

Ingeniería de software I

**Tema:**

Prueba de software.

**Facilitador:**

Diógenes Amaury Martínez

**Participante:**

Joan Nicolás De León, Roderi Peña, Eric Díaz, Jochimín Contreras y Otec

**Matrícula:**

2019-02433, 2019-03599, 2019-10048, 2019-05041, 2019-04172

**Carrera en curso:**

Ingeniería en software

**Fecha:**

01/09/2020

**Introducción :**

En esta tarea vamos a tratar el tema de la prueba de software para la evaluación de aplicaciones, vamos a ver diferentes tipos de prueba de software y mostraremos ejemplos de cada una de ellas para una mejor comprensión de las mismas.

**Desarrollo :**

**1 - Pruebas de software.**

**Las pruebas de software** son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o stakeholder. Es una actividad más en el proceso de control de calidad.

**Pruebas estáticas**

Son el tipo de pruebas que se realizan sin ejecutar el código de la aplicación.

Puede referirse a la revisión de documentos, ya que no se hace una ejecución de código. Esto se debe a que se pueden realizar "pruebas de escritorio" con el objetivo de seguir los flujos de la aplicación.

**Pruebas dinámicas**

Todas aquellas pruebas que para su ejecución requieren la ejecución de la aplicación.

Las pruebas dinámicas permiten el uso de técnicas de caja negra y caja blanca con mayor amplitud. Debido a la naturaleza dinámica de la ejecución de pruebas es posible medir con mayor precisión el comportamiento de la aplicación desarrollada.

**Pruebas con especificación**

Les sirve a los probadores beta trabajar con el documento de especificación de requerimientos, debemos saber qué son las especificaciones de requerimientos (ESRE).

Las Especificaciones de Requerimientos son un documento clave en el desarrollo de Software. Cuando consideramos los ciclos de vida clásicos, tiene la descripción completa de lo que va a hacer el sistema sin describir cómo lo va a hacer.

Los probadores beta se guían en este documento para validar si el sistema se comporta de la manera que indican las ESRE. Contiene información detallada sobre los requisitos funcionales y no funcionales que el Cliente desea en el sistema.

**Pruebas automáticas**

La automatización de pruebas consiste en el uso de software especial (casi siempre separado del software que se prueba) para controlar la ejecución de pruebas y la comparación entre los resultados obtenidos y los resultados esperados.​ La automatización de pruebas permite incluir pruebas repetitivas y necesarias dentro de un proceso formal de pruebas ya existente o bien adicionar pruebas cuya ejecución manual resultaría difícil.

**Testing Aleatorio**

El testing es una de las tareas más importantes, pero también es compleja y no siempre adoptada correctamente. Las pruebas unitaria, de carga, integración y funcionales son distintos tipos de testing, cada uno con objetivos diferentes y aplicados en diferentes etapas del desarrollo del software. En el testing de unidad se desarrollan pruebas individuales sobre componentes de un sistema.

**Pruebas funcionales**

Una prueba funcional es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software (requisitos funcionales).

Hay distintos tipos como por ejemplo:

* Pruebas unitarias
* Pruebas de componentes
* Pruebas de integración
* Pruebas de sistema
* Pruebas de humo
* Pruebas alpha
* Pruebas beta
* Pruebas de aceptación
* Pruebas de regresión

**Pruebas no funcionales**

Una prueba no funcional es una prueba cuyo objetivo es la verificación de un requisito que especifica criterios que pueden usarse para juzgar la operación de un sistema (requisitos no funcionales) como por ejemplo la disponibilidad, accesibilidad, usabilidad, mantenibilidad, seguridad, rendimiento. Podemos clasificar las pruebas no funcionales según el tipo de requisito no funcional que abarcan:

* Pruebas de compatibilidad
* Pruebas de seguridad
* Pruebas de Estrés
* Pruebas de usabilidad
* Pruebas de rendimiento
* Pruebas de internacionalización y localización
* Pruebas de escalabilidad
* Pruebas de mantenibilidad
* Pruebas de instalabilidad
* Pruebas de portabilidad

