1.- Introducción

Durante todo el proceso de desarrollo de software, desde la fase de diseño, en la implementación y una vez desarrollada la aplicación, es necesario realizar un conjunto de pruebas, que permitan verificar que el software que se está creando, es correcto y cumple con las especificaciones impuesta por el usuario.

En el proceso de desarrollo de software, nos vamos a encontrar con un conjunto de actividades, donde es muy fácil que se produzca un error humano.

Estos errores humanos pueden ser: una incorrecta especificación de los objetivos, errores producidos durante el proceso de diseño y errores que aparecen en la fase de desarrollo.

Mediante la realización de pruebas de software, se van a realizar las tareas de verificación y validación del software antes de su puesta en marcha.

La verificación es la comprobación de que un sistema o parte del mismo, cumple con las condiciones impuestas. Con la verificación se comprueba si la aplicación se está construyendo correctamente.

La validación es el proceso de evaluación del sistema o de uno de sus componentes, para determinar si satisface los requisitos especificados.

2.- Técnicas de diseño de casos de prueba

Un caso de pruebas es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollado para conseguir un objetivo particular o condición de prueba. Para llevar a acabo un caso de prueba, es necesario definir las precondiciones y post-condiciones, identificar unos valores de entrada y conocer el comportamiento que deberia tener el sistema ante dichos valores. Tras realizar ese análisis e introducir dichos datos en el sistema, se observa si su comportamiento es el previsto o no y por qué. De esta forma se determina si el sistema ha pasado o no la prueba.

Para llevar a cabo el diseño de casos de prueba se utilizan dos técnicas: prueba de caja blanca y prueba de caja negra.

2.1.- Prueba de caja blanca

Prueba de la Caja Blanca (White Box Testing): en este caso, se prueba la aplicación desde dentro, usando su lógica de aplicación. Se centran en validar la estructura interna del programa y para ello necesitan conocer los detalles procedimentales del código. Esta prueba va a probar directamente el código de la aplicación, para poder analizar los resultados de las pruebas.

También se la conoce como prueba estructural o de caja de cristal. Se basa en el minucioso examen de los detalles procedimentales del código de la aplicación. Mediante esta técnica se pueden obtener casos de prueba que:

- Garanticen que se ejecutan al menos una vez todos los caminos independientes de cada módulo
- Ejecuten todas las sentencias al menos una vez.
- Ejecuten todas las decisiones lógicas en su parte verdadera y en su parte falsa.
- Ejecuten todos los bucles en sus limites.
- Utilicen todas las estructuras de datos internas para asegurar su validez.

Este tipo de pruebas, se basan en unos criterios de cobertura lógica, cuyo cumplimiento determina la mayor o menor seguridad en la detección de errores.

Los criterios de cobertura que se siguen son:

Cobertura de sentencias: se han de generar casos de pruebas suficientes para que cada instrucción del programa sea ejecutada, al menos, una vez.

Cobertura de decisiones: se trata de crear los suficientes casos de prueba para que cada opción resultado de una prueba lógica del programa, se evalúe al menos una vez a cierto y otra a falso.

Cobertura de condiciones: se trata de crear los suficientes casos de prueba para que cada elemento de una condición, se evalúe al menos una vez a falso y otra a verdadero.

Cobertura de condiciones y decisiones: consiste en cumplir simultáneamente las dos anteriores.

Cobertura de caminos: es el criterio más importante. Establece que se debe ejecutar al menos una vez cada secuencia de sentencias encadenadas, desde la sentencia inicial del programa, hasta su sentencia final. La ejecución de este conjunto de sentencias, se conoce como camino. Como el número de caminos que puede tener una aplicación, puede ser muy grande, para realizar esta prueba, se reduce el número a lo que se conoce como camino prueba.

Cobertura del camino de prueba: Se pueden realizar dos variantes, una indica que cada bucle se debe ejecutar sólo una vez, ya que hacerlo más veces no aumenta la efectividad de la prueba y otra que recomienda que se pruebe cada bucle tres veces: la primera sin entrar en su interior, otra ejecutándolo una vez y otra más ejecutándolo dos veces.

