Unidad 2

Prueba a lo Largo del Ciclo de Vida de Desarrollo de Software

1. Modelos de Ciclo de Vida del Desarrollo de Software

Un modelo de ciclo de vida de desarrollo de software describe los tipos de actividad que se realizan encada etapa de un proyecto de desarrollo de software, y cómo las actividades se relacionan entre sí de forma lógica y cronológica. Hay diferentes modelos de ciclo de vida de desarrollo de software, cada unode los cuales requiere diferentes enfoques de prueba.

1.1 Desarrollo de Software y Prueba de Software

Una parte importante del rol de probador es estar familiarizado con los modelos de ciclo de vida de desarrollo de software más comunes, de modo que se puedan realizar las actividades de prueba adecuadas.

En cualquier modelo de ciclo de vida de desarrollo de software hay una serie de características que hacen que las pruebas sean adecuadas:

- **Para cada actividad de desarrollo, hay una actividad de prueba asociada.**
- **Cada nivel de prueba tiene objetivos de prueba específicos para ese nivel.**
- **El análisis y diseño de la prueba para un nivel de prueba dado comienza durante la actividad dedesarrollo correspondiente**.
- **Los probadores participan en discusiones para definir y refinar los requisitos y el diseño, y están involucrados en la revisión de los productos de trabajo (por ejemplo, requisitos, diseño, historias de usuario, etc.) tan pronto como las versiones borrador estén disponibles.**

Independientemente del modelo de ciclo de vida de desarrollo de software que se haya elegido, las actividades de prueba deben comenzar en las etapas iniciales del ciclo de vida, adhiriéndose al principiode “prueba temprana”.

Este programa de estudio clasifica los modelos de ciclo de vida de desarrollo de software comunes de lasiguiente manera:

- Modelos de desarrollo secuencial.
- Modelos de desarrollo iterativos e incrementales.

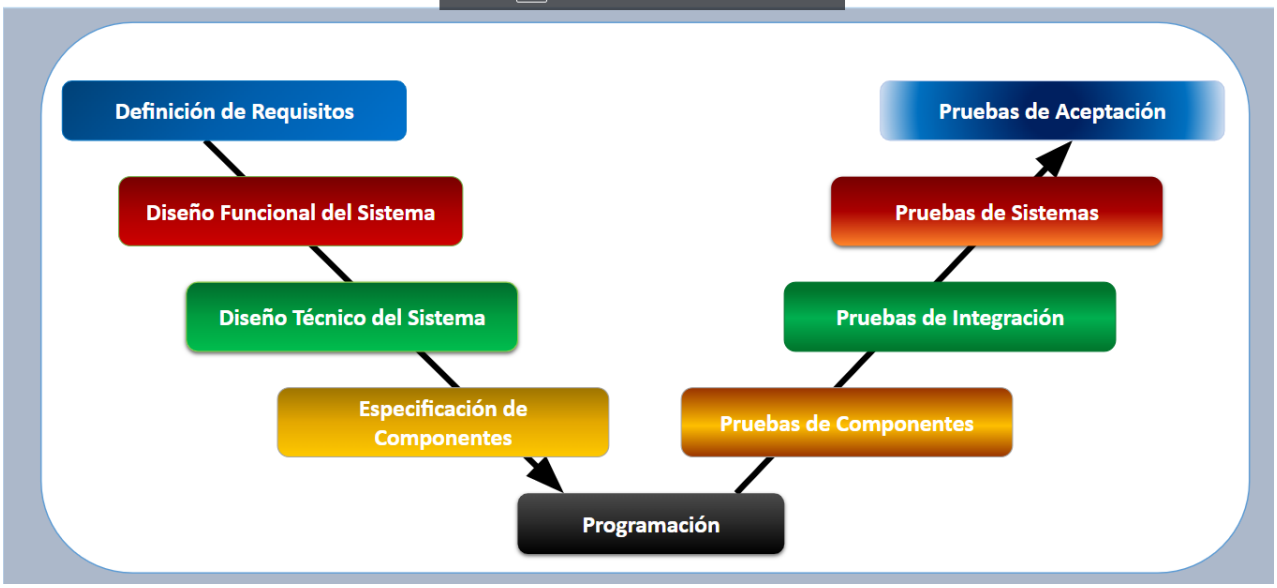
Un modelo de desarrollo secuencial describe el proceso de desarrollo de software como un flujo lineal y secuencial de actividades. Esto significa que cualquier fase del proceso de desarrollo debe comenzar cuando se haya completado la fase anterior. En teoría, no hay solapamiento de fases, pero en la práctica, es beneficioso tener una retroalimentación temprana de la fase siguiente.

En el modelo en Cascada, las actividades de desarrollo (por ejemplo, análisis de requisitos,

diseño, codificación, prueba) se completan una tras otra. En este modelo, las actividades de prueba sólo ocurren después de que todas las demás actividades de desarrollo hayan sido completadas.

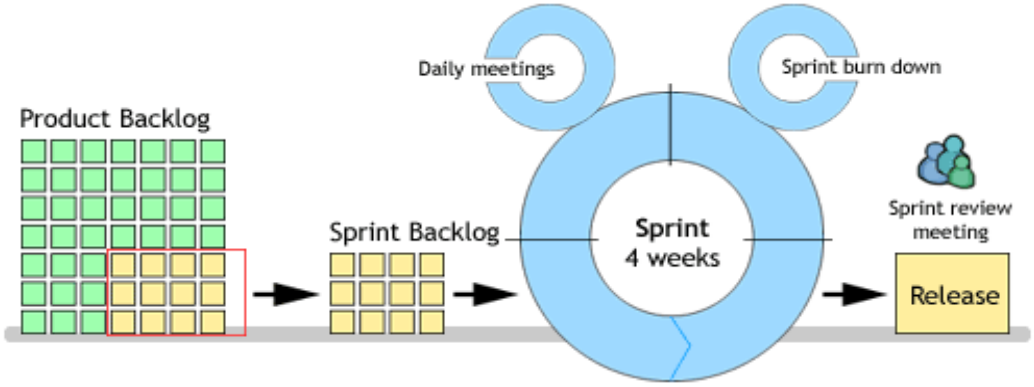
A diferencia del modelo en Cascada, el modelo en V integra el proceso de prueba a lo largo de todo el proceso de desarrollo, implementando el principio de la prueba temprana. Además, el modelo en V incluye niveles de prueba asociados con cada fase de desarrollo correspondiente, lo que favorece aún más la prueba temprana. En este modelo, la ejecución de las pruebas asociadas a cada nivel de prueba tiene lugar de forma secuencial, pero en algunos casos se producen solapamientos.