**Herramientas para realizar pruebas de software**

El control de la calidad de software lleva consigo aplicativos que permiten realizar pruebas autónomas y masivas permitiendo así la verificación desde el punto de vista estático y de caja blanca, es decir pruebas donde se analiza el software sin ejecutar el software mediante el código fuente del mismo. Podemos encontrar herramientas escritas en software libre, código abierto o software privativo.​ Estas herramientas podrán ser utilizadas para diferentes tipos de pruebas como por ejemplo:

* Herramientas de gestión de pruebas
* Herramientas para pruebas funcionales
* Herramientas para pruebas de carga y rendimiento

**2 - Prueba caja blanca.**

A este tipo de técnicas se le conoce también como Técnicas de Caja Transparente o de Cristal. Este método se centra en cómo diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa. Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento. El objetivo de la técnica es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como falsa. Como se ha indicado ya, puede ser impracticable realizar una prueba exhaustiva de todos los caminos de un programa.

Por ello se han definido distintos criterios de cobertura lógica, que permiten decidir qué sentencias o caminos se deben examinar con los casos de prueba. Estos criterios son:

**-Cobertura de Sentencias:** Se escriben casos de prueba suficientes para que cada sentencia en el programa se ejecute, al menos, una vez.

**-Cobertura de Decisión:** Se escriben casos de prueba suficientes para que cada decisión en el programa se ejecute una vez con resultado verdadero y otra con el falso.

**-Cobertura de Condiciones:** Se escriben casos de prueba suficientes para que cada condición en una decisión tenga una vez resultado verdadero y otra falso.

**-Cobertura Decisión/Condición:** Se escriben casos de prueba suficientes para que cada condición en una decisión tome todas las posibles salidas, al menos una vez, y cada decisión tome todas las posibles salidas, al menos una vez.

**-Cobertura de Condición Múltiple:** Se escriben casos de prueba suficientes para que todas las combinaciones posibles de resultados de cada condición se invoquen al menos una vez.

**-Cobertura de Caminos:** Se escriben casos de prueba suficientes para que se ejecuten todos los caminos de un programa. Entendiendo el camino como una secuencia de sentencias encadenadas desde la entrada del programa hasta su salida.

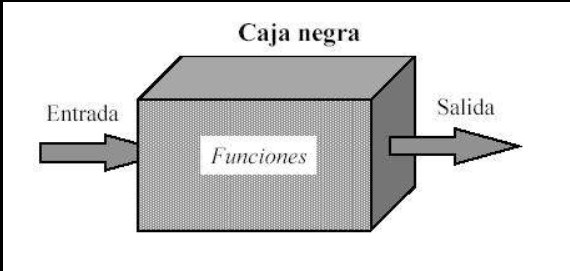
En las pruebas de Caja Blanca, se pretende indagar sobre la estructura interna del código, omitiendo detalles referidos a datos de entrada o salida. Su objetivo principal es probar la lógica del programa desde el punto de vista algorítmico.

Estas se basan en el diseño de Casos de Prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante las pruebas de Caja Blanca el ingeniero de software puede obtener Casos de Prueba que:

* Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
* Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
* Ejecuten todos los bucles en sus límites operacionales.
* Ejerciten las estructuras internas de datos para asegurar su validez.

Las pruebas de Caja Blanca son consideradas entre las más importantes que se aplican a los sistemas, con la que se obtienen como resultados la disminución en un gran porcentaje el número de errores existentes en el software y por ende una mayor calidad y confiabilidad en la codificación.

**3 - Prueba de caja negra.**

****

Centradas principalmente en los requisitos funcionales, permiten obtener un conjunto de condiciones de entrada que ejerciten completamente estos requisitos.

