaje para el Capítulo 4

4.1 Descripción General de las Técnicas de Prueba
FL-4.1.1 (K2) Distinguir las técnicas de prueba de caja negra, caja blanca y basadas en la experiencia
4.2 Técnicas de Prueba de Caja Negra
FL-4.2.1 (K3) Utilizar la partición de equivalencia para derivar los casos de prueba
FL-4.2.2 (K3) Utilizar el análisis de valores límite para derivar los casos de prueba
FL-4.2.3 (K3) Utilizar las pruebas de la tabla de decisiones para derivar los casos de prueba
FL-4.2.4 (K3) Utilizar las pruebas de transición de estado para derivar los casos de prueba
4.3 Técnicas de Caja Blanca
FL-4.3.1 (K2) Explicar las pruebas de sentencias
FL-4.3.2 (K2) Explicar las pruebas de rama
FL-4.3.3 (K2) Explicar el valor de las pruebas de caja blanca
4.4 Técnicas Basadas en la Experiencia
FL-4.4.1 (K2) Explicar la predicción de errores
FL-4.4.2 (K2) Explicar las pruebas exploratorias
FL-4.4.3 (K2) Explicar las pruebas basadas en listas de comprobación
4.5. Enfoques de Prueba Basados en la Colaboración
FL-4.5.1 (K2) Explicar cómo escribir historias de usuario en colaboración con desarrolladores y representantes de negocios
FL-4.5.2 (K2) Clasificar las diferentes opciones para escribir criterios de aceptación
FL-4.5.3 (K3) Utilizar el desarrollo guiado por pruebas de aceptación (ATDD) para derivar casos de prueba

4.1 Descripción General de las Técnicas de Prueba

Las **técnicas de prueba de caja negra** (también conocidas como técnicas basadas en especificaciones) se basan en un análisis del comportamiento especificado del objeto de prueba sin referencia a su estructura interna. Por lo tanto, los casos de prueba son independientes de cómo se implementa el software. En consecuencia, si la implementación cambia, pero el comportamiento

requerido sigue siendo el mismo, entonces los casos de prueba siguen siendo útiles.

Las **técnicas de prueba de caja blanca** (también conocidas como técnicas basadas en la estructura) se basan en un análisis de la estructura interna y el procesamiento del objeto de prueba. Como los casos de prueba dependen de cómo se diseña el software, solo se pueden crear después del diseño o la implementación del objeto de prueba.

Las **técnicas de prueba basadas en la experiencia** utilizan efectivamente el conocimiento y la experiencia de los probadores para el diseño y la implementación de casos de prueba. La efectividad de estas técnicas depende en gran medida de las habilidades del probador. Las técnicas de prueba basadas en la experiencia pueden detectar defectos que pueden pasarse por alto utilizando las técnicas de prueba de caja negra y caja blanca. Por lo tanto, las técnicas de prueba basadas en la experiencia son complementarias a las técnicas de prueba de caja negra y caja blanca.

4.2 Técnicas de Prueba de Caja Negra

Las técnicas de prueba de caja negra comúnmente utilizadas discutidas en las siguientes secciones son:

- **Partición Equivalente**
- **Análisis de Valores Límite**
- **Pruebas de Tabla de Decisión**
- **Pruebas de Transición de Estado**

Partición de Equivalencia

La partición de equivalencia (EP) divide los datos en particiones (conocidas como particiones de equivalencia) en función de la expectativa de que todos los elementos de una partición dada deben ser procesados de la misma manera por el objeto de prueba. La teoría detrás de esta técnica es que si un caso de prueba, que prueba un valor de una partición de equivalencia, detecta un defecto, este defecto también debe ser detectado por casos de prueba que prueban cualquier otro valor de la misma partición. Por lo tanto, una prueba para cada partición es suficiente.

Las particiones de equivalencia se pueden identificar para cualquier elemento de datos relacionado con el objeto de prueba, incluidas entradas, salidas, elementos de configuración, valores internos, valores relacionados con el tiempo y parámetros de interfaz. Las particiones pueden ser continuas o discretas, ordenadas o desordenadas, finitas o infinitas. Las particiones no deben superponerse y deben ser conjuntos no vacíos.

Para objetos de prueba simples la Partición de Equivalencia (EP) puede ser fácil, pero en la práctica, comprender cómo el objeto de prueba tratará diferentes valores a menudo es complicado. Por lo tanto, la partición debe hacerse con cuidado.

Una partición que contiene valores válidos se denomina partición válida. Una partición que contiene

valores inválidos se denomina partición inválida. Las definiciones de valores válidos e inválidos pueden variar entre equipos y organizaciones. Por ejemplo, los valores válidos pueden interpretarse como aquellos que deben ser procesados por el objeto de prueba o como aquellos para los que la especificación define su procesamiento. Los valores inválidos pueden interpretarse como aquellos que deben ser ignorados o rechazados por el objeto de prueba o como aquellos para los que no se define ningún procesamiento en la especificación del objeto de prueba.

En Partición de Equivalencia (EP), los elementos de cobertura son las particiones de equivalencia. Para lograr una cobertura del 100% con esta técnica, los casos de prueba deben ejercer todas las particiones identificadas (incluidas las particiones inválidas) cubriendo cada partición al menos una vez. La cobertura se mide como el número de particiones ejercidas por al menos un caso de prueba, dividido por el número total de particiones identificadas, y se expresa como un porcentaje.

Muchos objetos de prueba incluyen múltiples conjuntos de particiones (por ejemplo, objetos de prueba con más de un parámetro de entrada), lo que significa que un caso de prueba cubrirá particiones de diferentes conjuntos de particiones. El criterio de cobertura más simple en el caso de múltiples conjuntos de particiones se llama cobertura de Cada Elección (Ammann 2016). La cobertura de Cada Elección requiere que los casos de prueba ejerzan cada partición de cada conjunto de particiones al menos una vez. La cobertura de Cada Elección no tiene en cuenta combinaciones de particiones.

Análisis de Valores Límite

El análisis de valores límite (BVA) es una técnica basada en el ejercicio de los límites de las particiones de equivalencia. Por lo tanto, BVA solo se puede usar para particiones ordenadas. Los valores mínimo y máximo de una partición son sus valores límite. En el caso de BVA, si dos elementos pertenecen a la misma partición, todos los elementos entre ellos también deben pertenecer a esa partición.

BVA se centra en los valores límite de las particiones porque los desarrolladores son más propensos a cometer errores con estos valores límite. Los defectos típicos encontrados por BVA se encuentran donde los límites implementados están fuera de lugar en posiciones por encima o por debajo de sus posiciones previstas o se omiten por completo.

Este programa de estudio cubre dos versiones del BVA: BVA de 2 valores y BVA de 3 valores.

Difieren en términos de elementos (o ítems) de cobertura por límite que deben ejercerse para lograr una cobertura del 100%.

En BVA de 2 valores (Craig 2002, Myers 2011), para cada valor límite hay dos elementos de cobertura: este valor límite y su vecino más cercano que pertenece a la partición adyacente. Para lograr una cobertura del 100% con BVA de 2 valores, los casos de prueba deben ejercer todos los elementos de cobertura, es decir, todos los valores límite identificados.

La cobertura se mide como el número de valores límite que se ejercieron, dividido por el número total de valores límite identificados, y se expresa como un porcentaje.

En BVA de 3 valores (Koomen 2006, O'Regan 2019), para cada valor límite hay tres elementos de cobertura: este valor límite y sus dos vecinos. Por lo tanto, en BVA de 3 valores algunos de los elementos de cobertura pueden no ser valores límite.
Para lograr una cobertura del 100% con BVA de 3 valores, los casos de prueba deben ejercer todos los elementos de cobertura, es decir, los valores límite identificados y sus vecinos. La cobertura se mide como el número de valores límite y sus vecinos ejercidos, dividido por el número total de valores límite identificados y sus vecinos, y se expresa como un porcentaje.

