ÍNDICE

1. Planificación de las pruebas

Durante todo el proceso de desarrollo de software, desde la fase de análisis hasta la implantación en el cliente, es habitual incurrir en errores de varios tipos: incorrecta especificación de los objetivos, errores producidos en el diseño o errores en la fase de desarrollo.

Por lo tanto, se hace necesario hacer un conjunto de pruebas que permitan comprobar que el producto que se está creando, es correcto y cumple con las especificaciones solicitadas por el usuario.

Las pruebas tratan de **verificar y validar** las aplicaciones, entendiendo estos términos como:

- La **verificación** es la comprobación de que un sistema o parte de un sistema, 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.

Para llevar a cabo el proceso de pruebas, de manera adecuada, se definen estrategias de pruebas.

Siguiendo el **modelo en espiral**, las pruebas empezarán con las **pruebas unitarias** de cada porción de código.

Una vez pasadas estas pruebas con éxito, se seguirá con las **pruebas de integración**, donde se ponen todas las partes del código en común, el ensamblado de los bloques de código y sus pruebas atienden a los establecido durante la fase de diseño.

El siguiente paso será la **prueba de validación**, donde se comprueba que el sistema construido cumple con lo establecido en el análisis de requisitos de software.

Finalmente se alcanza la **prueba de sistema** que verifica el funcionamiento total del software y otros elementos del sistema.

Notas:

- El objetivo de las pruebas es conseguir un **software libre de errores**, por lo tanto, la detección de defectos en el software se considera un éxito en esta fase.

- **El programador debe evitar probar sus propios programas**, ya que aspectos no considerados durante la codificación podrán volver a pasar inadvertidos en las pruebas si son tratados por la misma persona.

2. Pruebas unitarias

Las pruebas unitarias tienen por objetivo validar el correcto funcionamiento de un módulo de código. El fin que se persigue, es que **cada módulo funcione correctamente por separado**.

Posteriormente, con la prueba de integración, se podrá asegurar **el correcto funcionamiento del sistema**.

Una unidad es la parte de la aplicación más pequeña que se puede probar. En programación procedural, una unidad puede ser una función o procedimiento. En programación orientada a objetos, una unidad es normalmente un método. En el diseño de los casos de pruebas unitarias, **habrá que tratar que cumpla las siguientes características**:

- **Automatizable**: en lo posible no debería requerirse una intervención manual.
- **Completas**: deben cubrir la mayor cantidad de código.
- **Repetibles o reutilizables**: no se deben crear pruebas que sólo puedan ser ejecutadas una vez.
- **Independientes**: la ejecución de una prueba no debe afectar a la ejecución de otra.
- **Profesionales**: las pruebas deben ser consideradas igual que el desarrollo del código, con la misma profesionalidad, documentación, etc.

El objetivo de las pruebas unitarias es aislar cada parte del programa y demostrar que las partes individuales son correctas. Las pruebas individuales nos proporcionan **cinco ventajas básicas**:

1. **Fomentan el cambio**: las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura, puesto que permiten hacer pruebas sobre los cambios y así asegurar que las modificaciones no han introducido errores.

2. **Simplifican la integración**: puesto que permiten llegar a la fase de integración con un alto grado de seguridad de que el código está funcionando correctamente.

3. **Documentan el código**: las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.

4. **Separación de la interfaz y la implementación**: dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.

5. **Los errores están más acotados y son más fáciles de localizar**: dado que tenemos pruebas unitarias que pueden desenmascararlos.

2.1. Procedimientos y casos de prueba

Según el IEEE, un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollados para un objetivo particular como, por ejemplo, ejercitar un camino concreto de un programa o verificar el cumplimiento de un determinado requisito, incluyendo toda la documentación asociada.

Dada la complejidad de las aplicaciones informáticas que se desarrollan en la actualidad, es prácticamente **imposible probar todas las combinaciones** que se pueden dar dentro de un programa o entre un programa y las aplicaciones que pueden interactuar con él. Por este motivo, en el **diseño** de los casos de

prueba, siempre es necesario asegurar que se obtiene un nivel aceptable de probabilidad de que se detectarán los errores existentes.

Las pruebas deben buscar un **compromiso** entre la cantidad de **recursos** que se consumirán en el proceso de prueba, y la **probabilidad** de que se detecten los errores existentes.

- Los procedimientos de prueba **identifican los casos de prueba y han de definir los resultados** esperados para dichos casos.