Esto quiere decir que este tipo de prueba se basa en los requisitos de las aplicaciones y en sus especificaciones técnicas, hablando en un lenguaje llano se enfoca en la parte de afuera del software no en la estructura del código, la arquitectura o los detalles relacionados con la implementación de los diferentes módulos, paquetes o rutas en las que se compone el código.

Para ello el tester solo se va a basar en las entradas y salidas de la aplicación, verificando que cada vez que se realice cierta acción, la salida tenga que ser la adecuada según los requerimientos.

Aquí se ignora la estructura de control permitiéndonos concentrarnos en estos requisitos funcionales del sistema y ejercitando los mismos.

Hay que aclarar que este tipos de pruebas no son una alternativa a las pruebas de caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca. Según varios autores expertos en este tema, gracias a estas prueba podremos encontrar :

1. Funciones incorrectas o ausentes.
2. Errores en el diseño de la interfaz.
3. Errores en estructuras de datos o en accesos a las Bases de datos externas.
4. Errores de rendimiento.
5. Errores de inicialización y terminación.

Para prepararlas es necesario que ya tengamos un número de datos que ayuden a la ejecución de estos casos y que permitan que el sistema se ejecute en todas sus variantes, estos datos pueden ser válidos o inválidos para el programa según si lo que se desea es hallar un error o probar una funcionalidad. Estos datos se van a escoger dependiendo de las especificaciones del problema, sin importar los detalles internos del programa con el fin de que el programa corra de manera satisfactoria.

Para desarrollar esta prueba, nos podemos auxiliar de varias técnicas, entre ellas están :

1. **La de Partición de equivalencia :** Esta técnica nos permite dividir el campo de entrada en clases de datos que tiendan a ejercitar determinadas funciones del programa informático.
2. **La del Análisis valores límites :** Esta técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
3. **La de Grafos de Causa - Efecto :** Técnica que le permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Dentro de estas técnicas, la que es considerada como la más efectiva es la de “Partición de equivalencia” ya que permite examinar los valores válidos e inválidos de las entradas existentes en el software, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente va dirigida a la definición de casos de pruebas que descubren clases de errores, reduciendo así el número de clases de prueba que hay que desarrollar.

**¿Cómo se realizan este tipos de pruebas?**

No hay una forma definitiva de cómo se van a realizar estas pruebas ya que cada empresa, organización y tester van a tener su propia estrategia a la hora de aplicarlas, dependiendo del tipo de aplicación o el tiempo asignado a las pruebas, entre otros factores, se realizarán pruebas de caja negra de una forma más intensiva o exploratoria. Aún así podemos tomar en cuenta una serie de pasos a seguir media estandarizada para poder realizar este tipos de pruebas de manera efectiva :

1. Debemos de hacer un previo análisis a los requisitos y especificaciones del software.
2. El tester va a diseñar un cuadro de entradas válidas, también llamado escenario de prueba positiva, esto con el objetivo de verificar si el software las procesa correctamente. También se va a diseñar un cuadro de entradas no válidas llamado escenario de prueba negativa para comprobar si el software es capaz de detectarlas y reaccionar ante estas entradas.
3. Basándose en las entradas, el tester determinará para cada una de ellas las salidas esperadas correspondientes.
4. Una vez obtenidas las entradas y salidas correspondientes, se procede a diseñar los casos de pruebas.
5. Se ejecutan esos casos de pruebas.
6. El tester comprueba la salida que ha emitido el software con la salida esperada de los casos de pruebas.
7. Si la salida del software coincide con la salida esperada, el software hace lo que tiene que hacer para esa entrada. Pero si la salida no coincide con la salida esperada, se ha encontrado un defecto en el software, lo que llevará a la posterior solución del mismo.

**Clases de pruebas de caja negra :**

**Clase de equivalencias**

1. Consisten en diseñar y clasificar entradas de datos para una funcionalidad similar del software, esperando que sean procesados de la misma manera.
2. Estas clases se pueden definir tanto para datos válidos como inválidos.
3. La forma de clasificación se define según criterios en función de las salidas de datos, valores internos, funcionalidades, eventos, etc.
4. Según esos datos se van a diseñar casos de pruebas para cubrir todos o parte de los datos válidos e inválidos.