Modelo V – Desarrollo Secuencial



Los modelos de desarrollo secuencial ofrecen software que contiene el conjunto completo de prestaciones, pero normalmente requieren meses o años para su entrega a los implicados y usuarios.

El desarrollo incremental implica establecer requisitos, diseñar, construir y probar un sistema en fragmentos, lo que significa que las prestaciones del software crecen de forma incremental. El tamaño de estos incrementos de prestaciones varía, ya que algunos métodos tienen piezas más grandes y otros más pequeñas. Los incrementos de prestaciones pueden ser tan pequeños como un simple cambio en una pantalla de la interfaz de usuario o una nueva opción en una consulta.



El desarrollo iterativo se produce cuando se especifican, diseñan, construyen y prueban conjuntamentegrupos de prestaciones en una serie de ciclos, a menudo, de una duración fija. Las iteraciones pueden implicar cambios en las prestaciones desarrolladas en iteraciones anteriores, junto con cambios en el alcance del proyecto. Cada iteración proporciona software operativo, que es un subconjunto creciente del conjunto general de prestaciones hasta que se entrega el software final o se detiene el desarrollo.

Algunos ejemplos son:

- Rational Unified Process: Cada iteración tiende a ser relativamente larga (por ejemplo, de dos atres meses), y los incrementos de las prestaciones son proporcionalmente grandes, como por ejemplo dos o tres grupos de prestaciones relacionadas.
- Scrum: Cada iteración tiende a ser relativamente corta (por ejemplo, horas, días o unas pocas semanas), y los incrementos de las prestaciones son proporcionalmente pequeños, como unaspocas mejoras y/o dos o tres prestaciones nuevas.
- Kanban: Implementado con o sin iteraciones de longitud fija, que puede ofrecer una sola mejora o prestación una vez finalizada, o puede agrupar prestaciones para liberarlas de una sola vez.
- Espiral (o prototipado): Implica la creación de incrementos experimentales, algunos de los cualespueden ser reelaborados en profundidad o incluso abandonados en trabajos de desarrollo posteriores.

Los componentes o sistemas desarrollados utilizando estos métodos, a menudo, implican niveles de prueba que se solapan e iteran a lo largo de todo el desarrollo. Lo ideal es que cada prestación se pruebeen varios niveles de prueba a medida que se aproxima la entrega. En algunos casos, los equipos utilizanla entrega continua o el despliegue continuo, lo que implica una automatización significativa de múltiplesniveles de prueba como parte de sus canales de entrega[1]. Muchos de los esfuerzos de desarrollo queutilizan estos métodos también incluyen el concepto de equipos auto-organizados, que pueden cambiar la forma en que se organiza el trabajo de prueba, así como la relación entre los probadores y los desarrolladores.

Estos métodos generan un sistema en crecimiento, que puede ser liberado a los usuarios finales prestacióna prestación, iteración a iteración, o en la forma de un lanzamiento a gran escala más tradicional. Independientemente de si los incrementos de software se liberan a los usuarios finales, la prueba de regresión es cada vez más importante a medida que el sistema crece.

A diferencia de los modelos secuenciales, los modelos iterativos e incrementales pueden ofrecer software utilizable en semanas o incluso días, pero sólo pueden ofrecer el conjunto completo de requisitos durante un período de meses o incluso años.

Para más información sobre las pruebas de software en el contexto del desarrollo ágil, véase Programade Estudio de Probador Certificado del ISTQB® de Nivel Básico, Extensión Ágil (ISTQB-AT por sus siglas en inglés) del ISTQB.

1.2 Modelos de Ciclo de Vida del Desarrollo de Software en Contexto

Los modelos de ciclo de vida de desarrollo de software deben seleccionarse y adaptarse al contexto delas características del proyecto y del producto. Se debe seleccionar y adaptar un modelo apropiado delciclo de vida del desarrollo de software en función del objetivo del proyecto, el tipo de producto que se está desarrollando, las prioridades del negocio (por ejemplo, el tiempo de comercialización), y los riesgos deproducto y de proyecto identificados. Por ejemplo, el desarrollo y la prueba de un sistema de gestión internamenor debe diferir del desarrollo y la prueba de un sistema crítico para la seguridad física, como el sistema de control de frenos de un automóvil. Como otro ejemplo, en algunos casos las cuestiones relativas a laorganización y de índole cultural pueden inhibir la comunicación entre los miembros del equipo, lo que puede impedir el desarrollo iterativo.

Dependiendo del contexto del proyecto, puede ser necesario combinar o reorganizar los niveles de prueba y/o las actividades de prueba. Por ejemplo, para la integración de un producto de software comercial dedistribución masiva (COTS por sus siglas en inglés) en un sistema más amplio, el comprador puede realizar pruebas de interoperabilidad a nivel de prueba de integración de sistemas (por ejemplo, integración en lainfraestructura y otros sistemas) y a nivel de prueba de aceptación (funcional y no funcional, junto con laprueba de aceptación del usuario y la prueba de aceptación operativa). Véase la sección 2.2 para una descripción de los niveles de prueba y la sección 2.3 para una descripción de los tipos de prueba.

Además, se pueden combinar los propios modelos de ciclo de vida de desarrollo de software. Por ejemplo, un modelo en V puede ser usado para el desarrollo y prueba de los sistemas backend y sus integraciones, mientras que un modelo de desarrollo ágil puede ser usado para desarrollar y probar la interfaz de usuario (UI) y funcionalidad del front-end. El prototipado puede ser utilizado en las primeras etapas de un proyecto, adoptando un modelo de desarrollo incremental una vez finalizada la fase experimental.

Los sistemas de Internet de las Cosas[20] (IoT por sus siglas en inglés), que consisten en muchos objetosdiferentes, tales como dispositivos, productos y servicios, suelen aplicar modelos de ciclo de vida de desarrollo de software separados para cada objeto. Esto supone un reto especial para el desarrollo de las versiones de sistemas de Internet de las Cosas. Además, el ciclo de vida de desarrollo de software de dichos objetos pone un mayor énfasis en las fases posteriores del ciclo de vida de desarrollo de softwaredespués de que se hayan introducido para su uso operativo (por ejemplo, las fases de operación, actualización y retirada).