- Tras la ejecución de las pruebas unitarias, una vez obtenidos los resultados del programa, habrá que **comparar y analizar** los datos esperados y los obtenidos para concluir si el programa está libre de errores o por el contrario precisa ser actualizado.

2.2. Tipos de pruebas

No existe una clasificación oficial o formal, sobre los diversos tipos de pruebas de software. En la ingeniería del software, nos encontramos básicamente con tres enfoques fundamentales:

- **Pruebas de la caja negra o funcionales.**

  La aplicación es probada de acuerdo a 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.

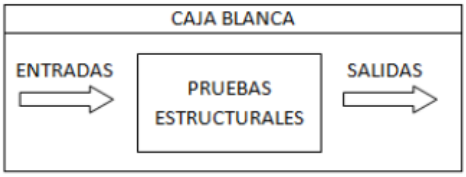
  Una prueba de tipo caja negra se lleva a cabo sin tener necesidad de conocer la estructura interna del sistema. Cuando se realiza este tipo de pruebas, sólo se conocen las entradas adecuadas que deberá recibir la aplicación, así como las salidas que les correspondan, pero no se conoce el proceso mediante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

  En el siguiente ejemplo se detecta un error mediante las pruebas de caja negra. Aunque las pruebas de caja blanca dan un resultado correcto (estructura condicional bien construida), se entiende que tener una nota superior a 5 debería dar el mensaje de superado y viceversa. Los mensajes están intercambiados.

```
if( iNota< 5)
{ System.out.println("Enhorabuena. Superado."); }
else if ( iNota >= 5)
{ System.out.println("La proxima vez sera"); }
```

- **Pruebas de la caja blanca o estructurales.**

  En este caso, se prueba la aplicación desde dentro, usando su lógica de aplicación.



Una prueba de caja blanca **va a analizar y probar directamente el código de la aplicación, intentando localizar estructuras incorrectas o ineficientes en el código.** Como se deriva de lo anterior, para llevar a cabo una prueba de caja blanca, **es necesario un conocimiento del código** en detalle.

A continuación, se muestra un código que presenta un error detectable mediante las pruebas de caja blanca.

```
int iNota = 3;

if( iNota< 5)
{ System.out.println("La proxima vez sera"); }
else if ( iNota >= 5)
{ System.out.println("Enhorabuena. Superado."); }
else
{ System.out.println("Codigo que no se ejecuta"); }
```

Aunque el programa dará el resultado esperado para cualquier nota (comprobación que se lleva a cabo con las pruebas de caja negra-funcionales), presenta un error en su estructura, el else nunca es alcanzado. De este error nos daremos cuenta gracias a las pruebas de caja blanca.

- **Enfoque aleatorio.**

  Una tercera colección de casos de prueba se puede determinar a partir del llamado enfoque aleatorio, que consiste en utilizar modelos que representen las posibles entradas al programa, para crear a partir de ellos los casos de prueba. En estas pruebas se intenta simular la entrada habitual que va a recibir el programa, para ello se crean datos entrada en la secuencia y con la frecuencia en que podrían aparecer. Es habitual utilizar generadores automáticos de casos de prueba.

  Se trata de una técnica menos sistemática, que será utilizada por programadores más experimentados.

2.3. Pruebas de regresión

Durante el proceso de prueba, tendremos éxito si detectamos un posible fallo o error. La consecuencia directa de ese descubrimiento, supone la modificación del componente donde se ha detectado. Estos cambios pueden generar errores colaterales, que no existían antes. La modificación realizada nos obliga a repetir pruebas que hemos realizado con anterioridad.

Las pruebas de regresión se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error, como para realizar una mejora. No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también es necesario controlar que las modificaciones no produzcan efectos negativos sobre el mismo u otros componentes.

El conjunto de pruebas de regresión contiene tres clases diferentes de pruebas:

1. Una muestra representativa de pruebas que ejercite **diversas funciones del software**.

2. Pruebas adicionales que se centran en las funciones del software que se van a ver probablemente **afectadas por el cambio**.
3. Pruebas que se centran en los componentes del software que **han cambiado**.

El diseño de las pruebas de regresión podrá ser una combinación de casos de uso obtenidos desde los enfoques de caja blanca, caja negra y aleatoria.

3. Documentación de la prueba

Como en otras etapas y tareas del desarrollo de aplicaciones, **la documentación de las pruebas es indispensable**. Unas pruebas bien documentadas podrán también servir como base de conocimiento en futuras tareas.

Las metodologías actuales, como Métrica v.3, proponen que la documentación de la fase de pruebas se base en los estándares ANSI / IEEE sobre verificación y validación de software.