Una de las técnicas utilizadas para desarrollar los casos de prueba de caja blanca es la prueba del camino básico.

2.2.- Pruebas de caja negra

Prueba de la Caja Negra (Black Box Testing): cuando una aplicación es probada usando su interfaz externa, sin preocuparnos de la implementación

de la misma. Aquí lo fundamental es comprobar que los resultados de la ejecución de la aplicación, son los esperados, en función de las entradas que recibe.

Una prueba de Caja Negra se lleva a cabo sin tener que conocer ni la estructura, ni el funcionamiento interno del sistema. Cuando se realiza este tipo de pruebas, solo se conocen las entradas adecuadas que deberá recibir la aplicación, así como las salidas que les correspondan, pero no se conoce el proceso mediante el cual la aplicación obtiene esos resultados.

Se trata de probar, si las salidas que devuelve la aplicación, o parte de ella, son las esperadas, en función de los parámetros de entrada que le pasemos. No nos interesa la implementación del software, solo si realiza las funciones que se esperan de él.

También se la conoce como prueba funcional o prueba de comportamiento. Su principal cometido, va a consistir, en comprobar el correcto funcionamiento de los componentes de la aplicación informática. Para realizar este tipo de pruebas, se deben analizar las entradas y las salidas de cada componente, verificando que el resultado es el esperado. Solo se van a considerar las entradas y salidas del sistema, sin preocuparnos por la estructura interna del mismo.

Si por ejemplo, estamos implementando una aplicación que realiza un determinado cálculo científico, en el enfoque de las pruebas funcionales, solo nos interesa verificar que ante una determinada entrada a ese programa el resultado de la ejecución del mismo devuelve como resultado los datos esperados. Este tipo de prueba, no consideraría, en ningún caso, el código desarrollado, ni el algoritmo, ni la eficiencia, ni si hay partes del código innecesarias, etc.

Con este tipo de prueba se intenta encontrar errores de las siguientes categorías:

- Funcionalidades incorrectas o ausentes.
- Errores de interfaz
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento
- Errores de inicialización y finalización.

Existen diferentes técnicas para confeccionar los casos de prueba de caja negra, entre los que destaco: clases de equivalencia, analisis de valores limite, métodos basados en grafos, pruebas de comparación, ...

3.- Estrategias de prueba del software

La estrategia de prueba del software se puede implementar siguiendo el modelo en espiral. Comenzaría con la prueba de unidad que se centra en

la unidad más pequeña de software, el módulo tal y como esté implementado en código fuente. La prueba avanza hasta llegar a la prueba de integración, donde se toman los módulos probados mediante la prueba de unidad y se construye una estructura de programa que esté de acuerdo con lo que dicta el diseño. El foco de atención es el diseño.

Avanzamos llegando a la prueba de validación (o de aceptación), donde se prueba el software en el entorno real de trabajo con intervención del usuario final y se validan los requisitos establecidos como parte del análisis de requisitos del software, comparándolos con el sistema que ha sido construido. Por último llegamos a la prueba del sistema donde se verifica que cada elemento encaja de forma adecuada y se alcanza la funcionalidad y rendimiento total. Se prueba como un todo el software y otros elementos del sistema.

3.1.- Prueba de unidad

En este nivel se prueba cada unidad o módulo con el objetivo de eliminar errores en la interfaz y en la lógica interna. Esta actividad utiliza técnicas de caja negra y caja blanca según convenga para lo que se desea probar. Se realizan pruebas sobre:

- La interfaz del módulo , para asegurar que la información fluye adecuadamente.
- Las estructuras de datos locales, para asegurar que mantienen su integridad durante todos los pasos de ejecución del programa.
- Las condiciones límite, para asegurar que funciona correctamente en los límites establecidos durante el proceso.
- Todos los caminos independientes de la estructura de control, con el fin de asegurar que todas las sentencias se ejecutan al menos una vez.
- Todos los caminos de manejo de errores.