Pruebas de Tabla de Decisión
Las tablas de decisión se utilizan para probar la implementación de los requisitos del sistema que especifican cómo diferentes combinaciones de condiciones resultan en diferentes resultados. Las tablas de decisión son una forma efectiva de registrar lógica compleja, como las reglas de negocio.

Al crear tablas de decisión, se definen las condiciones y las acciones resultantes del sistema. Estos forman las filas de la tabla. Cada columna corresponde a una regla de decisión que define una combinación única de condiciones, junto con las acciones asociadas.
En las tablas de decisión de entrada limitada, todos los valores de las condiciones y acciones (excepto los irrelevantes o inviables; ver a continuación) se muestran como valores booleanos (verdaderos o falsos). Alternativamente, en las tablas de decisión de entrada extendida algunas o todas las condiciones y acciones también pueden tomar múltiples valores (por ejemplo, rangos de números, particiones de equivalencia, valores discretos).

La notación para las condiciones es la siguiente: "T" (verdadero) significa que la condición se cumple. "F" (falso) significa que la condición no se cumple. "–" significa que el valor de la condición es irrelevante para el resultado de la acción. "N/A" significa que la condición no es factible para una regla dada. Para las acciones: "X" significa que la acción debe ocurrir. En blanco significa que la acción no debe ocurrir.

También se pueden usar otras notaciones.
Una tabla de decisión completa tiene suficientes columnas para cubrir cada combinación de condiciones.

La tabla se puede simplificar eliminando columnas que contengan combinaciones de condiciones inviables. La tabla también se puede minimizar fusionando columnas, en las que algunas condiciones no afectan el resultado, en una sola columna. Los algoritmos de minimización de la tabla de decisiones están fuera del alcance de este programa de estudio.

En las pruebas de tabla de decisiones, los elementos de cobertura son las columnas que contienen

combinaciones factibles de condiciones. **Para lograr una cobertura del 100% con esta técnica, los casos de prueba deben ejercer todas estas columnas. La cobertura se mide como el número de columnas ejercitadas, dividido por el número total de columnas factibles, y se expresa como un porcentaje.**
La fortaleza de la prueba de tabla de decisiones es que proporciona un enfoque sistemático para identificar todas las combinaciones de condiciones, algunas de las cuales podrían pasarse por alto.

También ayuda a encontrar cualquier laguna o contradicción en los requisitos. Si hay muchas condiciones, el ejercicio de todas las reglas de decisión puede llevar mucho tiempo, ya que el número de reglas crece exponencialmente con el número de condiciones. En tal caso, para reducir el número de reglas que se deben ejercer, se puede utilizar una tabla de decisiones minimizada o un enfoque basado en el riesgo.

Pruebas de Transición de Estado

Un diagrama de transición de estado modela el comportamiento de un sistema mostrando sus posibles estados y transiciones de estado válidos. Una transición es iniciada por un evento, que puede ser calificado adicionalmente por una condición de guardia. Se supone que las transiciones son instantáneas y a veces pueden dar como resultado que el software tome medidas.

La sintaxis común de etiquetado de transición es la siguiente: "evento [condición de guardia] / acción". Las condiciones y acciones de protección pueden omitirse si no existen o son irrelevantes para el probador.

Una tabla de estado es un modelo equivalente a un diagrama de transición de estado. Sus filas representan estados, y sus columnas representan eventos (junto con las condiciones de guardia si existen). Las entradas de la tabla (las celdas) representan transiciones y contienen el estado meta, así como las acciones resultantes, si se definen. A diferencia del diagrama de transición de estado, la tabla de estado muestra explícitamente transiciones inválidas, que están representadas por celdas vacías.

Un caso de prueba basado en un diagrama de transición de estado o tabla de estados generalmente se representa como una secuencia de eventos, que da como resultado una secuencia de cambios de estado (y acciones, si es necesario). Un caso de prueba puede, y generalmente lo hará, cubrir varias transiciones entre estados.

Existen muchos criterios de cobertura para las pruebas de transición de estado. Este programa de estudio discute tres de ellos.

Cobertura de todos los estados, los elementos de cobertura son los estados. Para lograr el 100% de la cobertura de todos los estados, los casos de prueba deben garantizar que se visiten todos los estados. La cobertura se mide como el número de estados visitados dividido por el número total de estados, y se expresa como un porcentaje.

Cobertura de las transiciones válidas (también llamada cobertura de conmutador 0), los elementos de cobertura son transiciones válidas únicas. Para lograr una cobertura de transiciones 100% válidas, los casos de prueba deben ejercer todas las transiciones válidas. La cobertura se mide como el número de transiciones válidas ejercidas dividido por el número total de transiciones válidas, y se expresa como un porcentaje.

Cobertura de todas las transiciones, los elementos de cobertura son todas las transiciones mostradas en una tabla de estados. Para lograr el 100% de cobertura de todas las transiciones, los casos de prueba deben ejercer todas las transiciones válidas e intentar ejecutar transiciones inválidas.

Probar solo una transición inválida en un solo caso de prueba ayuda a evitar el enmascaramiento de defectos, es decir, una situación en la que un defecto impide la detección de otro. La cobertura se mide como el número de transiciones válidas e inválidas ejercidas o intentadas de ser cubiertas por casos de prueba ejecutados, dividido por el número total de transiciones válidas e inválidas, y se expresa como un porcentaje.

Toda cobertura de estados es más débil que la cobertura de transiciones válidas, porque normalmente se puede lograr sin ejercer todas las transiciones. **La cobertura de transiciones válidas es el criterio de cobertura más utilizado.**

Lograr una cobertura de transiciones válida completa garantiza una cobertura completa en todos los estados. Lograr una cobertura total de todas las transiciones garantiza tanto una cobertura total de todos los estados como una cobertura total de transiciones válidas y debería ser un requisito mínimo para el software de misión y de seguridad crítica.

4.3 Técnicas de Caja Blanca

Esta sección se centra en dos técnicas de prueba de caja blanca relacionadas con el código:
• Pruebas de sentencias
• Pruebas de ramas

En las **pruebas de sentencia**, los elementos de cobertura son sentencias ejecutables. **El objetivo es diseñar casos de prueba que ejerzan sentencias en el código hasta que se logre un nivel aceptable de cobertura. La cobertura se mide como el número de sentencias ejercidas por los casos de prueba dividido por el número total de sentencias ejecutables en el código, y se expresa como un porcentaje.**

Cuando se alcanza el 100% de cobertura de sentencias, se asegura que todas las sentencias ejecutables en el código se han ejercido al menos una vez. En particular, **esto significa que cada sentencia con un defecto se ejecutará, lo que puede causar una falla que demuestre la presencia del defecto.** Sin embargo, el ejercicio de una sentencia con un caso de prueba no detectará defectos

en todos los casos. Por ejemplo, puede no detectar defectos que dependen de los datos. Además, la cobertura del 100% de sentencias no garantiza que se haya probado toda la lógica de decisión.

Pruebas de Ramas y Cobertura de Ramas. Una rama es una transferencia de control entre dos nodos en el gráfico de flujo de control, que muestra las posibles secuencias en las que se ejecutan sentencias de código fuente en el objeto de prueba. Cada transferencia de control puede ser incondicional (es decir, código de línea recta) o condicional (es decir, un resultado de decisión).
En las pruebas de rama, los elementos de cobertura son ramas y el objetivo es diseñar casos de prueba para ejercer las ramas en el código hasta que se logre un nivel aceptable de cobertura. La cobertura se mide como el número de ramas ejercidas por los casos de prueba dividido por el número total de ramas, y se expresa como un porcentaje.