2. Niveles de Prueba

Los niveles de prueba son grupos de actividades de prueba que se organizan y gestionan conjuntamente. Cada nivel de prueba es una instancia del proceso de prueba, que consiste en las actividades realizadas en relación con el software en un nivel de desarrollo determinado, desde unidades o componentes individuales hasta sistemas completos o, en su caso, sistemas de sistemas. Los niveles de prueba están relacionados con otras actividades dentro del ciclo de vida de desarrollo de software.

Los niveles de prueba utilizados en este programa de estudio son:

- Prueba de componente.
- Prueba de integración.
- Prueba de sistema.
- Prueba de aceptación.

Los niveles de prueba se caracterizan por los siguientes atributos:

- Objetivos específicos.
- Bases de prueba, referenciadas para generar casos de prueba.
- Objeto de prueba (es decir, lo que se está probando).
- Defectos y fallos característicos.
- Enfoques y responsabilidades específicos.

Se requiere un entorno de prueba adecuado para cada nivel de prueba. En la prueba de aceptación, porejemplo, un entorno de prueba similar al de producción es ideal, mientras que en la prueba de componente los desarrolladores suelen utilizar su propio entorno de desarrollo.

2.1 Prueba de Componente

Objetivos de la Prueba de Componente

La prueba de componente (también conocida como prueba unitaria o de módulo) se centra en los componentes que se pueden probar por separado. Los objetivos de la prueba de componente incluyen:

- Reducir el riesgo.
- Verificar que los comportamientos funcionales y no funcionales del componente son los diseñados y especificados.
- Generar confianza en la calidad del componente.
- Encontrar defectos en el componente.
- Prevenir la propagación de defectos a niveles de prueba superiores.

En algunos casos, especialmente en modelos de desarrollo incrementales e iterativos (por ejemplo, Ágiles) donde los cambios de código son continuos, la prueba de regresión de componente automatizada juegaun papel clave en la construcción de la confianza en que los cambios no han dañado a los componentesexistentes.

La prueba de componente, a menudo, se realiza de forma aislada del resto del sistema, dependiendo delmodelo de ciclo de vida de desarrollo de software y del sistema, lo que puede requerir objetos simulados,virtualización de servicios, arneses, stubs y controladores. La prueba de componente pueden cubrir la funcionalidad (por ejemplo, la exactitud de los cálculos), las características no funcionales (por ejemplo, la búsqueda de fugas de memoria) y las propiedades estructurales (por ejemplo, pruebas de decisión).

Bases de Prueba

Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba para la prueba decomponente incluyen:

- Diseño detallado.
- Código.
- Modelo de datos.
- Especificaciones de los componentes.

Objetos de Prueba

Los objetos de prueba característicos para la prueba de componente incluyen:

- Componentes, unidades o módulos.
- Código y estructuras de datos.
- Clases.
- Módulos de base de datos.

Defectos y Fallos Característicos

Ejemplos de defectos y fallos característicos de la prueba de componente incluyen:

- Funcionamiento incorrecto (por ejemplo, no como se describe en las especificaciones de diseño).
- Problemas de flujo de datos.

- Código y lógica incorrectos.

Por lo general, los defectos se corrigen tan pronto como se detectan, a menudo sin una gestión formal delos defectos. Sin embargo, cuando los desarrolladores informan sobre defectos, esto proporciona información importante para el análisis de la causa raíz y la mejora del proceso.

Enfoques y Responsabilidades Específicos

En general, el desarrollador que escribió el código realiza la prueba de componente, pero al menos requiere acceso al código que se está probando. Los desarrolladores pueden alternar el desarrollo de componentes con la búsqueda y corrección de defectos. A menudo, los desarrolladores escriben y ejecutan pruebas después de haber escrito el código de un componente. Sin embargo, especialmente en el desarrollo Ágil, la redacción de casos de prueba de componente automatizados puede preceder a la redacción del código de la aplicación.

Por ejemplo, considerar el desarrollo guiado por pruebas (TDD por sus siglas en inglés). El desarrollo guiado por pruebas es altamente iterativo y se basa en ciclos de desarrollo de casos de prueba automatizados, luego se construyen e integran pequeños fragmentos de código, a continuación, se ejecuta la prueba de componente, se corrige cualquier cuestión y se refactoriza el código. Este proceso continúahasta que el componente ha sido completamente construido y ha pasado toda la prueba de componente.El desarrollo guiado por pruebas es un ejemplo de un enfoque del tipo "prueba primero". Aunque el desarrollo guiado por pruebas se originó en eXtreme Programming (XP), se ha extendido a otras formasAgiles y también a ciclos de vida secuenciales (ver el Programa de Estudio de Probador Certificado delISTQB® de Nivel Básico, Extensión Ágil - ISTQB-AT).

2.2 Prueba de Integración

Objetivos de la Prueba de Integración

La prueba de integración se centra en las interacciones entre componentes o sistemas. Los objetivos dela prueba de integración incluyen:

- Reducir el riesgo.
- Verificar que los comportamientos funcionales y no funcionales de las interfaces sean losdiseñados y especificados.
- Generar confianza en la calidad de las interfaces.
- Encontrar defectos (que pueden estar en las propias interfaces o dentro de los componentes osistemas).
- Prevenir la propagación de defectos a niveles de prueba superiores.

De la misma forma que con la prueba de componente, en algunos casos la prueba de regresión de integración automatizada proporciona la confianza de que los cambios no han dañado a las interfaces, los componentes o los sistemas existentes.

Hay dos niveles de prueba de integración diferentes que se describen en este programa de estudio y quese pueden llevar a cabo sobre objetos de prueba de diferentes tamaños de la siguiente manera:

- Las pruebas de integración de componentes se centran en las interacciones e interfaces entre loscomponentes integrados. La prueba de integración de componentes se realiza después de las pruebas de componente y, en general, está automatizada. En el desarrollo iterativo e incremental,la prueba de integración de componentes suele formar parte del proceso de integración continua.
- Las pruebas de integración de sistemas se centran en las interacciones e interfaces entre sistemas, paquetes y microservicios. Las pruebas de integración de sistemas también

pueden cubrir las interacciones con, e interfaces proporcionadas por, organizaciones externas (por ejemplo, servicios web). En este caso, la organización que desarrolla no controla las interfaces externas, lo que puede crear varios obstáculos para la prueba (por ejemplo, asegurar que se resuelvan los defectos de bloqueo de pruebas en el código de la organización externa, organizar los entornos de pruebas, etc.). Las pruebas de integración de sistemas pueden realizarse después de la pruebade sistema o en paralelo con las actividades de prueba de sistema en curso (tanto en el desarrollo secuencial como en el desarrollo iterativo e incremental).