3.2.- Prueba de integración

En este tipo de prueba se observa como interaccionan los distintos módulos. Se podría pensar que esta prueba no es necesaria, ya que si todos los módulos funcionan por separado, también deberían funcionar juntos. Realmente el problema está aquí, en comprobar si funcionan juntos.

Existen dos enfoques fundamentales para llevar a cabo las pruebas:

- **Integración no incremental o big bang**. Se prueba cada módulo por separado y luego se combinan todos de una vez y se prueba todo el programa completo. En este enfoque se encuentran gran cantidad de errores y la corrección se hace difícil.
- **Integración incremental**. El programa completo se va construyendo y probando en pequeños segmentos, en este caso los errores son más fáciles de localizar. Se dan dos estrategias Ascendente y Descendente. En la integración Ascendente la construcción y prueba del programa empieza

desde los módulos de los niveles más bajos de la estructura el programa. En la Descendente la integración comienza en el módulo principal (programa principal) moviéndose hacia abajo por la jerarquía de control.

3.3.- Prueba de validación

La validación se consigue cuando el software funciona de acuerdo con las expectativas razonables del cliente definidas en el documento de especificación de requisitos del software o ERS. Se llevan a cabo una serie de pruebas de caja negra que demuestran la conformidad con los requisitos.

Las técnicas a utilizar son:

- **Prueba Alfa**. Se lleva a cabo por el cliente o usuario en el lugar de desarrollo. El cliente utiliza el software de forma natural bajo la observación del desarrollador que irá registrando los errores y problemas de uso.
- **Prueba Beta**. Se lleva a cabo por los usuarios finales del software en su lugar de trabajo. El desarrollador no está presente. El usuario registra todos los problemas que encuentra, reales y/o imaginarios, e informa al desarrollador en los intervalos definidos en el plan de prueba. Como resultado de los problemas informados el desarrollador lleva a cabo las modificaciones y prepara una nueva versión del producto.

3.4.- Prueba del sistema

La prueba del sistema está formada por un conjunto de pruebas cuya misión es ejercitar profundamente el software. Son las siguientes:

- **Prueba de recuperación**. En este tipo de prueba se fuerza el fallo del software y se verifica que la recuperación se lleva a cabo apropiadamente.
- **Prueba de seguridad**. Esta prueba intenta verificar que el siustema está protegido contar accesos ilegales.
- **Prueba de resistencia (Stress)**. Trata de enfrentar el sistema con situaciones que demandan gran cantidad de recursos, por ejemplo, diseñando casos de prueba que requieran el máximo de memoria, incrementando la frecuencia de datos de entrada, que den problemas en un sistema operativo virtual, …

4.- Documentación para las pruebas

Son las siguientes:

- **Plan de pruebas**. Describe el alcance, el enfoque, los recursos y el calendario de las actividades de prueba. Identifica los elementos a probar, las características que se van a probar, las tareas que se van a realizar, el personal responsable de cada tarea y los riesgos asociados al plan.
- **Especificaciones de prueba**. Están cubiertas por tres tipos de documentos: la especificación del diseño de la prueba (se identifican los requisitos, casos de prueba y procedimientos de prueba necesarios para

llevar a cabo las pruebas y se especifica la función de los criterios de pasa no-pasa), la especificación de los casos de prueba (documenta los valores reales utilizados para la entrada, junto con los resultados previstos), y la especificación de los procedimientos de prueba (donde se identifican los pasos necesarios para hacer funcionar el sistema y ejecutar los casos de prueba especificados).

- **Informes de pruebas**. Se definen cuatro tipo de documentos: un informe que identifica los elementos que están siendo probados, un registro de las pruebas (donde se registra lo que ocurre durante la ejecución de la prueba), un informe de incidentes de prueba (describe cualquier evento que se produce durante la ejecución de la prueba que requiere mayor investigación) y un informe resumen de las actividades de prueba.