Cuando se alcanza el 100% de cobertura de ramas, todas las ramas en el código, incondicionales y condicionales, son ejercidas por casos de prueba. Las ramas condicionales normalmente corresponden a un resultado verdadero o falso de una decisión "if...then", un resultado de una sentencia de switch/case o una decisión de salir o continuar en un bucle. Sin embargo, el ejercicio de una rama con un caso de prueba no detectará defectos en todos los casos. Por ejemplo, puede que no detecte defectos que requieran la ejecución de una ruta (camino) específica en un código.
La cobertura de rama engloba la cobertura de sentencias. Esto significa que cualquier conjunto de casos de prueba que logre una cobertura de rama del 100% también logra una cobertura de sentencia del 100% (pero no viceversa).

Una fortaleza fundamental que comparten todas las técnicas de caja blanca es que toda la implementación del software se tiene en cuenta durante las pruebas, lo que facilita la detección de defectos incluso cuando la especificación del software es vaga, obsoleta o incompleta.
Una debilidad correspondiente es que si el software no implementa uno o más requisitos, las pruebas de caja blanca pueden no detectar los defectos de omisión resultantes (Watson 1996).

Las técnicas de caja blanca se pueden usar en pruebas estáticas (por ejemplo, durante ejecuciones secas- dry run- de código). Son muy adecuados para revisar código que aún no está listo para su ejecución (Hetzel 1988), así como pseudocódigo y otra lógica de alto nivel o de arriba hacia abajo que se puede modelar con un gráfico de flujo de control.

Realizar solo pruebas de caja negra no proporciona una medida de la cobertura de código real. Las medidas de cobertura de caja blanca proporcionan una medición objetiva de la cobertura y proporcionan la información necesaria para permitir que se generen pruebas adicionales para aumentar esta cobertura y, posteriormente, aumentar la confianza en el código.

4.4 Técnicas Basadas en la Experiencia

Las técnicas de prueba basadas en la experiencia comúnmente utilizadas son:
- Predicción de errores
- Pruebas exploratorias
- Pruebas basadas en listas de comprobación

La **predicción de errores** es una **técnica utilizada para anticipar la ocurrencia de errores, defectos y fallas, basada en el conocimiento del probador, que incluye:**
• Cómo funcionó la aplicación en el pasado
• Los tipos de errores que los desarrolladores tienden a cometer y los tipos de defectos que
resultan de estos errores
• Los tipos de fallas que se han producido en otras aplicaciones similares

En general, los errores, defectos y fallas pueden estar relacionados con: entrada (por ejemplo, entrada correcta no aceptada, parámetros incorrectos o faltantes), salida (por ejemplo, formato incorrecto,resultado incorrecto), lógica (por ejemplo, casos faltantes, operador incorrecto), cálculo (por ejemplo, operando incorrecto, cálculo incorrecto), interfaces (por ejemplo, desajuste de parámetros, tipos incompatibles) o datos (por ejemplo, inicialización incorrecta, tipo incorrecto).

Los ataques de faltas son un enfoque metódico para la implementación de la predicción de errores. Esta técnica requiere que el probador cree o adquiera una lista de posibles errores, defectos y fallas, y que diseñe pruebas que identifiquen los defectos asociados con los errores, expongan los defectos o causen las fallas. Estas listas se pueden construir en función de la experiencia, los datos de defectos y fallas, o del conocimiento común sobre por qué falla el software.

En las **pruebas exploratorias**, **las pruebas se diseñan, ejecutan y evalúan simultáneamente mientras el probador aprende sobre el objeto de prueba. Las pruebas se utilizan para aprender más sobre el objeto de prueba, para explorar más profundamente con pruebas enfocadas y para crear pruebas para áreas no probadas.**

Las pruebas exploratorias a veces se realizan utilizando pruebas basadas en sesiones para estructurar las pruebas. En un enfoque basado en sesiones, las pruebas exploratorias se llevan a cabo dentro de un intervalo de tiempo definido. El probador utiliza una carta de prueba que contiene los objetivos de la prueba para guiar la prueba. La sesión de prueba suele ir seguida de una sesión informativa que implica una discusión entre el probador y las partes interesadas en los resultados de la prueba. En este enfoque, los objetivos de la prueba pueden tratarse como condiciones de prueba de alto nivel. Los elementos de cobertura se identifican y ejercen durante la sesión de prueba. El probador puede usar hojas de sesión de prueba para documentar los pasos seguidos y los descubrimientos realizados.

Las pruebas exploratorias son útiles cuando hay pocas o inadecuadas especificaciones o hay una presión de tiempo significativa en las pruebas. Las pruebas exploratorias también son útiles para complementar otras técnicas de prueba más formales. Las pruebas exploratorias serán más efectivas si el probador tiene experiencia, tiene conocimiento del dominio y tiene un alto grado de habilidades esenciales, como habilidades analíticas, curiosidad y creatividad.

Las pruebas exploratorias pueden incorporar el uso de otras técnicas de prueba (por ejemplo, partición por equivalencia).

En las **pruebas basadas en listas de comprobación**, un probador diseña, implementa y ejecuta pruebas para cubrir las condiciones de prueba de una lista de comprobación. Las listas de comprobación se pueden construir en función de la experiencia, el conocimiento sobre lo que es importante para el usuario o la comprensión de por qué y cómo falla el software. Las listas de comprobación no deben contener elementos que se puedan comprobar automáticamente, elementos más adecuados como criterios de entrada/salida o elementos que son demasiado generales (Brykczynski 1999).

Los elementos de la lista de comprobación a menudo se formulan en forma de pregunta. Debe ser posible comprobar cada ítem por separado y directamente. Estos elementos pueden referirse a requisitos, propiedades de interfaz gráfica, características de calidad u otras formas de condiciones de prueba. Se pueden crear listas de comprobación para admitir varios tipos de pruebas, incluidas las pruebas funcionales y no funcionales (por ejemplo, 10 heurísticas para pruebas de usabilidad (Nielsen 1994)).

Algunas entradas de la lista de verificación pueden gradualmente volverse menos efectivas con el tiempo porque los desarrolladores aprenderán a evitar cometer los mismos errores. También es posible que se deban agregar nuevas entradas para reflejar los defectos de alta gravedad recientemente encontrados. Por lo tanto, las listas de comprobación deben actualizarse regularmente en función del análisis de defectos. Sin embargo, se debe tener cuidado para evitar que la lista de comprobación sea demasiado larga (Gawande 2009).

En ausencia de casos de prueba detallados, las pruebas basadas en listas de comprobación pueden proporcionar pautas y cierto grado de consistencia para las pruebas. Si las listas de comprobación son de alto nivel, es probable que ocurra alguna variabilidad en las pruebas reales, lo que resulta en una cobertura potencialmente mayor pero menos repetibilidad.

4.5. Enfoques de Prueba Basados en la Colaboración

Los enfoques basados en la colaboración, por otro lado, se centran también en evitar defectos mediante la colaboración y la comunicación.

Una historia de usuario representa una característica que será valiosa para un usuario o comprador de un sistema o software. Las historias de usuario tienen tres aspectos críticos (Jeffries 2000), llamados juntos las "3 C":
• **Tarjeta (Card):** el medio que describe una historia de usuario (por ejemplo, una tarjeta pequeña, una
entrada en un tablero electrónico)
• **Conversación:** explica cómo se utilizará el software (puede ser documentado o verbal)
• **Confirmación** – los criterios de aceptación (ver sección 4.5.2)
El **formato más común para una historia de usuario es**
"As [rol], I want that [objetivo se logre], so that [valor comercial resultante para el rol]", seguido de los criterios de aceptación.

La autoría colaborativa de la historia del usuario puede utilizar técnicas como la lluvia de ideas y mapeo mental. La colaboración permite al equipo obtener una visión compartida de lo que se debe entregar, teniendo en cuenta tres perspectivas: negocio, desarrollo y pruebas.
Las buenas historias de usuario deben ser: independientes, negociables, valiosas, estimables, pequeñas y comprobables (INVEST). Si una parte interesada no sabe cómo probar una historia de usuario, esto puede indicar que la historia de usuario no es lo suficientemente clara, o que no refleja algo valioso para ellos, o que la parte interesada precisamente necesita ayuda en las pruebas (Wake 2003).