Bases de Prueba

Algunos ejemplos de productos de trabajo que pueden utilizarse como base de prueba para la prueba deintegración incluyen:

- Diseño de software y sistemas.
- Diagramas de secuencia.
- Especificaciones de interfaz y protocolos de comunicación.
- Casos de uso.
- Arquitectura a nivel de componente o de sistema.
- Flujos de trabajo.
- Definiciones de interfaces externas.

Objetos de Prueba

Los objetos de prueba característicos para la prueba de integración incluyen:

- Subsistemas.
- Bases de datos.
- Infraestructura.
- Interfaces.
- Interfaces de programación de aplicaciones (API por sus siglas en inglés).
- Microservicios.

Defectos y Fallos Característicos

Entre los ejemplos de defectos y fallos característicos de la prueba de integración de componentes se incluyen los siguientes:

- Datos incorrectos, datos faltantes o codificación incorrecta de datos.
- Secuenciación o sincronización incorrecta de las llamadas a la interfaz.
- Incompatibilidad de la interfaz.
- Fallos en la comunicación entre componentes.
- Fallos de comunicación entre componentes no tratados o tratados de forma incorrecta.
- Suposiciones incorrectas sobre el significado, las unidades o las fronteras de los datos que se transmiten entre componentes.

Entre los ejemplos de defectos y fallos característicos de la prueba de integración de sistemas se incluyen los siguientes:

- Estructuras de mensajes inconsistentes entre sistemas.
- Datos incorrectos, datos faltantes o codificación incorrecta de datos.
- Incompatibilidad de la interfaz.
- Fallos en la comunicación entre sistemas.
- Fallos de comunicación entre sistemas no tratados o tratados de forma incorrecta. Suposicionesincorrectas sobre el significado, las unidades o las fronteras de los datos que se transmiten entresistemas.
- Incumplimiento de las normas de seguridad obligatorias.

Enfoques y Responsabilidades Específicos

La prueba de integración de componentes y la prueba de integración de sistemas deben concentrarse enla integración propiamente dicha. Por ejemplo, si se integra el módulo A con el módulo B, la prueba debecentrarse en la comunicación entre los módulos, no en la funcionalidad de los módulos individuales, como debería haberse hecho durante la prueba de componente. Si se integra el sistema X con el sistema Y, laprueba debe centrarse en la comunicación entre los sistemas, no en la funcionalidad de los sistemas individuales, como debería haberse hecho durante la prueba de sistema. Se puede utilizar los tipos de prueba funcional, no funcional y estructural.

La prueba de integración de componentes suele ser responsabilidad de los desarrolladores. La prueba de integración de sistemas es, en general, responsabilidad de los probadores. En condiciones ideales, losprobadores que realizan la prueba de integración de sistemas deberían entender la arquitectura del sistema y deberían haber influido en la planificación de la integración.

Si la prueba de integración y la estrategia de integración se planifican antes de que se construyan los componentes o sistemas, estos componentes o sistemas se pueden construir en el orden adecuado para que la prueba sea más eficiente. Las estrategias de integración sistemática pueden basarse en la arquitectura del sistema (por ejemplo, ascendente[21] y descendente[22]), en tareas funcionales, ensecuencias de procesamiento de transacciones o en algún otro aspecto del sistema o de los componentes. Para simplificar el aislamiento de defectos y detectar defectos de forma temprana, la integración debe ser normalmente incremental (es decir, un pequeño número de componentes o sistemas adicionales a la vez) en lugar de "big bang" (es decir, la integración de todos los componentes o sistemas en un solo paso). Unanálisis de riesgo de las interfaces más complejas puede ayudar a centrar la prueba de integración.

Cuanto mayor sea el alcance de la integración, más difícil será aislar los defectos de un componente osistema específico, lo que puede conducir a un mayor riesgo y a un tiempo adicional para la resolución de problemas. Esta es una de las razones por las que la integración continua, en la que el software se integra componente a componente (es decir, la integración funcional), se ha convertido en una práctica común.Esta integración continua incluye, a menudo, pruebas de regresión automatizadas, idealmente en los diferentes niveles de prueba.

2.3 Prueba de Sistema

Objetivos de la Prueba de Sistema

La prueba de sistema se centra en el comportamiento y las capacidades de todo un sistema o producto, a menudo teniendo en cuenta las tareas extremo a extremo que el sistema puede realizar y loscomportamientos no funcionales que exhibe mientras realiza esas tareas. Los objetivos de la prueba desistema incluyen:

- Reducir el riesgo.

- Verificar que los comportamientos funcionales y no funcionales del sistema son los diseñados yespecificados.
- Validar que el sistema está completo y que funcionará como se espera.
- Generar confianza en la calidad del sistema considerado como un todo.
- Encontrar defectos.
- Prevenir la propagación de defectos a niveles de prueba superiores o a producción.

Para ciertos sistemas, la verificación de la calidad de los datos puede ser un objetivo. Al igual que con laprueba de componente y la prueba de integración, en algunos casos la prueba de regresión de sistemaautomatizada proporciona la confianza de que los cambios no han dañado características existentes o capacidades extremo a extremo. A menudo, la prueba de sistema produce información que es utilizadapor los implicados para tomar decisiones con respecto al lanzamiento. La prueba de sistema también puede satisfacer requisitos o estándares legales o regulatorios.

El entorno de prueba debe corresponder, en condiciones ideales, al entorno objetivo final o entorno deproducción.

Bases de Prueba

Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba para la prueba desistema incluyen:

- Especificaciones de requisitos del sistema y del software (funcionales y no funcionales).
- Informes de análisis de riesgo.
- Casos de uso.
- Épicas e historias de usuario.
- Modelos de comportamiento del sistema.
- Diagramas de estado.
- Manuales del sistema y del usuario.

Objetos de Prueba

Los objetos de prueba característicos para la prueba de sistema incluyen:

- Aplicaciones.
- Sistemas hardware/software.
- Sistemas operativos.
- Sistema sujeto a prueba (SSP).
- Configuración del sistema y datos de configuración.

Defectos y Fallos Característicos

Entre los ejemplos de defectos y fallos característicos de la prueba de sistema se incluyen los siguientes:

- Cálculos incorrectos.
- Comportamiento funcional o no funcional del sistema incorrecto o inesperado.
- Control y/o flujos de datos incorrectos dentro del sistema.
- Incapacidad para llevar a cabo, de forma adecuada y completa, las tareas funcionales

extremo aextremo.

