UD3: INTRODUCCION A LAS PRUEBAS DE SOFTWARE Y DEPURACIÓN

DISEÑO Y REALIZACIÓN DE PRUEBAS

Las pruebas son necesarias en la fabricación de cualquier producto industrial y, de forma análoga, en el desarrollo de proyectos informáticos.
¿Quién pondría a la venta una aspiradora sin estar seguro que aspira correctamente, o una radio digital sin haber comprobado que pueda sintonizar los canales? ¿Cuándo hay que llevar a cabo las pruebas?¿Qué hay que probar?
Todas las fases establecidas en el desarrollo del software son importantes. La falta o mala ejecución de alguna de ellas puede provocar que el resto del proyecto arrastre uno o varios errores que serán determinantes para su éxito. Cuanto antes se detecte un error, menos costoso será de solucionar.

DEFINICIÓN

La prueba (testing) es el proceso de ejecutar un programa con la intención de encontrar errores.
En la literatura existen otras definiciones. Por ejemplo, de acuerdo a la IEEE [IEEE90] el concepto de prueba se define como “El proceso de operar un sistema o un componente bajo unas condiciones
específicas, observando o registrando los resultados obtenidos con el fin de realizar a posteriori una evaluación de ciertos aspectos clave” descubrir un error es el éxito de una prueba. Con el fin de desarrollar las pruebas de una manera óptima, los casos de prueba cobran especial importancia.
La IEEE [IEEE90] define caso de prueba como “un conjunto de entradas, condiciones de ejecución y resultados esperados que son desarrollados con el fin de cumplir un determinado objetivo”.
La prueba exhaustiva del software es impracticable (no se pueden probar todas las posibilidades de su funcionamiento ni siquiera en programas sencillos).
El programa se prueba ejecutando solo unos pocos casos de prueba dado que por lo general es física, económica o técnicamente imposible ejecutarlo para todos los valores de entrada posibles de aquí la frase de Dijkstra. “Las pruebas del software pueden usarse para demostrar la existencia de errores, nunca su ausencia”
El principal objetivo de las pruebas es el de encontrar errores en el programa, tantos como sea posible.
Entorno al concepto de error existen distintos términos que poseen ciertos aspectos en común y a menudo inducen a confusión. Por ello, el estándar IEEE [IEEE90] establece tres conceptos:

DEFECTO, FALLO, ERROR

• Defecto o falla (bug): características que pueden causar que un componente o sistema no realice las funciones requeridas, por ejemplo una sentencia o una definición de datos incorrectas. Un defecto potencialmente origina un fallo. Coincide con la definición de diccionario, "imperfección".
• Fallo (failure): La manifestación física o funcional que se produce al ejecutar un programa con un defecto el cual es incapaz de funcionar correctamente.
• Error (error): acción humana que produce resultados incorrectos. Es una equivocación cometida por el desarrollador. Algunos ejemplos son: un error de digitación, una malinterpretación de un requerimiento o de la funcionalidad de un método. Coincide con la definición de diccionario de error como "una idea falsa o equivocada".

RELACIÓN DEFECTO, FALLO, ERROR

El error cometido inyecta un defecto que puede provocar un fallo en el sistema. No todos los defectos corresponden a un fallo. Por ejemplo: si un trozo de código defectuoso nunca se ejecuta, el defecto nunca provocará el fallo del código.
Sin embargo, el término error tiene varias acepciones:

- Diferencia entre la salida actual de un software y la correcta.

TIPOS DE DEFECTOS

En algoritmos
- Faltas típicas o bifurcar a destiempo o
- Preguntar por la condición equivocada
- No inicializar variables
- No evaluar una condición particular
- Comparar variables de tipos no adecuados

De sintaxis
- Confundir un 0 por una O
- Los compiladores detectan la mayoría

De precisión o de cálculo
- Formulas no implementadas correctamente
- No entender el orden correcto de las operaciones
- Faltas de precisión como un truncamiento no esperado

De documentación
- La documentación no es consistente con lo que hace el Software. Ejemplo: El manual de usuario tiene un ejemplo que no funciona en el sistema