Los **criterios de aceptación** para una historia de usuario son las condiciones que debe cumplir una implementación de la historia de usuario para ser aceptada por las partes interesadas. Desde esta perspectiva, los criterios de aceptación pueden ser vistos como las condiciones de prueba que deben ser ejercidas por las pruebas. Los criterios de aceptación generalmente son el resultado de la conversación

Los criterios de aceptación se utilizan para:
• Definir el alcance de la historia del usuario
• Alcanzar el consenso entre las partes interesadas
• Describir escenarios positivos y negativos
• Servir de base para las pruebas de aceptación de la historia del usuario (ver sección 4.5.3)
• Permitir una planificación y estimación precisas

Hay varias maneras de escribir criterios de aceptación para una historia de usuario. Los formatos más comunes son:
• Orientado a escenarios (por ejemplo, formato Given/When/Then utilizado en BDD)
• Orientado a reglas (por ejemplo, lista de verificación de viñetas o forma tabulada de mapeo de entrada-salida)

La mayoría de los criterios de aceptación se pueden documentar en uno de estos dos formatos. Sin embargo, el equipo puede usar otro formato personalizado, siempre que los criterios de aceptación estén bien definidos y sean inequívocos.

Desarrollo Guiado por Pruebas de Aceptación (ATDD) es un enfoque de prueba primero (ver sección 2.1.3). Los casos de prueba se crean antes de implementar la historia de usuario. Los casos de prueba son creados por miembros del equipo con diferentes perspectivas, por ejemplo, clientes, desarrolladores y probadores (Adzic 2009). Los casos de prueba pueden ejecutarse manualmente o automatizados.

El primer paso es un taller de especificación donde la historia del usuario y (si aún no está definida) sus criterios de aceptación son analizados, discutidos y escritos por los miembros del equipo. Incompletitud, ambigüedades o defectos en la historia del usuario se resuelven durante este proceso. El siguiente paso es crear los casos de prueba. Esto puede ser hecho por el equipo en su conjunto o por el probador individualmente. Los casos de prueba se basan en los criterios de aceptación y pueden verse como ejemplos de cómo funciona el software. Esto ayudará al equipo a implementar la historia del usuario correctamente.

Dado que los ejemplos y las pruebas son los mismos, estos términos a menudo se usan indistintamente.

Típicamente, los primeros casos de prueba son positivos, confirmando el comportamiento correcto sin excepciones o condiciones de error, y comprendiendo la secuencia de actividades ejecutadas si todo sale como se espera. Después de que se realicen los casos de prueba positivos, el equipo debe realizar pruebas negativas. Finalmente, el equipo también debe cubrir las características de calidad no funcionales (por ejemplo, eficiencia de rendimiento, usabilidad).
Los casos de prueba deben expresarse de una manera que sea comprensible para las partes interesadas. Por lo general, los casos de prueba contienen oraciones en lenguaje natural que involucran las condiciones previas necesarias (si las hay), las entradas y las condiciones posteriores.

Los casos de prueba deben cubrir todas las características de la historia del usuario y no deben ir más allá de la historia. Sin embargo, los criterios de aceptación pueden detallar algunos de los problemas descritos en la historia del usuario. Además, no debería haber dos casos de prueba que describan las mismas características de la historia del usuario.

Cuando se captura en un formato soportado por un marco de trabajo de automatización de pruebas, los desarrolladores pueden automatizar los casos de prueba escribiendo el código de soporte a medida que implementan la función descrita por una historia de usuario. Las pruebas de aceptación se convierten entonces en requisitos ejecutables.

• Capítulo 5: Gestión de las Actividades de Prueba (335 minutos)

- El estudiante aprende cómo planificar las pruebas en general y cómo estimar el esfuerzo de la prueba.
- El estudiante aprende cómo los riesgos pueden influir en el alcance de las pruebas.
- El estudiante aprende a monitorear y controlar las actividades de la prueba.
- El estudiante aprende cómo la gestión de la configuración admite las pruebas.
- El estudiante aprende a reportar defectos de una manera clara y comprensible.

• K1: Recordar
• K2: Comprender
• K3: Aplicar

Términos

gestión de defectos, informe del defecto, criterios de entrada, criterios de salida, riesgo de producto, riesgo de proyecto, riesgo, análisis de riesgos, evaluación de riesgos, control de riesgos, identificación de riesgos, nivel de riesgo, gestión de riesgos, mitigación de riesgos, monitoreo de riesgos, pruebas basadas en riesgos, enfoque de pruebas, informe de finalización de pruebas, control de pruebas, monitoreo de pruebas, plan de pruebas, planificación de pruebas, informe de progreso de pruebas, pirámide de pruebas, cuadrantes de pruebas

Objetivos de Aprendizaje para el Capítulo 5:

5.1 Planificación de Pruebas
FL-5.1.1 (K2) Ejemplificar el propósito y el contenido de un plan de prueba
FL-5.1.2 (K1) Reconocer cómo un probador agrega valor a la iteración y la planificación de la entrega
FL-5.1.3 (K2) Comparar y contrastar los criterios de entrada y los criterios de salida
FL-5.1.4 (K3) Utilizar técnicas de estimación para calcular el esfuerzo de prueba requerido
FL-5.1.5 (K3) Aplicar la priorización de casos de prueba
FL-5.1.6 (K1) Recordar los conceptos de la pirámide de prueba
FL-5.1.7 (K2) Resumir los cuadrantes de prueba y sus relaciones con los niveles de prueba y los tipos de prueba
5.2 Gestión de Riesgos
FL-5.2.1 (K1) Identificar el nivel de riesgo mediante el uso de la probabilidad de riesgo y el impacto del riesgo
FL-5.2.2 (K2) Distinguir entre los riesgos de proyecto y los riesgo de producto
FL-5.2.3 (K2) Explicar cómo el análisis de riesgo de producto puede influir en la rigurosidad y el alcance de las pruebas
FL-5.2.4 (K2) Explicar qué medidas se pueden tomar en respuesta a los riesgos de producto analizados
5.3 Monitoreo de Pruebas, Control de Pruebas y Finalización de Pruebas
FL-5.3.1 (K1) Recordar Métricas utilizadas para las pruebas
FL-5.3.2 (K2) Resumir los propósitos, el contenido y las audiencias de los informes de prueba
FL-5.3.3 (K2) Ejemplificar cómo comunicar el estado de las pruebas
5.4 Gestión de la Configuración
FL-5.4.1 (K2) Resumir cómo la gestión de configuración apoya las pruebas
5.5 Gestión de Defectos
FL-5.5.1 (K3) Preparar un informe de defecto

5.1 Planificación de Pruebas

Un plan de prueba describe los objetivos, recursos y procesos para un proyecto de prueba. Un plan de prueba:

- Documenta los medios y el cronograma para lograr los objetivos de la prueba
- Ayuda a garantizar que las actividades de prueba realizadas cumplan con los criterios establecidos
- Sirve como medio de comunicación con los miembros del equipo y otras partes interesadas
- Demuestra que las pruebas se adherirán a la política de pruebas y la estrategia de pruebas existentes (o explica por qué la prueba se desviará de ellas)

La planificación de pruebas guía el pensamiento de los probadores y obliga a los probadores a enfrentar los desafíos futuros relacionados con los riesgos, los horarios, las personas, las herramientas, los costos, el esfuerzo, etc. El proceso de preparación de un plan de prueba es una forma útil de pensar en los esfuerzos necesarios para lograr los objetivos del proyecto de prueba.
El contenido típico de un plan de prueba incluye:

- Contexto de las pruebas (por ejemplo, alcance, objetivos de la prueba, limitaciones, base de la prueba)

• Supuestos y limitaciones del proyecto de prueba
• Partes interesadas (por ejemplo, roles, responsabilidades, relevancia para las pruebas, contratación y necesidades de capacitación)
• Comunicación (por ejemplo, formularios y frecuencia de comunicación, plantillas de documentación)
• Registro de riesgos (por ejemplo, riesgos de producto, riesgos de proyecto)
• Enfoque de prueba (por ejemplo, niveles de prueba, tipos de prueba, técnicas de prueba, resultados de prueba, criterios de entrada y criterios de salida, independencia de las pruebas, métricas que se recopilarán, requisitos de datos de prueba, requisitos del entorno de prueba, desviaciones de la política de pruebas y estrategia de prueba de la organización)
• Presupuesto y cronograma

En los SDLC iterativos, generalmente ocurren dos tipos de planificación: planificación de entregas y planificación de iteraciones.
La planificación de entregas anticipa la entrega de un producto, define y redefine la pila (el backlog) del producto, y puede implicar refinar las historias de usuarios más grandes en un conjunto de historias de usuarios más pequeñas. También sirve como base para el enfoque de prueba y el plan de prueba en todas las iteraciones. **Los probadores involucrados en la planificación de la versión participan en la redacción de historias de usuario comprobables y criterios de aceptación**, **participan en análisis de riesgos de calidad y proyectos, estiman el esfuerzo de prueba asociado con las historias de usuario, determinan el enfoque de prueba y planifican las pruebas para la versión.**

La **planificación de la iteración** mira hacia el final de una sola iteración y se ocupa de la pila (el backlog) de la iteración. **Los probadores involucrados en la planificación de la iteración participan en el análisis detallado de riesgos de las historias de usuario, determinan la comprobabilidad de las historias de usuario, dividen las historias de usuario en tareas (particularmente las tareas de prueba), estiman el esfuerzo de prueba para todas las tareas de prueba e identifican y refinan aspectos funcionales y no funcionales del objeto de prueba.**

Los **criterios de entrada** definen las condiciones previas para emprender una actividad determinada. Si no se cumplen los criterios de ingreso, es probable que la actividad resulte ser más difícil, lenta, costosa y arriesgada.
Los **criterios de salida** definen lo que se debe lograr para declarar una actividad completada. Los criterios de entrada y los criterios de salida deben definirse para cada nivel de prueba y diferirán según los objetivos de la prueba.

Los criterios de entrada típicos incluyen: disponibilidad de recursos (por ejemplo, personas, herramientas, entornos, datos de prueba, presupuesto, tiempo), disponibilidad de testware (por ejemplo, base de prueba, requisitos comprobables, historias de usuario, casos de prueba) y nivel de calidad inicial de un objeto de prueba (por ejemplo, todas las pruebas de humo han pasado).

Los criterios de salida típicos incluyen: medidas de rigurosidad (por ejemplo, nivel de cobertura

alcanzado, número de defectos no resueltos, densidad de defectos, número de casos de prueba fallidos) y criterios de finalización (por ejemplo, se han ejecutado las pruebas planificadas, se han realizado las pruebas estáticas, se informan todos los defectos encontrados, todas las pruebas de regresión están automatizadas).
Quedarse sin tiempo o presupuesto también puede ser visto como un criterio de salida válido. Incluso sin que se cumplan otros criterios de salida, puede ser aceptable finalizar las pruebas en tales circunstancias, si las partes interesadas han revisado y aceptado el riesgo de entrar en funcionamiento sin más pruebas.
En el desarrollo de software Ágil, los criterios de salida a menudo se llaman Definición de Hecho/Terminado, definiendo las métricas objetivas del equipo para un elemento liberable. Los criterios de entrada que una historia de usuario debe cumplir para comenzar las actividades de desarrollo y/o prueba se llaman Definición de Preparado/Listo.

La **estimación del esfuerzo de prueba** implica predecir la cantidad de trabajo relacionado con la prueba necesario para cumplir con los objetivos de un proyecto de prueba. Es importante dejar claro a las partes interesadas que la estimación se basa en una serie de supuestos y siempre está sujeta a errores de estimación. La estimación para tareas pequeñas suele ser más precisa que para las grandes. Por lo tanto, al estimar una tarea grande, se puede descomponer en un conjunto de tareas más pequeñas que luego, a su vez, se pueden estimar.

En este programa de estudio, se describen las siguientes cuatro técnicas de estimación.

Estimación basada en proporciones. En esta técnica basada en métricas, se recopilan cifras de proyectos anteriores dentro de la organización, lo que permite derivar proporciones "estándar" para proyectos similares. Las proporciones de los propios proyectos de una organización (por ejemplo, tomadas de datos históricos) son generalmente la mejor fuente para usar en el proceso de estimación.
Estas relaciones estándar se pueden usar para estimar el esfuerzo de prueba para el nuevo proyecto. Por ejemplo, si en el proyecto anterior la relación entre el desarrollo y el esfuerzo de prueba fue de 3:2, y en el proyecto actual se espera que el esfuerzo de desarrollo sea de 600 días-persona, el esfuerzo de prueba se puede estimar en 400 días-persona.

Extrapolación. En esta técnica basada en métricas, las mediciones se realizan lo antes posible en el proyecto actual para recopilar los datos. Al tener suficientes observaciones, el esfuerzo requerido para el trabajo restante se puede aproximar extrapolando estos datos (generalmente aplicando un modelo matemático). Este método es muy adecuado en los Ciclos de Vida de Desarrollo de Software (SDLC) iterativos. Por ejemplo, el equipo puede extrapolar el esfuerzo de prueba en la próxima iteración como el esfuerzo promediado de las últimas tres iteraciones.

Delphi de banda ancha. En esta técnica iterativa basada en expertos, los expertos hacen estimaciones basadas en la experiencia. Cada experto, de forma aislada, estima el esfuerzo. Los resultados se recogen y si hay desviaciones que están fuera del rango de los límites acordados, los expertos discuten sus estimaciones actuales. Luego se le pide a cada experto que haga una nueva

estimación basada en esa retroalimentación, nuevamente de forma aislada. Este proceso se repite hasta que se alcanza un consenso. **La Planificación Poker es una variante de Wideband Delphi, comúnmente utilizada en el desarrollo de software Ágil. En la Planificación Poker, las estimaciones generalmente se hacen usando tarjetas con números que representan el tamaño del esfuerzo.**

Estimación de tres puntos. En esta técnica basada en expertos, los expertos realizan tres estimaciones: la estimación más optimista (a), la estimación más probable (m) y la estimación más pesimista (b). La estimación final (E) es su media aritmética ponderada. En la versión más popular de esta técnica, la estimación se calcula como E = (a + 4*m + b) / 6. La ventaja de esta técnica es que permite a los expertos calcular el error de medición: SD = (b – a) / 6. Por ejemplo, si las estimaciones (en horas-persona) son: a =6, m=9 y b=18, entonces la estimación final es 10±2 horas-persona (es decir, entre 8 y 12 horas-persona), porque E = (6 + 4*9 + 18) / 6 = 10 y SD = (18 – 6) / 6 = 2.