- Fallo del sistema para operar correctamente en el/los entorno/s de producción.
- Fallo del sistema para funcionar como se describe en los manuales del sistema y de usuario.

Enfoques y Responsabilidades Específicos

La prueba de sistema debe centrarse en el comportamiento global y extremo a extremo del sistema en su conjunto, tanto funcional como no funcional. La prueba de sistema debe utilizar las técnicas más apropiadas (véase el capítulo 4) para los aspectos del sistema que serán probados. Por ejemplo, se puede crear una tabla de decisión para verificar si el comportamiento funcional es el descrito en términos de reglas de negocio.

Los probadores independientes, en general, llevan a cabo la prueba de sistema. Los defectos en las especificaciones (por ejemplo, la falta de historias de usuario, los requisitos de negocio establecidos deforma incorrecta, etc.) pueden llevar a una falta de comprensión o a desacuerdos sobre el comportamiento esperado del sistema. Tales situaciones pueden causar falsos positivos y falsos negativos, lo que haceperder tiempo y reduce la eficacia de la detección de defectos, respectivamente. La participación temprana de los probadores en el perfeccionamiento de la historia de usuario o en actividades de prueba estática,como las revisiones, ayuda a reducir la incidencia de tales situaciones.

2.4 Prueba de Aceptación

Objetivos de la Prueba de Aceptación

La prueba de aceptación, al igual que la prueba de sistema, se centra normalmente en el comportamiento y las capacidades de todo un sistema o producto. Los objetivos de la prueba de aceptación incluyen:

- Establecer confianza en la calidad del sistema en su conjunto.
- Validar que el sistema está completo y que funcionará como se espera.
- Verificar que los comportamientos funcionales y no funcionales del sistema sean los especificados.

La prueba de aceptación puede producir información para evaluar el grado de preparación del sistema para su despliegue y uso por parte del cliente (usuario final). Los defectos pueden encontrarse durante las prueba de aceptación, pero encontrar defectos no suele ser un objetivo, y encontrar un número significativo de defectos durante la prueba de aceptación puede, en algunos casos, considerarse un riesgo importante para el proyecto. La prueba de aceptación también pueden satisfacer requisitos o normas legales o reglamentarios.

Las formas comunes de prueba de aceptación incluyen las siguientes:

- Prueba de aceptación de usuario.
- Prueba de aceptación operativa.
- Prueba de aceptación contractual y de regulación.
- Prueba alfa y beta.

Cada uno de estas se describe en las siguientes cuatro subsecciones.

- **Prueba de Aceptación de Usuario (UAT por sus siglas en inglés)**

La prueba de aceptación de sistema por parte de los usuarios se centra normalmente en la

validación dela idoneidad para el uso del sistema por parte de los usuarios previstos en un entorno operativo real o simulado. El objetivo principal es crear confianza en que los usuarios pueden utilizar el sistema para satisfacer sus necesidades, cumplir con los requisitos y realizar los procesos de negocio con la mínimadificultad, coste y riesgo.

- **Prueba de Aceptación Operativa**

La prueba de aceptación de sistema por parte del personal de operaciones o de la administración del sistema se realiza, por lo general, en un entorno de producción (simulado). La prueba se centra en los aspectos operativos, y pueden incluir:

- Prueba de copia de seguridad y restauración.
- Instalación, desinstalación y actualización.
- Recuperación ante desastres.
- Gestión de usuarios.
- Tareas de mantenimiento.
- Carga de datos y tareas de migración.
- Comprobación de vulnerabilidades de seguridad.
- Prueba de rendimiento.

El objetivo principal de la prueba de aceptación operativa es generar confianza en que los operadores oadministradores del sistema pueden mantener el sistema funcionando correctamente para los usuarios en el entorno operativo, incluso en condiciones excepcionales o difíciles.

- **Prueba de Aceptación Contractual y Normativa**

La prueba de aceptación contractual se realiza en función de los criterios de aceptación del contrato parael desarrollo de software a medida. Los criterios de aceptación deben definirse cuando las partes acuerdan el contrato. La prueba de aceptación contractual suele ser realizada por usuarios o por probadores independientes.

La prueba de aceptación normativa se lleva a cabo con respecto a cualquier norma que deba cumplirse,como las normas gubernamentales, legales o de seguridad física. La prueba de aceptación normativa suele ser realizada por los usuarios o por probadores independientes, en ocasiones los resultados sonpresenciados o auditados por agencias reguladoras.

El principal objetivo de la prueba de aceptación contractual y normativa es crear confianza en que se halogrado la conformidad contractual o normativa.

- **Pruebas Alfa y Beta**

Las pruebas alfa y beta suelen ser utilizadas por los desarrolladores de software comercial de distribución masiva (COTS por sus siglas en inglés) que desean obtener retroalimentación de los usuarios, clientes y/u operadores potenciales o existentes antes de que el producto de software sea puesto en el mercado. Laprueba alfa se realiza en las instalaciones de la organización que desarrolla, no por el equipo de desarrollo, sino por clientes potenciales o existentes, y/u operadores o un equipo de prueba independiente. La prueba beta es realizada por clientes potenciales o existentes, y/u operadores en sus propias instalaciones. Laprueba beta puede tener lugar después de la prueba alfa, o puede ocurrir sin que se haya realizado ninguna prueba alfa previa.

Uno de los objetivos de las pruebas alfa y beta es generar confianza entre los clientes potenciales o existentes y/u operadores de que pueden utilizar el sistema en condiciones normales y cotidianas, así como en el entorno o los entornos operativos, para lograr sus objetivos con la mínima dificultad, coste y riesgo. Otro objetivo puede ser la detección de defectos relacionados con las condiciones y el entorno olos entornos en los que se utilizará el sistema, especialmente cuando las condiciones y los entornos sean difíciles de reproducir por parte del equipo de desarrollo.

Base de Prueba

Entre los ejemplos de productos de trabajo que se pueden utilizar como base de prueba para cualquierforma de prueba de aceptación se encuentran:

- Procesos de negocio.
- Requisitos de usuario o de negocio.
- Normativas, contratos legales y estándares.
- Casos de uso.
- Requisitos de sistema.
- Documentación del sistema o del usuario.
- Procedimientos de instalación.
- Informes de análisis de riesgo.

Además, como base de prueba para derivar casos de prueba para la prueba de aceptación operativa, sepueden utilizar uno o más de los siguientes productos de trabajo:

- Procedimientos de copia de seguridad y restauración.
- Procedimientos de recuperación de desastres.
- Requisitos no funcionales.
- Documentación de operaciones.
- Instrucciones de despliegue e instalación.
- Objetivos de rendimiento.
- Paquetes de base de datos.
- Estándares o reglamentos de seguridad.