De estrés o sobrecarga
- Exceder el tamaño máximo de un área de almacenamiento intermedio.
- Ejemplos:
    - El sistema funciona bien con 100 usuarios pero no con 110 Sistema que funciona bien al principio del día y se va degradando paulatinamente el desempeño hasta ser espantoso al caer la tarde.
    - Defecto: había tareas que no liberaban memoria

De capacidad o de borde
- Más de lo que el sistema puede manejar
- Ejemplos:
    - El sistema funciona bien con importes < 1000000
    - Año 2000. sistema trata bien fechas hasta el 31/12/99

De sincronización o coordinación
- No cumplir requerimiento de tiempo o frecuencia.
- Ejemplo: comunicación entre procesos con fallos
-

De capacidad de procesamiento o desempeño
- No terminar el trabajo en el tiempo requerido
- Tiempo de respuesta inadecuado

De recuperación
- No poder volver a un estado normal luego de una falla

De estándares o procedimientos
- No cumplir con la definición de estándares y/o procedimientos

De hardware o software del sistema
- Incompatibilidad entre componentes

LAS PRUEBAS EN EL CICLO DE VIDA DE DESARROLLO DE SOFTWARE

En cada una de las fases del ciclo de vida de un proyecto, será necesario que el trabajo llevado a cabo sea validado y verificado. Existe una correspondencia entre cada nivel de prueba y el trabajo realizado en cada etapa del desarrollo:

- Verificación: ¿El software está de acuerdo con su especificación? El papel de la verificación comprende comprobar que el software cumple los requisitos funcionales y no funcionales de su especificación.
- Validación: ¿El software cumple las expectativas del cliente? La validación es un proceso más general. Se debe asegurar que el software hace lo que el usuario espera.

Existen muchos tipos de pruebas que deben cubrir las especificaciones de un proyecto informático a través de los procedimientos y de los casos de prueba.

- Pruebas unitarias: una prueba unitaria es la manera de comprobar el correcto funcionamiento de un componente de código (una parte de un programa). Esto nos permite asegurar que todos los componentes del sistema desarrollado funcionen correctamente por separado. Detectan errores en los datos, lógica y algoritmos. Participan los programadores
- Pruebas de Integración: Una vez que se hayan probado que los componentes funcionan correctamente en forma aislada se procede a probar cómo funcionan integrados con otros. El objetivo es detectar errores de interfaces y relaciones entre componentes. Participan programadores.
- Pruebas de sistema: Esta prueba tiene como objetivo verificar que se han integrado adecuadamente todos los elementos del sistema (hardware y software) y que realizan las funciones adecuadas. Permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen. Participan probadores y analistas
- Pruebas de aceptación: El propósito es confirmar que el sistema está terminado, que desarrolla puntualmente las necesidades del cliente y que es aceptado por los usuarios finales. Participan probadores, analistas y cliente.

PLANIFICACIÓN DE LAS PRUEBAS

A la vez que se avanza en el desarrollo del software se van planificando las pruebas que se harán en cada fase del proyecto. Esta planificación se concretará en un plan de pruebas que se aplicará a cada producto desarrollado. Cuando se detectan errores en un producto se debe volver a la fase anterior para depurarlo y corregirlo. Es importante tener presente que, cuanto antes se detecte un error, más fácil será contrarrestar y solucionar este error. El coste de la resolución de un problema crece exponencialmente a medida que avanzan las fases del proyecto en las que se detecte.

UD3: INTRODUCCION A LAS PRUEBAS DE SOFTWARE Y DEPURACIÓN

TÉCNICAS DE DISEÑO DE CASOS DE PRUEBAS

- Las pruebas de caja blanca permitirán recorrer todos los posibles caminos del código y ver qué sucede en cada caso posible. Se probará qué ocurre con las condiciones y los bucles que se ejecutan. Las pruebas se llevarán a cabo con datos que garanticen que han tenido lugar todas las combinaciones posibles. Para decidir qué valores deberán tomar estos datos es necesario saber cómo se ha desarrollado el código, buscando que no quede ningún rincón sin revisar.
- Las técnicas de pruebas de caja negra pretenden encontrar errores en funciones incorrectas o ausentes, errores de interfaz, errores de rendimiento, inicialización y finalización. Se centra en las funciones y en sus entradas y salidas.