Las estrategias de **priorización de casos de prueba** más utilizadas son las siguientes:
• **Priorización basada en el riesgo**, donde el orden de ejecución de la prueba se basa en los resultados del análisis de riesgos. Los casos de prueba que cubren los riesgos más importantes se ejecutan primero.
• **Priorización basada en la cobertura**, donde el orden de ejecución de la prueba se basa en la cobertura (por ejemplo, la cobertura de sentencia). Los casos de prueba que logran la cobertura más alta se ejecutan primero. En otra variante, llamada priorización de cobertura adicional, el caso de prueba que logra la cobertura más alta se ejecuta primero; cada caso de prueba posterior es el que logra la cobertura adicional más alta.
• **Priorización basada en requisitos**, donde el orden de ejecución de la prueba se basa en las prioridades de los requisitos rastreados hasta los casos de prueba correspondientes. Las prioridades de los requisitos son definidas por las partes interesadas. Los casos de prueba relacionados con los requisitos más importantes se ejecutan primero.
Idealmente, los casos de prueba se ordenarían para ejecutarse en función de sus niveles de prioridad, utilizando, por ejemplo, una de las estrategias de priorización mencionadas anteriormente. Sin embargo, esta práctica puede no funcionar si los casos de prueba o las características que se están probando tienen dependencias. Si un caso de prueba con una prioridad más alta depende de un caso de prueba con una prioridad más baja, el caso de prueba de prioridad más baja debe ejecutarse primero.
El orden de ejecución de la prueba también debe tener en cuenta la disponibilidad de recursos. Por ejemplo, las herramientas de prueba requeridas, los entornos de prueba o las personas que solo pueden estar disponibles para una ventana de tiempo específica.

La **pirámide de prueba** es un modelo que muestra que diferentes pruebas pueden tener una granularidad diferente. El modelo de pirámide de prueba apoya al equipo en la automatización de pruebas y en la asignación del esfuerzo de prueba al mostrar que los diferentes objetivos están respaldados por diferentes niveles de automatización de pruebas. Las capas piramidales representan grupos de pruebas.

Cuanto más alta sea la capa, menor será la granularidad de la prueba, el aislamiento de la prueba y el tiempo de ejecución de la prueba. Las pruebas en la capa inferior son pequeñas, aisladas, rápidas y comprueban una pequeña pieza de funcionalidad, por lo que generalmente se necesitan muchas de ellas para lograr una cobertura razonable. La capa superior representa pruebas complejas, de alto nivel y de extremo a extremo. Estas pruebas de alto nivel son generalmente más lentas que las pruebas de las capas inferiores, y generalmente comprueban una gran pieza de funcionalidad, por lo que generalmente solo se necesitan algunas de ellas para lograr una cobertura razonable. El número y el nombre de las capas pueden diferir. Por ejemplo, el modelo de pirámide de prueba original (Cohn 2009) define tres capas: “pruebas unitarias”, “pruebas de servicio” y “pruebas de interfaz de usuario”. Otro modelo popular define las pruebas de unidad (componentes), pruebas de integración (integración de componentes) y pruebas de extremo a extremo.

Los **cuadrantes de prueba**, definidos por Brian Marick (Marick 2003, Crispin 2008), agrupan los niveles de prueba con los tipos de prueba, actividades, técnicas de prueba y productos de trabajo apropiados en el desarrollo de software Ágil.
El modelo apoya la gestión de pruebas en la visualización de estos para garantizar que todos los tipos de prueba y niveles de prueba apropiados estén incluidos en el Ciclo de Vida de Desarrollo de Software (SDLC) y en la comprensión de que algunos tipos de prueba son más relevantes para ciertos niveles de prueba que otros. Este modelo también proporciona una forma de diferenciar y describir los tipos de pruebas a todas las partes interesadas, incluidos desarrolladores, probadores y representantes de negocio.
En este modelo, las pruebas pueden estar orientadas al negocio o a la tecnología. Las pruebas también pueden apoyar al equipo (es decir, guiar el desarrollo) o criticar el producto (es decir, medir su comportamiento contra las expectativas). La combinación de estos dos puntos de vista determina los cuatro cuadrantes:
• **Cuadrante Q1 (de cara a la tecnología, apoyo al equipo)**. **Este cuadrante contiene pruebas de integración de componentes y componentes.** Estas pruebas deben automatizarse e incluirse en el proceso de Integración Continua (IC).
• **Cuadrante Q2 (de cara al negocio, apoyo al equipo). Este cuadrante contiene pruebas funcionales, ejemplos, pruebas de historias de usuario, prototipos de experiencia de usuario, pruebas de API y simulaciones.** Estas pruebas comprueban los criterios de aceptación y pueden ser manuales o automatizadas.
• **Cuadrante Q3 (de cara al negocio, critica el producto). Este cuadrante contiene pruebas exploratorias, pruebas de usabilidad, pruebas de aceptación del usuario.** Estas pruebas están orientadas al usuario y, a menudo, son manuales.
• **Cuadrante Q4 (de cara a la tecnología, critica el producto). Este cuadrante contiene pruebas de humo y pruebas no funcionales (excepto pruebas de usabilidad).** Estas pruebas a menudo son automatizadas.

5.2 Gestión de Riesgos

La gestión de riesgos permite a las organizaciones aumentar la probabilidad de alcanzar objetivos, mejorar la calidad de sus productos y aumentar la confianza de las partes interesadas.

Las principales actividades de gestión de riesgos son:
• Análisis de riesgos (que consiste en la identificación y evaluación de riesgos;)
• Control de riesgos (consistente en la mitigación de riesgos y monitoreo de riesgos)
El enfoque de prueba, en el que las actividades de prueba se seleccionan, priorizan y gestionan en función del análisis de riesgos y el control de riesgos, se denomina prueba basada en el riesgo.

El riesgo es un evento potencial, peligro, amenaza o situación cuya ocurrencia causa un efecto adverso.
Un riesgo se puede caracterizar por dos factores:
• Probabilidad del riesgo: la probabilidad de que ocurra el riesgo (mayor que cero y menor que uno)
• Impacto del riesgo (daño) – las consecuencias de este hecho
Estos dos factores expresan el nivel de riesgo, que es una medida del riesgo. Cuanto mayor es el nivel de riesgo, más importante es su tratamiento.

En las pruebas de software, uno generalmente se preocupa por dos tipos de riesgos: los riesgos de proyecto y los riesgos de producto.
Los **riesgos de proyecto** están relacionados con la gestión y el control del proyecto. Los riesgos de proyecto incluyen:
• Cuestiones de organización (por ejemplo, retrasos en las entregas de productos de trabajo, estimaciones inexactas, reducción de costos)
• Problemas con las personas (por ejemplo, habilidades insuficientes, conflictos, problemas de comunicación, escasez de personal)
• Problemas técnicos (por ejemplo, expansión del alcance (scope creep), soporte deficiente de la herramienta)
• Problemas con el proveedor (por ejemplo, falla en la entrega de terceros, quiebra de la empresa de soporte)
Los riesgos de proyecto, cuando ocurren, pueden tener un impacto en el cronograma, presupuesto o alcance del proyecto, lo que afecta la capacidad del proyecto para lograr sus objetivos.

Los **riesgos de producto** están relacionados con las características de calidad del producto. Ejemplos de riesgos de producto incluyen:
funcionalidad faltante o incorrecta, cálculos incorrectos, errores de tiempo de ejecución, arquitectura deficiente, algoritmos ineficientes, tiempo de respuesta inadecuado, experiencia de usuario deficiente, vulnerabilidades de seguridad. Los riesgos de producto, cuando ocurren, pueden resultar en varias consecuencias negativas, incluyendo:
• Insatisfacción del usuario
• Pérdida de ingresos, confianza, reputación
• Daños a terceros
• Altos costos de mantenimiento, sobrecarga del servicio de asistencia
• Sanciones penales
• En casos extremos, daños físicos, lesiones o incluso la muerte

El **objetivo del análisis de riesgo de producto** es proporcionar una conciencia del riesgo de producto para enfocar el esfuerzo de prueba de una manera que minimice el nivel residual de riesgo de producto. Idealmente, el análisis del riesgo de producto comienza temprano en el Ciclo de Vida de Desarrollo de Software (SDLC).