Objetos de Prueba Característicos

Los objetos de prueba característicos para cualquier forma de prueba de aceptación incluyen:

- Sistema sujeto a prueba.
- Configuración del sistema y datos de configuración.
- Procesos de negocio para un sistema totalmente integrado.
- Sistemas de recuperación y sitios críticos (para pruebas de continuidad del negocio y recuperación de desastres).
- Procesos operativos y de mantenimiento.
- Formularios.
- Informes.
- Datos de producción existentes y transformados.

Defectos y Fallos Característicos

Entre los ejemplos de defectos característicos de cualquier forma de prueba de aceptación se encuentran:

- Los flujos de trabajo del sistema no cumplen con los requisitos de negocio o de usuario.
- Las reglas de negocio no se implementan de forma correcta.
- El sistema no satisface los requisitos contractuales o reglamentarios.
- Fallos no funcionales tales como vulnerabilidades de seguridad, eficiencia de rendimientoinadecuada bajo cargas elevadas o funcionamiento inadecuado en una plataforma soportada.

Enfoques y Responsabilidades Específicos

La prueba de aceptación es, a menudo, responsabilidad de los clientes, usuarios de negocio, propietarios de producto u operadores de un sistema, y otros implicados también pueden estar involucrados.

La prueba de aceptación se considera, a menudo, como el último nivel de prueba en un ciclo de vida dedesarrollo secuencial, pero también pueden ocurrir en otros momentos, por ejemplo:

- La prueba de aceptación de un producto software comercial de distribución masiva (COTS por sus siglas en inglés) puede tener lugar cuando se instala o integra.
- La prueba de aceptación de una mejora funcional nueva puede tener lugar antes de la prueba desistema.

En el desarrollo iterativo, los equipos de proyecto pueden emplear varias formas de prueba de aceptación durante y al final de cada iteración, como las que se centran en verificar una nueva característica en relación con sus criterios de aceptación y las que se centran en validar que una nueva característica satisface las necesidades de los usuarios. Además, se pueden realizar pruebas alfa y beta, ya sea al final de cada iteración, después de la finalización de cada iteración, o después de una serie de iteraciones. La prueba de aceptación de usuario, prueba de aceptación operativa, prueba de aceptación normativa y prueba de aceptación contractual también pueden t, ya sea al cierre de cada iteración, después de la finalización de cada iteración, o después de una serie de iteraciones.

3. Tipos de prueba

Un tipo de prueba es un grupo de actividades de prueba destinadas a probar características específicasde un sistema de software, o de una parte de un sistema, basadas en objetivos de prueba específicos.

Dichos objetivos pueden incluir:

- Evaluar las características de calidad funcional, como la completitud, corrección y pertinencia.
- Evaluar características no funcionales de calidad, tales como fiabilidad, eficiencia de desempeño, seguridad, compatibilidad y usabilidad.
- Evaluar si la estructura o arquitectura del componente o sistema es correcta, completa y según lo especificado.
- Evaluar los efectos de los cambios, tales como confirmar que los defectos han sido corregidos (prueba de confirmación) y buscar cambios no deseados en el comportamiento que resulten decambios en el software o en el entorno (prueba de regresión).

3.1 Prueba Funcional

La prueba funcional de un sistema incluye pruebas que evalúan las funciones que el sistema debe realizar. Los requisitos funcionales pueden estar descritos en productos de trabajo tales como especificaciones de requisitos de negocio, épicas, historias de usuario, casos de uso, o especificaciones funcionales, o pueden estar sin documentar. Las funciones describen "qué" debe hacer el sistema.

Se deben realizar pruebas funcionales en todos los niveles de prueba (por ejemplo, la prueba de componente puede basarse en una especificación de componentes), aunque el enfoque es diferente encada nivel.

La prueba funcional observa el comportamiento del software, por lo que se pueden utilizar técnicas de caja negra para obtener las condiciones de prueba y los casos de prueba para la funcionalidad del componente o sistema.

Se puede medir la intensidad de la prueba funcional a través de la cobertura funcional. La cobertura funcional es la medida en que algún tipo de elemento funcional ha sido practicado por pruebas, y se expresa como un porcentaje del tipo o tipos de elemento cubiertos. Por ejemplo, utilizando la trazabilidadentre pruebas y requisitos funcionales, se puede calcular el porcentaje de estos requisitos abordados por las pruebas, identificando potencialmente carencias en la cobertura.

El diseño y la ejecución de pruebas funcionales pueden implicar competencias o conocimientos especiales, como el conocimiento del problema de negocio específico que resuelve el software (por ejemplo, el software de modelado geológico para las industrias del petróleo y el gas) o el papel particular que desempeña el software (por ejemplo, el software de juegos de azar de ordenador que proporciona entretenimiento interactivo).

3.2 Prueba No Funcional

La prueba no funcional de un sistema evalúa las características de sistemas y software, como la usabilidad,la eficiencia del desempeño o la seguridad. Consulte el estándar ISO (ISO/IEC 25010) para una clasificación de las características de calidad de producto software. La prueba no funcional prueba "cómo de bien" se comporta el sistema[23].

Contrariamente a las percepciones erróneas generalizadas, se pueden y, a menudo, se deben realizarpruebas no funcionales en todos los niveles de prueba, y se deben realizar tan pronto como sea posible.El descubrimiento tardío de defectos no funcionales puede ser extremadamente peligroso para el éxito de un proyecto.

Se pueden utilizar técnicas de caja negra para obtener condiciones de prueba y casos de prueba para pruebas no funcionales. Por ejemplo, se puede utilizar el análisis de valores frontera para definir las condiciones de estrés para las pruebas de rendimiento.

Se puede medir la intensidad de la prueba no funcional a través de la cobertura no funcional. La cobertura no funcional es la medida en que algún tipo de elemento no funcional ha sido practicado por pruebas, y se expresa como un porcentaje del tipo o tipos de elemento cubiertos. Por ejemplo, utilizando la trazabilidadentre las pruebas y los dispositivos compatibles con una aplicación móvil, se puede calcular el porcentaje de dispositivos tratados por las pruebas de compatibilidad, identificando potencialmente las carencias enla cobertura.

El diseño y la ejecución de la prueba no funcional pueden implicar competencias o conocimientos especiales, como el conocimiento de las debilidades inherentes a un diseño o tecnología (por ejemplo,vulnerabilidades de seguridad asociadas con determinados lenguajes de programación) o la base de usuarios concreta (por ejemplo, la persona[24] de los usuarios de los sistemas de gestión de centros sanitarios).