Un buen caso de prueba es aquel que tiene una probabilidad muy alta de descubrir un nuevo error.

CASOS DE PRUEBA

A partir del plan de pruebas se deberán especificado las partes de código a tratar, en qué orden habrá que hacer las pruebas, quien las hará y mucha información más. Ahora sólo falta entrar en detalle, especificando el caso de prueba para cada una de las pruebas a realizar.

Un caso de prueba define como se llevaran a cabo las pruebas, especificando entre otros: el tipo de pruebas, la entradas de las pruebas, los resultados esperados o las condiciones bajo las que se tendrán que desarrollar.

Los casos de pruebas sirven para identificar los errores que hay en el software para que éstos no lleguen al usuario final. Estos errores pueden encontrarse como defectos en la interfaz de usuario, en la ejecución de estructuras de datos o un determinado requisito funcional.
A la hora de diseñar los casos de prueba, no sólo se debe validar que la aplicación hace lo que se espera ante entradas correctas, sino que también se debe validar que tenga un comportamiento estable ante entradas no esperadas, informando del error.

Los casos de prueba siguen el siguiente ciclo de vida:

- Definición de los casos de prueba.
- Creación de los casos de prueba.
- Selección de los valores para las pruebas.
- Ejecución de los casos de prueba.
- Comparación de los resultados obtenidos con los resultados esperados.

Cada caso de prueba deberá ser independiente de los demás, tendrá un comienzo y un final muy marcado y deberá almacenar toda la información referente a su definición, creación, ejecución y validación final.

A continuación se indican algunas informaciones que debería contemplar cualquier caso de prueba:

- Identificador del caso de prueba.
- Módulo o función a probar.
- Descripción del caso de prueba detallado.
- Entorno que se deberá cumplir antes de la ejecución del caso de prueba.
- Datos necesarios para el caso, especificando sus valores.
- Tareas que ejecutará el plan de pruebas y su secuencia.
- Resultado esperado.
- Resultado obtenido.
- Observaciones o comentarios después de la ejecución.
- Responsable del caso de prueba.
- Fecha de ejecución.
- Estado (finalizado, pendiente, en proceso).

Todo caso de prueba está asociado, como mínimo, a un procedimiento de prueba.

Los procedimientos de prueba especifican como se podrán llevar a cabo los casos de prueba o parte de estos de forma independiente o de forma conjunta, estableciendo las relaciones entre ellos y el orden en que deberán atender.

DEFINICIÓN Y EJEMPLO DE CASO DE PRUEBA

1.CASO DE PRUEBA: Es un conjunto de entradas de prueba, condiciones de ejecución y resultados esperados. Tiene un objetivo concreto (probar algo). Caso de prueba, Entrada, Condiciones de ejecución, Resultado esperado: no deja entrar y cambia el registro, Objetivo del caso de prueba,

2.PROCEDIMIENTO DE PRUEBA: Pasos que hay que llevar a cabo para probar uno (o varios) casos de prueba. ¿Cómo probar el caso de prueba y verificar si ha tenido éxito?

UD3: INTRODUCCION A LAS PRUEBAS DE SOFTWARE Y DEPURACIÓN

DEPURACIÓN DE CÓDIGO FUENTE

Una técnica muy importante para la ejecución de las pruebas y, en general, para los programadores, es la depuración del código fuente. Los depuradores (debuggers) son aplicaciones o herramientas permiten la ejecución controlada de un programa o un código, siguiendo las instrucciones ejecutadas observando los estados intermedios de las variables y los datos implicados para facilitar la localización los defectos (bugs).

La depuración ocurre como consecuencia de una prueba. En el proceso de depuración se evalúan los resultados puesto que se detecta una discrepancia entre los resultados esperados y los observados.

DEPURACIÓN

Los procedimientos que están vinculados a la depuración del código fuente son:

- Identificar la casuística para poder reproducir el error.
- Diagnosticar el problema.
- Solucionar el error atacando el problema.
- Verificar la corrección del error y que no se han introducido nuevos errores en el resto del software. Volver a probar

La depuración del código es una herramienta muy útil si se sabe localizar la parte del código fuente donde se encuentra un determinado error.
Si el error es difícil de reproducir será muy complicado encontrar una solución para solucionarlo. Por ello, acotándolo dentro del código, se puede ir localizando, haciendo pruebas para llegar a las circunstancias que lo reproducen.
Muchas veces una modificación en el código para solucionar un problema puede generar otro error que esté relacionado o que no tenga nada que ver. Habrá que volver a confirmar la correcta ejecución del código una vez finalizada la corrección.