El análisis de riesgos de producto consiste en la identificación de los riesgos y la evaluación de los riesgos. La identificación de riesgos consiste en generar una lista completa de riesgos. Las partes interesadas pueden identificar los riesgos mediante el uso de diversas técnicas y herramientas, por ejemplo, lluvia de ideas, talleres, entrevistas o diagramas de causa-efecto. La evaluación de riesgos implica: categorizar los riesgos identificados, determinar su probabilidad de riesgo, impacto y nivel de riesgo, priorizar y proponer formas de manejarlos. La categorización ayuda a asignar acciones de mitigación, porque generalmente los riesgos que caen en la misma categoría se pueden mitigar utilizando un enfoque similar.

La evaluación de riesgos puede utilizar un enfoque cuantitativo o cualitativo, o una combinación de ellos.

En el enfoque cuantitativo, el nivel de riesgo se calcula como la multiplicación de la probabilidad de riesgo y el impacto del riesgo. En el enfoque cualitativo, el nivel de riesgo se puede determinar utilizando una matriz de riesgo.

El análisis de riesgos de producto puede influir en la rigurosidad y el alcance de las pruebas. Sus resultados se utilizan para:

- **Determinar el alcance de las pruebas a realizar**
- **Determinar los niveles de prueba particulares y proponer los tipos de prueba que se realizarán**
- **Determinar las técnicas de prueba que se emplearán y la cobertura que se logrará**
- **Estimar el esfuerzo de prueba requerido para cada tarea**
- **Priorizar las pruebas en un intento de encontrar los defectos críticos tan pronto como sea posible**
- **Determinar si se podría emplear alguna actividad además de las pruebas para reducir el riesgo**

El control de riesgos de producto consiste en la mitigación del riesgo y el monitoreo del riesgo. La mitigación de riesgos implica implementar las acciones propuestas en la evaluación de riesgos para reducir el nivel de riesgo. El objetivo del monitoreo de riesgos es garantizar que las acciones de mitigación sean efectivas, obtener más información para mejorar la evaluación de riesgos e identificar riesgos emergentes.

Con respecto al control de riesgos de producto, una vez que se ha analizado un riesgo, son posibles varias opciones de respuesta al riesgo, por ejemplo, mitigación del riesgo mediante pruebas, aceptación del riesgo, transferencia del riesgo o plan de contingencia (Veenendaal 2012). **Las acciones que se pueden tomar para mitigar los riesgos del producto mediante pruebas son las siguientes:**

- **Seleccionar los probadores con el nivel adecuado de experiencia y habilidades, adecuados para un**

tipo de riesgo determinado

- **Aplicar un nivel adecuado de independencia de las pruebas**
- **Realizar revisiones y realizar análisis estáticos**
- **Aplicar las técnicas de prueba y los niveles de cobertura adecuados**
- **Aplicar los tipos de prueba apropiados que aborden las características de calidad afectadas**
- **Realizar pruebas dinámicas, incluidas las pruebas de regresión**

5.3 Monitoreo de Pruebas, Control de Pruebas y Finalización de Pruebas

El monitoreo de pruebas se refiere a la recopilación de información sobre las pruebas. Esta información se utiliza para evaluar el progreso de la prueba y para medir si se cumplen los criterios de salida de la prueba o las tareas de prueba asociadas con los criterios de salida, como el cumplimiento de los objetivos de cobertura de los riesgos de producto, los requisitos o los criterios de aceptación.
El control de pruebas utiliza la información del monitoreo de pruebas para proporcionar, en forma de directivas de control, orientación y las acciones correctivas necesarias para lograr las pruebas más efectivas y eficientes. Ejemplos de directivas de control incluyen:
• Repriorizar las pruebas cuando un riesgo identificado se convierte en un problema
• Volver a evaluar si un elemento de prueba cumple con los criterios de entrada o los criterios de salida debido a la reelaboración
• Ajustar del cronograma de prueba para abordar un retraso en la entrega del entorno de prueba
• Agregar nuevos recursos cuando y donde sea necesario
La finalización de la prueba recopila datos de las actividades de prueba completadas para consolidar la experiencia, el software de prueba y cualquier otra información relevante. Las actividades de finalización de pruebas ocurren en los hitos del proyecto, como cuando se completa un nivel de prueba, se termina una iteración Ágil, se completa (o cancela) un proyecto de prueba, se entrega un sistema de software o se completa una entrega de mantenimiento.

Las **métricas de prueba** se recopilan para mostrar el progreso en relación con el cronograma y el presupuesto planificados, la calidad actual del objeto de prueba y la efectividad de las actividades de prueba con respecto a los objetivos o una meta de la iteración. El monitoreo de pruebas reúne una variedad de métricas para respaldar el control de la prueba y la finalización de la prueba.
Las métricas comunes incluyen:
• Métricas de progreso del proyecto (por ejemplo, finalización de tareas, uso de recursos, esfuerzo de prueba)
• Métricas de progreso de las pruebas (por ejemplo, progreso de la implementación de casos de prueba, progreso de la preparación del entorno de prueba, número de casos de prueba ejecutados/no ejecutados, pasados/fallidos, tiempo de ejecución de las pruebas)
• Métricas de calidad del producto (por ejemplo, disponibilidad, tiempo de respuesta, tiempo promedio hasta la falla)
• Métricas de defectos (por ejemplo, número y prioridades de defectos encontrados/corregidos, densidad de defectos, porcentaje de detección de defectos)
• Métricas de riesgo (por ejemplo, nivel de riesgo residual)
• Métricas de cobertura (por ejemplo, cobertura de requisitos, cobertura de código)
• Métricas de costos (por ejemplo, costo de las pruebas, costo de la calidad en la organización)

Los **informes de prueba** resumen y comunican la información de la prueba durante y después de la prueba. Los informes de progreso de la prueba respaldan el control continuo de la prueba y deben proporcionar suficiente información para realizar modificaciones en el

cronograma de la prueba, los recursos o el plan de prueba, cuando dichos cambios sean necesarios debido a una desviación del plan o circunstancias cambiadas. Los informes de finalización de la prueba resumen una etapa específica de la prueba (por ejemplo, nivel de prueba, ciclo de prueba, iteración) y pueden proporcionar información para pruebas posteriores.
Durante el monitoreo y control de la prueba, el equipo de prueba genera informes de progreso de la prueba para que las partes interesadas los mantengan informados. **Los informes de progreso de la prueba generalmente se generan de forma regular (por ejemplo, diariamente, semanalmente, etc.) e incluyen:**
• Duración de prueba
• Progreso de la prueba (por ejemplo, adelantar o retrasar el cronograma), incluidas las desviaciones
notables
• Impedimentos para las pruebas y sus soluciones alternativas
• Métricas de prueba (consulte la sección 5.3.1 para ver ejemplos)
• Riesgos nuevos y modificados dentro del período de prueba
• Pruebas previstas para el próximo período
Un informe de finalización de la prueba se prepara durante la finalización de la prueba, cuando un proyecto, nivel de prueba o tipo de prueba está completo y cuando, idealmente, se han cumplido sus criterios de salida. Este informe utiliza informes de progreso de la prueba y otros datos. Los informes típicos de finalización de la prueba incluyen:
• Resumen de la prueba
• Evaluación de la prueba y la calidad del producto según el plan de pruebas original (es decir, objetivos de prueba y criterios de salida)
• Desviaciones del plan de prueba (por ejemplo, diferencias con el cronograma, la duración y el esfuerzo planificados).
• Impedimentos de prueba y soluciones alternativas
• Métricas de prueba basadas en informes de progreso de la prueba
• Riesgos no mitigados, defectos no corregidos
• Lecciones aprendidas que son relevantes para la prueba
Diferentes audiencias requieren información diferente en los informes e influyen en el grado de formalidad y la frecuencia de los informes. Informar sobre el progreso de las pruebas a otros en el mismo equipo a menudo es frecuente e informal, mientras que informar sobre las pruebas para un proyecto completado sigue una plantilla establecida y ocurre solo una vez.