3.3 Prueba de Caja Blanca

La prueba de caja blanca obtiene pruebas basadas en la estructura interna del sistema o en su implementación. La estructura interna puede incluir código, arquitectura, flujos de trabajo y/o flujos de datosdentro del sistema.

Se puede medir la intensidad de la prueba de caja blanca a través de la cobertura estructural. La cobertura estructural es la medida en que algún tipo de elemento estructural ha sido practicado mediante pruebas, y se expresa como un porcentaje del tipo de elemento cubierto.

En el nivel de prueba de componente, la cobertura de código se basa en el porcentaje de código del componente que ha sido probado, y puede medirse en términos de diferentes aspectos del código (elementos de cobertura), tales como el porcentaje de sentencias ejecutables probadas en el componente, o el porcentaje de resultados de decisión probados. Estos tipos de cobertura se denominan, de forma colectiva, cobertura de código. En el nivel de prueba de integración de componentes, la prueba de cajablanca pueden basarse en la arquitectura del sistema, como las interfaces entre componentes, y la cobertura estructural puede medirse en términos del porcentaje de interfaces practicadas por las pruebas.

El diseño y la ejecución de la prueba de caja blanca pueden implicar competencias o conocimientos especiales, como la forma en que se construye el código (por ejemplo, para utilizar herramientas de cobertura de código), cómo se almacenan los datos (por ejemplo, para evaluar posibles consultas a la base de datos), y cómo utilizar las herramientas de cobertura e interpretar correctamente sus resultados.

3.4 Prueba Asociada al Cambio

Cuando se realizan cambios en un sistema, ya sea para corregir un defecto o debido a una funcionalidadnueva o modificada, se debe probar para confirmar que los cambios han corregido el defecto o implementado la funcionalidad correctamente, y no han causado ninguna consecuencia adversaimprevista.

- Prueba de confirmación: Una vez corregido un defecto, el software se puede probar con todos los casos de prueba que fallaron debido al defecto, que se deben volver a ejecutar en la nueva versión de software. El software también puede probarse con nuevas pruebas si, por ejemplo, el defectoconsistía en la falta de una funcionalidad. Como mínimo, los pasos para reproducir el fallo o losfallos causados por el defecto deben volver a ejecutarse en la nueva versión del software. El objetivo de una prueba de confirmación es confirmar que el defecto original se ha solucionado deforma satisfactoria.
- Prueba de regresión: Es posible que un cambio hecho en una parte del código, ya sea una corrección u otro tipo de cambio, pueda afectar accidentalmente el comportamiento de otras partes del código, ya sea dentro del mismo componente, en otros componentes del mismo sistema, o incluso en otros sistemas. Los cambios pueden incluir modificaciones en el entorno, tales comouna nueva versión de un sistema operativo o de un sistema de gestión de bases de datos. Estosefectos secundarios no deseados se denominan regresiones. La prueba de regresión implica larealización de pruebas para detectar estos efectos secundarios no deseados.

La prueba de confirmación y la prueba de regresión se realizan en todos los niveles de prueba.

Especialmente en los ciclos de vida de desarrollo iterativos e incrementales (por ejemplo, Agile), las nuevascaracterísticas, los cambios en las características existentes y la refactorización del código dan como resultado cambios frecuentes en el código, lo que también requiere pruebas asociadas al cambio. Debido a la naturaleza evolutiva del sistema, la prueba de confirmación y la prueba regresión son muy importantes.Esto es particularmente relevante para los sistemas de Internet de las Cosas, donde los objetos individuales (por ejemplo, los dispositivos) se actualizan o reemplazan con frecuencia.

Los juegos de prueba de regresión se ejecutan muchas veces y generalmente evolucionan lentamente,por lo que la prueba de regresión es un fuerte candidato para la automatización. La automatización de estas pruebas debería comenzar al principio del proyecto (ver capítulo 6).

4. Tipos de Prueba y Niveles de Prueba

Es posible realizar cualquiera de los tipos de prueba mencionados anteriormente en cualquier nivel de prueba. Para ilustrar, se darán ejemplos de pruebas funcionales, no funcionales, de caja blanca y asociadas al cambio en todos los niveles de prueba, para una aplicación bancaria, comenzando con pruebas funcionales:

- Para la prueba de componente, las pruebas se diseñan en base a la forma en que un componente debe calcular el interés compuesto.
- Para la prueba de integración de componentes, las pruebas se diseñan en función de cómo lainformación de la cuenta capturada en la interfaz de usuario se transfiere a la lógica de negocio.
- Para la prueba de sistema, las pruebas se diseñan en base a cómo los titulares de cuentas puedensolicitar una línea de crédito sobre sus cuentas corrientes.
- Para la prueba de integración de sistemas, las pruebas se diseñan en función de cómo el sistema utiliza un microservicio externo para comprobar la calificación crediticia del titular de una cuenta.
- Para la prueba de aceptación, las pruebas se diseñan en base a la forma en que el empleado delbanco tramita la aprobación o rechazo de una solicitud de crédito.

Los siguientes son ejemplos de pruebas no funcionales:

- Para la prueba de componente, las pruebas de rendimiento están diseñadas para evaluar elnúmero de ciclos de CPU necesarios para realizar un cálculo de intereses totales complejo.
- Para la prueba de integración de componentes, las pruebas de seguridad están diseñadas para vulnerabilidades de desbordamiento de memoria intermedia debido a que los datos pasan de la interfaz de usuario a la lógica de negocio.
- Para la prueba de sistema, las pruebas de portabilidad están diseñadas para comprobar si lacapa de presentación funciona en todos los navegadores y dispositivos móviles soportados.
- Para la prueba de integración de sistemas, las pruebas de fiabilidad están diseñadas para evaluar la solidez del sistema en caso de que el microservicio de calificación crediticia falle enresponder.
- Para la prueba de aceptación, las pruebas de usabilidad están diseñadas para evaluar la accesibilidad de la interfaz de procesamiento de crédito bancario para personas con discapacidades.

Los siguientes son ejemplos de prueba de caja blanca:

- Para la prueba de componente, las pruebas están diseñadas para lograr una cobertura completade sentencia y decisión para todos los componentes que realizan cálculos financieros.
- Para la prueba de integración de componentes, las pruebas están diseñadas para practicar cómo cada pantalla de la interfaz del navegador pasa datos a la siguiente pantalla y a la lógica de negocio.
- Para la prueba de sistema, las pruebas están diseñadas para cubrir las secuencias de páginasweb que pueden ocurrir durante una solicitud de línea de crédito.