La depuración de código permite la creación de un punto en el código fuente hasta el cual el software será ejecutado de forma directa. Cuando la ejecución haya llegado a este punto de ruptura, se podrá avanzar en la ejecución del código paso a paso hasta encontrar el error que se busca.

Otra forma de llevar a cabo la depuración de código es ir atrás, desde el punto del error, hasta encontrar el causante.
Esta táctica se utiliza en casos más complejos y no es tan habitual, pero será muy útil en el caso de no tener detectada la ubicación del error, que puede encontrarse oculto en alguna parte del código fuente.
En estos casos, a partir del lugar donde se genera el error, se llevará a cabo un recorrido hacia atrás, yendo paso a paso en sentido inverso.

FRAMEWORKS PARA PRUEBAS UNITARIAS

Las pruebas unitarias se ocupan por probar los programas/clases individuales para asegurarse de que cada programa/clase se ejecute según las especificaciones. Existen frameworks para realizar pruebas unitarias prácticamente en cualquier lenguaje de programación. En el caso de Java podemos incluso elegir entre varias opciones.
Antes de la llegada de estos frameworks, los programadores probaban sus programas imprimiendo en la consola o en un archivo de seguimiento expresiones de prueba. Este enfoque no es satisfactorio porque requiere el criterio humano para analizar los resultados producidos.

JUNIT

Una de estos frameworks es JUnit, puede que el más consolidado para realizar pruebas unitarias en Java. Este es el enlace a la página principal del framework. JUnit no está incluido en JDK, pero está incluido en la mayoría de los IDEs como Eclipse y NetBeans. Usaremos la versión JUnit 4.

ESCRIBIENDO CASOS DE PRUEBA. PRUEBAS UNITARIAS. LAS MEJORES PRÁCTICAS.

¿Cómo probar un programa para hacerlo correctamente? Hay dos técnicas: prueba de caja blanca y prueba de caja negra.

- Las pruebas de caja blanca inspeccionan los códigos del programa y prueban la lógica del programa.
- Las pruebas de caja negra no inspeccionan los códigos del programa, sino que miran la entrada-salida y tratan el programa como un recuadro negro.

PRUEBAS DE INTEGRACIÓN

¿Son suficientes las pruebas que se hacen encada parte de una aplicación para asegurarnos de que se ha validado el funcionamiento del software? La respuesta es no; es necesario validar también los diferentes módulos combinados.
Una vez se han probado los componentes individuales del programa y se ha garantizado que no contienen errores, habrá que integrarlos a fin de crear un sistema completo que también deberá ser probado. Este es el nivel de las pruebas de integración.

Un objetivo importante de las pruebas de integración es localizar errores en las interfaces entre las diferentes unidades. Además, las pruebas de integración sirven para validar que las partes de código que ya han sido probadas de forma independiente continúen funcionando correctamente al ser integradas

PRUEBAS DE SISTEMA

Las pruebas de sistema servirán para validar la aplicación una vez este haya sido integrada con el resto del sistema del usuario. Aunque la aplicación ya haya sido validada de forma independiente, a las pruebas de sistema se llevará a cabo una segunda validación con la aplicación ya integrada en su entrono de trabajo real.

Algunos tipos de pruebas para desarrollar durante las pruebas del sistema:

- Pruebas de rendimiento: valorarán los tiempos de respuesta de la nueva aplicación, el espacio que ocupará en disco, el flujo de datos que generará a través de un canal de comunicación.
- Pruebas de resistencia: valorarán la resistencia de la aplicación para determinadas situaciones del sistema.
- Pruebas de robustez: valorarán la capacidad de la aplicación para soportar varias entradas no correctas.
- Pruebas de seguridad: ayudarán a determinar los niveles de permisos de los usuarios, las operaciones que podrán llevar a cabo y las de acceso al sistema ya los datos.
- Pruebas de usabilidad: determinarán la calidad de la experiencia de un usuario en la manera de interactuar con el sistema.
- Pruebas de instalación: indicarán las operaciones de arranque y de actualización de los softwares.