**Análisis de valores límites**

1. Estos se valen de las pruebas de clave de equivalencias para obtener datos límites que acoten las posibles casuísticas de los datos. Basándose en criterios como la probabilidad que se presenten más errores.
2. Los valores máximos y mínimos son los valores límites.
3. Se pueden aplicar para datos válidos e inválidos.
4. Se diseña un caso de prueba por cada valor límite.
5. Es una forma de identificar defectos de forma óptima y eficiente.

**Tablas de decisiones**

Son una herramienta fundamental para la documentación de las reglas de negocio que conllevan una alta complejidad. Se crean a partir de las especificaciones funcionales y de las reglas del negocio. Aquí muchas veces las entradas y salidas se van a representar como valores booleanos ( v o f).

Estas tablas contienen secuencias de condiciones encadenadas con la combinación de los valores booleanos para cada entrada de datos, así como su resultado esperado de cada combinación. Las columnas de las tablas de decisiones se corresponde con una de las reglas del negocio que se representan con la combinación de las condiciones y de sus resultados.

**Transición entre estados**

1. Se basan en los diferentes estados de un software.
2. Se representan en un diagrama de transición entre estados.
3. Este diagrama permite al tester visualizar información importante sobre el software, como pueden ser las transiciones, las entradas, salidas o eventos encadenados.
4. Una tabla de estado nos permite ver las relaciones entre los estados del software y las entradas de datos. Nos ayuda a ver posibles transiciones válidas.

**Pruebas de casos de uso**

Se basan en la representación de interacción entre actores, estos son los usuarios o sistemas que van a interactuar con nuestro software. Basándonos en estas interacciones podremos hacer casos de pruebas, estas suelen ir acompañadas de precondiciones que deben cumplirse para que los actores funcionen de forma adecuada.

Cada caso de uso termina con postcondiciones que van a ser los resultados analizados después de la ejecución.

Las pruebas de casos de uso se suelen usar para definir las pruebas de aceptación en la que participan usuarios o clientes.

**Pruebas de historias de usuario**

Muy utilizada en las metodologías ágiles como la Scrum. Los requerimientos de usuarios son preparados como historias de usuario. Estas historias describen una funcionalidad que puede ser desarrollada o probada en una misma secuencia.

En estas descripciones en las historias de usuarios suelen aparecer funcionalidades a implementar, requerimientos no funcionales y los criterios de aceptación.

Los criterios de aceptación abarcan la cobertura mínima requerida para la historia de usuario. En estos mismos criterios se basan los casos de pruebas.

**Relación con el ciclo de vida del software**

Las pruebas de caja negra también tienen su propio ciclo de vida llamado “Software Testing Life Cycle”, y dependen de cada etapa del ciclo de vida del software. Esto nos quiere dejar dicho que en cada una de las etapas de este ciclo de vida, nosotros vamos a poder realizar distintos tipos de pruebas.

**4 - Ejemplo de prueba de caja negra.**

**Ejemplo 1:**

**Envió de correo electrónico al registrarse una transacción**

Caso: El sistema enviará un correo electrónico cuando se registre alguna de las siguientes transacciones: pedido de venta de cliente, despacho de mercancía al cliente, emisión de factura al cliente y registro de cobro al cliente.

**Técnica de pruebas de caja negra: Requerimiento funcional / Caso de uso**

Caso 1.1: Datos de entrada: Registrar pedido de venta. Resultado esperado (Salida): El sistema envía un correo electrónico al cliente como constancia de que su pedido se ha recibido.

Caso 1.2: Datos de entrada: Registrar despacho de mercancía al cliente. Resultado esperado (Salida): El sistema envía un correo electrónico al cliente como constancia de que se ha realizado el despacho.