- Para la prueba de integración de sistemas, las pruebas están diseñadas para practicar todoslos tipos de consulta posibles que se envían al microservicio de calificación crediticia.
- Para la prueba de aceptación, las pruebas están diseñadas para cubrir todas las estructuras de archivos de datos financieros soportados y rangos de valores para transferencias de banco-a- banco.

Por último, los siguientes son ejemplos de pruebas asociadas al cambio:

- Para la prueba de componente, se construyen pruebas de regresión automatizadas para cada componente y se incluyen dentro del marco de integración continua.
- Para la prueba de integración de componentes, las pruebas están diseñadas para confirmar la corrección de defectos relacionados con la interfaz a medida que las correcciones se registran en el repositorio de código.
- Para la prueba de sistema, todas las pruebas de un flujo de trabajo dado se ejecutan de nuevo sicambia alguna pantalla de ese flujo de trabajo.
- Para la prueba de integración de sistemas, las pruebas de la aplicación que interactúa con el microservicio de calificación de crédito se vuelven a ejecutar diariamente como parte del despliegue continuo de ese microservicio.
- Para la prueba de aceptación, todas las pruebas que han fallado previamente se vuelven a ejecutar después de que se haya corregido un defecto encontrado en la prueba de aceptación.

Aunque esta sección proporciona ejemplos de cada tipo de prueba en todos los niveles, no es necesario,para todo software, que cada tipo de prueba esté presente en todos los niveles. Sin embargo, es importanteejecutar los tipos de prueba aplicables en cada nivel, especialmente en el nivel más temprano en el quetiene lugar el tipo de prueba.

5. Prueba de Mantenimiento

Una vez desplegados en los entornos de producción, es necesario mantener el software y los sistemas.Los cambios de diversa índole son casi inevitables en el software y en los sistemas entregados, ya seapara corregir defectos descubiertos en el uso operativo, para añadir nuevas funcionalidades o para eliminar o alterar funcionalidades ya entregadas. El mantenimiento también es necesario para preservar o mejorar las características de calidad no funcionales del componente o sistema a lo largo de su vida útil, especialmente en lo que se refiere a eficiencia de desempeño, compatibilidad, fiabilidad, seguridad, compatibilidad y portabilidad.

Cuando se realizan cambios como parte del mantenimiento, se debe realizar una prueba de mantenimiento, tanto para evaluar el éxito con el que se realizaron los cambios como para comprobar losposibles efectos secundarios (por ejemplo, regresiones) en las partes del sistema que permanecen inalteradas (que suele ser la mayor parte del sistema). La prueba de mantenimiento se centra en probarlos cambios en el sistema, así como en probar las piezas no modificadas que podrían haberse visto afectadas por los cambios. El mantenimiento puede incluir lanzamientos planificados y no planificados (soluciones en caliente).

El lanzamiento de un mantenimiento puede requerir probar el mantenimiento en múltiples niveles de prueba, utilizando varios tipos de prueba, según su alcance. El alcance de la prueba de mantenimientodepende de:

- El grado de riesgo del cambio, por ejemplo, el grado en que el área modificada del software secomunica con otros componentes o sistemas.
- El tamaño del sistema existente.

- El tamaño del cambio.

5.1 Activadores para el Mantenimiento

Existen varias razones por las que el mantenimiento del software y, por lo tanto, la prueba de mantenimiento, se llevan a cabo, tanto para las modificaciones planificadas como para las no planificadas.

Se puede clasificar los activadores de mantenimiento de la siguiente manera:

- Modificación, tales como mejoras planificadas (por ejemplo, basadas en lanzamientos), cambioscorrectivos y de emergencia, cambios en el entorno operativo (tales como actualizaciones previstas del sistema operativo o de la base de datos), actualizaciones del software comercial de distribución masiva ("COTS" por sus siglas en inglés), y parches para los defectos y las vulnerabilidades.
- Migración, por ejemplo, de una plataforma a otra, que puede requerir pruebas operativas del nuevo entorno y del software modificado, o pruebas de conversión de datos cuando los datos de otra aplicación se migren al sistema en mantenimiento.
- Retirada, por ejemplo, cuando una aplicación llega al final de su vida útil.

Cuando se retira una aplicación o sistema, esto puede requerir la comprobación de la migración de datoso el archivo si se requieren largos períodos de retención de datos[26]. También puede ser necesario probar los procedimientos de restauración/recuperación después de archivar durante largos períodos de retención. Además, es posible que sea necesario realizar una prueba de regresión para garantizar que cualquier funcionalidad que permanezca en servicio siga funcionando.

En el caso de los sistemas de Internet de las Cosas, la prueba de mantenimiento puede activarse por laintroducción de cosas completamente nuevas o modificadas, tales como dispositivos de hardware y servicios de software, en el sistema global. La prueba de mantenimiento de estos sistemas hace especial énfasis en la prueba de integración a diferentes niveles (por ejemplo, nivel de red, nivel de aplicación) y en los aspectos de seguridad, en particular los relativos a los datos personales.

5.2 Análisis de Impacto para el Mantenimiento

El análisis de impacto evalúa los cambios que se hicieron para el lanzamiento de un mantenimiento con el objetivo de identificar las consecuencias previstas, así como los efectos secundarios esperados y posibles de un cambio, y para identificar las áreas del sistema que se verán afectadas por el cambio. El análisis de impacto también puede ayudar a identificar el impacto de un cambio en las pruebas existentes. Los efectos secundarios y las áreas afectadas en el sistema necesitan ser probados para las regresiones, posiblemente después de haber actualizado cualquier prueba existente afectada por el cambio.

Se puede realizar un análisis de impacto antes de realizar un cambio, para ayudar a decidir si se deberealizar el cambio, basándose en las consecuencias potenciales en otras áreas del sistema.

El análisis de impacto puede ser difícil si:

- Las especificaciones (por ejemplo, requisitos de negocio, historias de usuario, arquitectura) están desactualizadas o no existen.
- Los casos de prueba no están documentados o están desactualizados.
- No se ha mantenido la trazabilidad bidireccional entre las pruebas y la base de prueba.
- El soporte de herramientas es débil o inexistente.
- Las personas involucradas no tienen conocimientos de dominio y/o sistema.

- No se ha prestado suficiente atención a la mantenibilidad del software durante el desarrollo.