PRUEBAS DE ACEPTACIÓN

Estas pruebas las hace el cliente o usuario final de la aplicación desarrollada. Son básicamente pruebas funcionales, sobre el sistema completo, y buscan una cobertura de la especificación de requisitos y del manual del usuario.
Estas pruebas no se realizan durante el desarrollo, ya que sería impresentable de cara al cliente, sino una vez pasadas todas las pruebas anteriores por parte del desarrollador o el equipo de tests.

El objetivo de la prueba de aceptación es obtener la aprobación del cliente sobre la calidad de funcionamiento del sistema desarrollado y probado.

La experiencia demuestra que, aún después del proceso más preciso de pruebas por parte del desarrollador y el equipo de trabajo, quedan una serie de errores que sólo aparecen cuando el cliente pone en funcionamiento la aplicación o el sistema desarrollado.

Sea como sea, el cliente siempre tiene la razón. Por este motivo, muchos desarrolladores ejercitan unas técnicas denominadas pruebas alfa y pruebas beta .

- ✓ Las pruebas alfa consisten en invitar al cliente que venga en el entorno de desarrollo a probar el sistema. Se trabaja en un entorno controlado y el cliente siempre tienen un experto a mano para ayudarle a usar el sistema y analizar los resultados.

- ✓ Las pruebas beta vienen después de las pruebas alfa, y se desarrollan en el entorno del cliente, un entorno que esta fuera de control para el desarrollador y el equipo de trabajo. Aquí el cliente se queda solo con el producto y trata de encontrar los errores, de los que informará al desarrollador.

Las pruebas alfa y beta son habituales en productos que se venderán a muchos clientes o que utilizarán muchos usuarios. Algunos de los compradores potenciales se prestan a estas pruebas, ya sea para ir entrenando su personal con tiempo, ya sea en compensación de algún ventaja económica (mejor precio sobre el producto acabado, derecho a mantenimiento gratuito, a nuevas versiones ...).
La experiencia muestra que estas prácticas son muy eficaces. En un entorno de desarrollo de software tienen sentido cuando la aplicación o sistema para desarrollar el utilizará un gran número de usuarios finales (empresas grandes con diferentes departamentos que tendrán que utilizar esta nueva herramienta).

EJECUCIÓN DE LAS PRUEBAS

Después de la planificación de los procedimientos de pruebas y del diseño de los casos de pruebas, el siguiente paso será el proceso de ejecución. La ejecución de las pruebas comportará seguir los siguientes pasos:

1. Ejecución de las pruebas.
2. Comprobación de si se ha producido algún error en la ejecución de las pruebas.
3. Si no ha habido ningún error:
   - Se verifica la finalización de las pruebas.
   - Se valida si se necesitan pruebas adicionales.

- Si se necesitan pruebas adicionales, hay que validar que no existan condiciones anormales. Si hay condiciones anormales, se finaliza el proceso de pruebas haciendo una evaluación del mismo proceso; si no, habrá depurar las pruebas.
- Si no se necesitan pruebas adicionales, se llevará a cabo una finalización del proceso de pruebas haciendo una evaluación del mismo proceso.

4. En el caso de haber encontrado errores en la ejecución de las pruebas, habrá que ver si estos errores han sido debidos a:
   - Un defecto del software. En este caso, la ejecución de las pruebas ha cumplido su objetivo y habrá depurar el código de programación, localizar el o los errores y solucionarlo para volver al punto inicial, en el que se volverán a ejecutar las pruebas y se volverá a validar si el cambio efectuado ha sido exitoso.
   - Un defecto del diseño de las pruebas. En este caso, habrá que revisar las pruebas que se han ejecutado, depurándose las, localizando el o los errores y solucionarlo, por tener unas pruebas correctas, sin errores, listas para volver al punto inicial y volver a ejecutarlas.

Una ejecución de las pruebas exitosa no es aquella que encuentra errores a toda costa ni aquella que sólo evalúa dos o tres posibilidades, sino que es aquella que cumple lo planificado al plan de pruebas y que garantiza que lo diseñado sea el que se valide.