Las opciones para comunicar el estado de las pruebas incluyen:
• Comunicación verbal con los miembros del equipo y otras partes interesadas
• Tableros de control (por ejemplo, tableros de Integración Continua o Entrega Continua (CI/CD), tableros de tareas y diagramas de trabajo restante)
• Canales de comunicación electrónica (por ejemplo, correo electrónico, chat)
• Documentación en línea
• Informes formales de prueba.

5.4 Gestión de la Configuración

Para respaldar adecuadamente las pruebas, la Gestión de la Configuración (CM) garantiza lo siguiente:

• Todos los elementos de configuración, incluidos los elementos de prueba (partes individuales del objeto de prueba), se identifican de forma única, se controlan las versiones, se realiza un seguimiento de los cambios y se relacionan con otros elementos de configuración para que se pueda mantener la trazabilidad durante todo el proceso de prueba.

• Toda la documentación identificada y los elementos de software se mencionan inequívocamente en la documentación de prueba

La integración continua, la entrega continua, el despliegue continuo y las pruebas asociadas generalmente se implementan como parte de un canal (pipeline) automatizado de DevOps (consulte la sección 2.1.4), en la que normalmente se incluye la Gestión de la Configuración (CM) automatizada.

5.5 Gestión de Defectos

el proceso de gestión de defectos incluye un flujo de trabajo para manejar anomalías individuales desde su descubrimiento hasta su cierre y reglas para su clasificación. El flujo de trabajo generalmente comprende actividades para registrar las anomalías informadas, analizarlas y clasificarlas, decidir una respuesta adecuada, como arreglarla o mantenerla como está y finalmente cerrar el informe de defecto. El proceso debe ser seguido por todas las partes interesadas involucradas.

Es aconsejable manejar los defectos de las pruebas estáticas (especialmente el análisis estático) de manera similar.

Los informes de defecto típicos tienen los siguientes objetivos:

• Proporcionar información suficiente para resolver el problema a los responsables de manejar y resolver los defectos reportados

• Proporcionar un medio de seguimiento de la calidad del producto de trabajo

• Proporcionar ideas para mejorar el proceso de desarrollo y prueba

Un informe de defecto registrado durante las pruebas dinámicas generalmente incluye:

• Identificador único

• Título con un breve resumen de la anomalía que se informa

• Fecha en que se observó la anomalía, organización emisora y autor, incluido su rol

• Identificación del objeto de prueba y el entorno de prueba

• Contexto del defecto (por ejemplo, caso de prueba que se está ejecutando, actividad de prueba que se está realizando, fase del Ciclo de Vida de Desarrollo de Software (SDLC) y otra información relevante, como la técnica de prueba, la lista de comprobación o los datos de prueba que se están utilizando)

• Descripción de la falla para habilitar la reproducción y la resolución, incluidos los pasos que detectaron la anomalía, y cualquier registro de prueba, copias de seguridad de la base de datos, capturas de pantalla o grabaciones relevantes

• Resultados esperados y resultados reales

• Gravedad del defecto (grado del impacto) en los intereses de las partes interesadas o requisitos Prioridad para corregir

• Estado del defecto (por ejemplo, abierto, diferido, duplicado, en espera de ser corregido, en

espera de pruebas de confirmación, reabierto, cerrado, rechazado)
• Referencias (por ejemplo, al caso de prueba)

• Capítulo 6: Herramientas de Prueba (20 minutos)

o El estudiante aprende a clasificar herramientas y a comprender los riesgos y beneficios de la automatización de pruebas.
• K1: Recordar
• K2: Comprender
• K3: Aplicar

Términos
Automatización de pruebas

Objetivos de Aprendizaje para el Capítulo 6

6.1 Soporte de Herramientas para las Pruebas
FL-6.1.1 (K2) Explicar cómo los diferentes tipos de herramientas de prueba soportan las pruebas
6.2 Beneficios y Riesgos de la Automatización de Pruebas
FL-6.2.1 (K1) Recordar los beneficios y riesgos de la automatización de pruebas

6.1 Soporte de Herramientas para las Pruebas

Las herramientas de prueba apoyan y facilitan muchas actividades de prueba. Los ejemplos incluyen, pero no se limitan a:
• Herramientas de gestión: aumentan la eficiencia del proceso de prueba al facilitar la gestión del Ciclo de Vida del Desarrollo de Software (SDLC), los requisitos, las pruebas, los defectos y la configuración
• Herramientas de pruebas estáticas: apoyan al probador en la realización de revisiones y análisis estáticos
• Herramientas de diseño e implementación de pruebas: facilitan la generación de casos de prueba, datos de prueba y procedimientos de prueba
• Herramientas de ejecución y cobertura de pruebas: facilitan la ejecución automatizada de pruebas y la medición de la cobertura
• Herramientas de prueba no funcionales: permiten al probador realizar pruebas no funcionales que son difíciles o imposibles de realizar manualmente.
• Herramientas DevOps: admiten la canalización de entrega de DevOps, el seguimiento del flujo de trabajo, los procesos de compilación automatizados, Integración Continua/Entrega Continua (CI/CD)
• Herramientas de colaboración – facilitan la comunicación
• Herramientas que admiten escalabilidad y estandarización de implementación (por ejemplo, máquinas virtuales, herramientas de contenedorización)
• Cualquier otra herramienta que ayude en las pruebas (por ejemplo, una hoja de cálculo es una herramienta de prueba en el contexto de las pruebas)

6.2 Beneficios y Riesgos de la Automatización de Pruebas

Los beneficios potenciales del uso de la automatización de pruebas incluyen:

• Tiempo ahorrado al reducir el trabajo manual repetitivo (por ejemplo, ejecutar pruebas de regresión, volver a ingresar los mismos datos de prueba, comparar los resultados esperados con los resultados reales y comprobar los estándares de codificación)
• Prevención de errores humanos simples a través de una mayor consistencia y repetibilidad (por ejemplo, las pruebas se derivan consistentemente de los requisitos, los datos de prueba se crean de manera sistemática y las pruebas se ejecutan mediante una herramienta en el mismo orden con la misma frecuencia)
• Evaluación más objetiva (por ejemplo, cobertura) y proporcionar medidas que son demasiado complicadas para que los humanos las obtengan
• Acceso más fácil a la información sobre las pruebas para respaldar la gestión de pruebas y los informes de pruebas (por ejemplo, estadísticas, gráficos y datos agregados sobre el progreso de la prueba, las tasas de defectos y la duración de la ejecución de la prueba)
• Reducción de los tiempos de ejecución de pruebas para proporcionar una detección más temprana de defectos, una retroalimentación más rápida y un tiempo de comercialización más rápido
• Más tiempo para que los probadores diseñen pruebas nuevas, más profundas y más efectivas
Los riesgos potenciales de usar la automatización de pruebas incluyen:
• Expectativas poco realistas sobre los beneficios de una herramienta (incluida la funcionalidad y la facilidad de uso).
• Estimaciones inexactas de tiempo, costos, esfuerzo requerido para introducir una herramienta, mantener los scripts de prueba y cambiar el proceso de prueba manual existente.
• Usar una herramienta de prueba cuando la prueba manual es más apropiada.
• Confiar demasiado en una herramienta, por ejemplo, ignorar la necesidad del pensamiento crítico humano.
• La dependencia del proveedor de la herramienta que puede salir del negocio, retirar la herramienta, vender la herramienta a un proveedor diferente o proporcionar un soporte deficiente (por ejemplo, respuestas a consultas, actualizaciones y correcciones de defectos).
• El uso de un software de código abierto que puede ser abandonado, lo que significa que no hay más actualizaciones disponibles, o sus componentes internos pueden requerir actualizaciones bastante frecuentes como un desarrollo adicional.
• La herramienta de automatización no es compatible con la plataforma de desarrollo.
• Elegir una herramienta inadecuada que no cumpla con los requisitos reglamentarios y/o estándares de seguridad.