Caso 1.3: Datos de entrada: Registrar factura de cliente. Resultado esperado (Salida): El sistema envía un correo electrónico al departamento de facturación y al cliente.

Caso 1.4: Datos de entrada: Registrar cobro. Resultado esperado (Salida): El sistema envía un correo electrónico al departamento de cuentas por cobrar y al agente comercial (vendedor) que lleva la cuenta del cliente

**Ejemplo 2:**

**Campo de texto que solo acepta caracteres alfabéticos**

Caso: Se tiene un campo de texto que solo acepta caracteres alfabéticos. La longitud del valor ingresado debe estar entre 6 y 10 caracteres.

Técnica de pruebas de caja negra: Partición de equivalencias.

Usando partición de equivalencias, se pueden establecer tres particiones, longitudes entre 0 y 5 caracteres (partición inválida), longitudes entre 6 y 10 caracteres (partición válida), y longitudes mayores a 10 caracteres (partición inválida). Además, el ingreso de caracteres no alfabéticos (por ejemplo un número), se considera también dato inválido.

Caso 2.1: Datos de entrada: cadena de 5 caracteres. Resultado esperado (Salida): La aplicación no permite el ingreso del dato y muestra un mensaje de error.

Caso 2.2: Datos de entrada: cadena de 7 caracteres, incluyendo uno o más caracteres no alfabéticos. Resultado esperado (Salida): La aplicación no permite el ingreso del dato y muestra un mensaje de error.

Caso 2.3: Datos de entrada: cadena de 7 caracteres, solo de caracteres alfabéticos. Resultado esperado (Salida): La aplicación permite el ingreso del dato

Caso 2.4: Datos de entrada: cadena de 11 caracteres. Resultado esperado (Salida): La aplicación no permite el ingreso del dato y muestra un mensaje de error.

**Ejemplo 3:**

**Ingreso de datos en un campo numérico**

Caso: Supongamos que tenemos una aplicación que posee una pantalla para el ingreso de un valor numérico, como por ejemplo un monto (en alguna moneda), cuyo valor debe estar entre 1 y 1.000. Por lo tanto, todo valor menor que 1 y mayor a 1.000 es invalido.

Técnica de pruebas de caja negra: Partición de equivalencias y análisis de valores borde.

Caso 3.1: Datos de entrada: 450. Resultado esperado (Salida): La aplicación permite el ingreso del dato.

Caso 3.2: Datos de entrada: 1 (Valor borde). Resultado esperado (Salida): La aplicación permite el ingreso del dato.

Caso 3.3: Datos de entrada: 1.000 (Valor borde). Resultado esperado (Salida): La aplicación permite el ingreso del dato.

Caso 3.4: Datos de entrada: 0. Resultado esperado (Salida): La aplicación no permite el ingreso del dato y muestra un mensaje de error.

Caso 3.5: Datos de entrada: 1.001. Resultado esperado (Salida): La aplicación no permite el ingreso del dato y muestra un mensaje de error.

1. **Ejemplo de prueba de caja blanca.**

**CASOS DE PRUEBA CAJA BLANCA**

Prueba de la Ruta Básica.

Permite conocer una medida de la complejidad lógica de un diseño procedural y usar esta medida como guía para definir un conjunto básico de rutas de ejecución. Estas garantizan que se ejecute cada instrucción del programa por lo menos una vez durante la prueba.

**Conclusión :**

Al concluir esta tarea se ha podido aprender sobre las diferentes pruebas de software que existen y su importancia en este mundo del desarrollo de software, vimos las pruebas de caja negra y de caja blanca y cómo cada una de estas pruebas son importantes para la eficiencia y calidad de nuestro software ya que estos tienen la finalidad de decir qué hace y cómo funciona a la hora de interactuar con nuestro software que nos van a servir para poder probar nuestros productos en las dos áreas que engloban el desarrollo del mismo, la interfaz y el código.