HERRAMIENTAS PARA LA REALIZACIÓN DE PRUEBAS

Las herramientas CASE para la realización de pruebas ayudarán mucho poder automatizar la tarea de ejecutar las pruebas y algunos de los otros procesos que hay que hacer para implementarlas (planificación, diseño ...).
Tal como sucede con las herramientas específicas para el desarrollo de software, también existen herramientas específicas para la ayuda a los procedimientos de pruebas.

Las herramientas de pruebas del software se pueden clasificar según muchos criterios:

- en función del o de los lenguajes de programación a que se apoya,
- en función de si son privativos o de código abierto, o en función, por ejemplo, del tipo de pruebas que permiten llevar a cabo.

Se debe considerar que la gran mayoría de los IDE llevan integradas herramientas que permiten la depuración del código fuente. Algunas de ellas también permiten el desarrollo de pruebas o el hecho de añadir algún módulo que lo permita.

A continuación se enumeran algunas herramientas que permiten el desarrollo de pruebas en función del tipo de pruebas:

1. Pruebas unitarias:

JUnit . Automatiza las pruebas unitarias y de integración. Provee clases y métodos que facilitan la tarea de efectuar pruebas en el sistema para asegurar la consistencia y funcionalidad del software desarrollado.

2. Pruebas estáticas de código:

PMD . Puede ser integrado a varias herramientas: JDeveloper, Eclipse, jEdit, etc. Permite encontrar en el código errores en el manejo de excepciones, código muerto, código sin optimizar, código duplicado ...
FindBugs . Puede integrarse en Eclipse. Efectúa un escaneo de código mediante el cual encuentra errores comunes, malas prácticas de programación, código vulnerable, rendimiento, seguridad ...
YASCA . Permite encontrar vulnerabilidades de seguridad, calidad en el código, rendimiento ... Aprovecha la funcionalidad de los conectores FindBugs, PMD y Jlint.

3. Pruebas de rendimiento:

JMeter. Permite efectuar pruebas de rendimiento, de estrés, de carga y de volumen, sobre recursos estáticos o dinámicos.
OpenSTA. Permite captar las peticiones del usuario generadas en un navegador web, luego guardarlas, y poderlas editar para su posterior uso.
WEbLoad. permite llevar a cabo pruebas de rendimiento, a través de un entorno gráfico en el que se pueden desarrollar, grabar y editar script de pruebas.
Grinder. es un framework (entorno de trabajo) escrito en Java, con el que se pueden efectuar pruebas de rendimiento, por medio de script escritos en lenguaje Jython. Permite grabar las peticiones del cliente sobre un navegador web para ser luego reproducido.

4. Pruebas de aceptación:

Fitness. Permite comparar lo que tiene que hacer el software con el que realmente hace. Se pueden efectuar pruebas de aceptación y pruebas de reglas de negocio.
Aviñón. Permite a los usuarios expresar pruebas de aceptación de una forma no ambigua antes de que comience el desarrollo. Trabaja en conjunto con JUnit, HTTPUnit

UD3: INTRODUCCION A LAS PRUEBAS DE SOFTWARE Y DEPURACIÓN

CALIDAD DEL SOFTWARE

La calidad puede definirse como: “Una característica o atributo de un elemento que permite juzgar su valor”.
La calidad de un producto se refiere a las características que se pueden comparar con estándares conocidos como longitud, color, propiedades eléctricas, maleabilidad, etc. En el software entre estas características se incluyen la complejidad ciclomática, el número de puntos de función, las líneas de código, etc.
Por calidad de software entendemos el grado con el que un sistema, componente o proceso cumple con los requerimientos especificados y las necesidades o expectativas del cliente o usuario.

- Validación: El proceso de validación tiene como objetivo determinar si el sistema final cumplen los objetivos para los que se construyó el producto, respondiendo de este modo a la pregunta: ¿El producto es el correcto?

- Verificación: El proceso de verificación intenta determinar si los productos software se ajustan a los requisitos o a las condiciones impuestas para su elaboración. De este modo la pregunta a la que responde este proceso es ¿Se está construyendo el producto correctamente?