5.- Pruebas de código

La prueba de código consiste en la ejecución del programa (o parte de él) con el objetivo de encontrar errores. Se parte para su ejecución de un conjunto de entradas y una serie de condiciones de ejecución; se observan y registran los resultados y se comparan con los resultados esperados. Se observará si el comportamiento del programa es el previsto o no y por qué.

Para las pruebas de código, se van a mostrar diferentes técnicas que dependerán del tipo de enfoque utilizado: de caja blanca, se centran en la estructura interna del programa; o de caja negra, más centrado en las funciones, entradas y salidas del programa.

5.1.- Prueba del camino básico

La prueba del camino básico es una técnica de prueba de caja blanca que permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Para la obtención de la medida de la complejidad lógica (o complejidad ciclomática) emplearemos una representación del flujo de control denominada grafo de flujo o grafo del programa.

5.2.- Partición o clases de equivalencia

La partición equivalente es un método de prueba de caja negra que divide los valores de los campos de entrada de un programa en clases de equivalencia. Por ejemplo, supongamos un campo de entrada llamado numero de empleado, definido con una serie de condiciones: numérico de tres dígitos y el primero no puede ser cero. Entonces se puede definir una clase de equivalencia no válida: numero de empleado > 100; y otra válida:

numero de empleado entre 100 y 999.

Para identificar las clases de equivalencia se examina cada condición de entrada (son parte del dominio de valores de la entrada y normalmente son una frase en la especificación) y se divide en dos o más grupos. Se definen dos tipos de clases de equivalencia:

- Clases válidas: son los valores de entrada válidos
- Clases no válidas: son los valores de entrada no válidos.

Las clases de equivalencia se definen según una serie de directrices:

- Si una condición de entrada especifica un rango se define una clase de equivalencia válida y dos no válidas
- Si una condición de entrada requiere un valor especifico, se define una clase de equivalencia válida y dos no válidas
- Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una no válida
- Si una condición de entrada es lógica, se define una clase de equivalencia válida y una no válida

5.3.- Análisis de valores limite

El análisis de valores limite se basa en que los errores tienden a producirse con mas probabilidad (por razones que no estn del todo claras) en los limites o extremos de los campos de entrada.

Esta técnica complementa a la anterior y los casos de prueba elegidos ejercitan los valores justo por encima y por debajo de los margenes de la clase de equivalencia. Además, no solo se centra en las condiciones de entrada, sino que también se exploran las condiciones de salida definiendo las clases de equivalencia de salida.

Las reglas son las siguientes:

- Si una condición de entrada especifica un rango de valores, se deben diseñar casos de prueba para los limites del rango y para los valores justo por encima y por debajo del rango. Por ejemplo: si una entrada requiere un rango de valores enteros comprendidos entre 1 y 10, hay que escribir casos de prueba por el valor 1, 10, 0 y 11.
- Si una condición de entrada especifica un número de valores, se deben diseñar casos de prueba que ejerciten los valores máximo, mínimo, un valor justo por encima del máximo y un valor justo por debajo del mínimo. Por ejemplo, si el programa requiere de dos a diez datos de entradas, hay que escribir casos de prueba para 2,10, 1 y 11 datos de entrada.
- Aplicar la regla 1 para la condición de salida. Por ejemplo, si se debe aplicar sobre un campo de salida un descuento de entre un 10 % mínimo y un 50 % máximo (dependiendo del tipo de cliente); se generarán casos de prueba para 9,99%, 10%, 50% y 50,01%.
- Usar la regla 2 para la condicion de salida. Por ejemplo, si la salida de

un programa es una tabla de temperaturas de 1 a 10 elementos , se deben diseñar casos de prueba para que la salida del programa produzca 0, 1, 10 y 11 elementos. Tanto en esta regla, como en la anterior hay que tener en cuenta que no siempre se podrán generar resultados fuera del rango de salida.

- Si las estructuras de datos internas tienen limites preestablecidos (por ejemplo un array de 100 elementos), hay que asegurarse de diseñar casos de prueba que ejercite la estructura de datos en sus límites, primer y último elemento.