El propósito de los estándares ANSI/IEEE es describir un conjunto de documentos para las pruebas de software. Un documento de pruebas estándar facilita la comunicación entre desarrolladores al suministrar un marco de referencia común.

Acompañando las distintas etapas en el ciclo de vida de los proyectos, se van generando los siguientes **documentos** relacionados con las pruebas:

1. **Plan de Pruebas**. Al principio se desarrollará una planificación general de las pruebas a nivel de sistema, que quedará reflejada en el "Plan de Pruebas". El plan de pruebas forma parte del Análisis del Sistema.

2. **Especificación del diseño de pruebas**. De la ampliación y detalle del plan de pruebas, surge el documento "Especificación del diseño de pruebas". Determina pautas de pruebas para los bloques en los que ha sido dividido el programa en la fase de diseño.

3. **Especificación de un caso de prueba**. Los casos de prueba se concretan a partir de la especificación del diseño de pruebas. Fundamentalmente, se nutre de los casos obtenidos para las pruebas de caja blanca y caja negra. Fase de implementación.

4. **Especificación de procedimiento de prueba**. Una vez determinados los casos de prueba, será preciso detallar el modo en que van a ser ejecutados cada uno de ellos, quedando recogidos en el documento "Especificación del procedimiento de prueba". También deberá incluir los resultados esperados en cada caso de prueba. Determina las pruebas unitarias a realizar.

5. **Registro de pruebas**. En el "Registro de pruebas" se recogen los sucesos que tengan lugar durante las pruebas.

6. **Informe de incidente de pruebas**. Para cada incidente, defecto detectado, solicitud de mejora, etc. se elaborará un "Informe de incidente de pruebas". Se determinan los incidentes mediante análisis comparativo de los resultados esperados y los obtenidos.

4. Normas de calidad

Los estándares a los que principalmente se ha ido haciendo referencia en la fase de prueba de software son:

- Metodología Métrica v3.
- Estándares BSI
  - BS 7925-1, Pruebas de software. Parte 1. Vocabulario.
  - BS 7925-2, Pruebas de software. Parte 2. Pruebas de los componentes software.
- Estándares IEEE de pruebas de software:
  - IEEE estándar 829, Documentación de la prueba de software.
  - IEEE estándar 1008, Pruebas de unidad. Otros estándares ISO / IEC 12207, 15289.

Sin embargo, estos estándares no cubren determinadas facetas de la fase de pruebas, como son la organización del proceso y gestión de las pruebas y presentan pocas pruebas funcionales y no funcionales. Ante esta problemática, la industria ha desarrollado la norma ISO/IEC 29119. La norma ISO/IEC 29119 de prueba de software, pretende unificar todos los estándares, de forma que proporcione vocabulario, procesos, documentación y técnicas para cubrir todo el ciclo de vida del software.

La norma ISO/IEC 29119 se compone de las siguientes partes:

- **Parte 1.** Conceptos y vocabulario.
  - Introducción a la prueba.
  - Pruebas basadas en riesgo.
  - Fases de prueba (unidad, integración, sistema, validación) y tipos de prueba (estática, dinámica, no funcional, ...). Prueba en diferentes ciclos de vida del software.
  - Roles y responsabilidades en la prueba.
  - Métricas y medidas.

- **Parte 2**. Procesos de prueba.
  - Política de la organización.
  - Gestión del proyecto de prueba.
  - Procesos de prueba estática.
  - Procesos de prueba dinámica.

- **Parte 3**. Documentación.
  - Contenido.
  - Plantilla.

- **Parte 4**. Técnicas de prueba.
  - Descripción y ejemplos.
  - Estáticas: revisiones, inspecciones, etc.
  - Dinámicas: caja negra, caja blanca, técnicas de prueba no funcional (seguridad, rendimiento, usabilidad, etc).

5. Validación

En el proceso de validación, interviene de manera decisiva el **cliente**. Hay que tener en cuenta, que estamos desarrollando una aplicación para terceros, y que son éstos los que deciden si la aplicación se ajusta a lo establecido en el análisis.

La validación del software se consigue mediante una serie de **pruebas de caja negra** que demuestran la conformidad con lo acordado (requisitos).

Se genera un plan de prueba que traza la clase de pruebas que se han de llevar a cabo, y un procedimiento de prueba que define los casos de prueba específicos a realizar.

Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisfacen todos los requisitos: funcionales, de rendimiento, de portabilidad, de compatibilidad, de recuperación de errores, de facilidad de mantenimiento etc.