Unidad 3

Diseño y realización de pruebas

Entornos de desarrollo

3

Índice

Introducción

Cuando se ha desarrollado un software, realizar pruebas que nos aseguren que no habrá ningún tipo de fallo es casi imposible a la par que muy costoso, por eso, a no ser que el software sea de vital importancia, se harán pruebas que determinen que el programa no tenga fallos específicos y de gran envergadura.

Las pruebas tienen como función que se convenza a desarrolladores y cliente de que el software va a ser operativo en su mayoría y que cumplirá con los objetivos planteados en el diseño.

Cuanto más exhaustivas sean las pruebas que se realice de un software, más difícil será que haya un fallo y por tanto mayor fiabilidad habrá.

Al finalizar esta unidad

- Conoceremos la importancia de la fase de pruebas para el software.
- Habremos comprendido los conceptos de procedimientos y casos de pruebas que tipos de pruebas tiene un software.
- Sabremos cual es la diferencia entre una prueba de caja blanca y otra de caja negra.
- Seremos capaces de ver ampliamente de que consta un desarrollo de software al completo.

3.1. Procedimientos de pruebas y casos de prueba

El procedimiento de prueba detalla que queremos conseguir realizando una prueba, que se va a probar y como se va a realizar la prueba.

Cuando hacemos una prueba no siempre la hacemos con la intención de que detectemos errores, sino que además de esto, que el rendimiento sea aceptable y que cumpla con todos los requisitos mínimos para los que se desarrolla dicho software.

Esto se traduce en que, aunque en las pruebas no haya errores, eso no quiere decir que se hayan pasado.

En el diseño de los procedimientos también se decide que personas hacen las pruebas y como las van a realizar.

Es muy recomendable que ciertas pruebas las realice personal externo al equipo de programación para detectar errores que a lo mejor los desarrolladores pasaron por alto.

Los planes de prueba es un documento en el que se detallan las pruebas a realizar y que cubre normalmente los siguientes aspectos:

1. **Introducción.** Una pequeña introducción al software desarrollado.
2. **Módulos o partes del software por probar.** Se detallan estas partes o módulos.
3. **Características del software por probar.** Individual y en conjunto.
4. **Características del software que no ha de probarse.**
5. **Enfoque de las pruebas.** Aquí se detallan los responsables de las pruebas, la planificación de dichas pruebas, cuanto van a durar, etc.
6. **Criterios de validez o invalidez del software.** En esta fase detallamos que criterios ha de cumplir el software para ser aprobado como válido.
7. **Proceso de pruebas.** Proceso y especificación de las pruebas a realizar.
8. **Requerimientos del entorno.** Comunicaciones, herramientas, etc. Necesidades para la realización de pruebas.
9. **Homologación o aprobación del plan.** Firma de los interesados en el software y responsables del proyecto para aprobar este plan de prueba.

Las demás fases del proceso de pruebas, es cumplir con este plan preestablecido.

3.1.1. Casos de prueba

En la fase de pruebas de un software diseñamos y preparamos los casos de prueba creados para encontrar fallos en el programa desarrollado.

Lo más importante aquí es no malgastar el tiempo en redundancias, es decir, que cada prueba se enfoque a una parte del software en especial, porque si probamos algo y funciona, no hay necesidad de probar esto de nuevo.

Las pruebas no deben de constar de amplía complejidad, porque entonces puede que no se encuentre el origen del fallo, pero tampoco muy sencilla, pues podría no detectar los fallos.

El diseño de los casos de prueba es muy importante, porque en la mayoría de los casos, los fallos que se encontrarán son los que se van buscando y no otros, es por eso que debemos de contemplar varias posibilidades y prestar atención a esta parte del diseño de pruebas.

3.1.2. Codificación y ejecución de las pruebas

Cuando se hayan diseñado los casos de prueba la infraestructura debe de adecuarse para que podamos realizar estos casos. En la mayoría de los casos, deberán de ser codificados dando lugar a set o conjunto de datos. En dichos sets, los que se va a almacenar son los datos válidos e inválidos y ciertos datos que se encuentran fuera de rango o disparatados.

Además, las máquinas sobre las que se efectuarán las pruebas deberán de contar con recursos y software necesario.

3.2. Tipos de pruebas: funcionales, estructurales y regresión

El tipo de pruebas que podemos realizar es muy variado, y a continuación, vamos a ver las más frecuentes.

El primer lote de pruebas que se usa son las pruebas funcionales que buscando que los componentes del software funcionen de acuerdo con el objetivo de su diseño.

Lo que se va a testear aquí son las funcionalidades o características que ofrece el sistema, es decir, que hace.

El realizador de pruebas o tester se basará en la documentación que se haya elaborado durante el desarrollo del software para la realización de las pruebas además de solicitar que ciertos usuarios finales del aplicativo también testeen el software puesto que ellos saben que debe de ejecutar.

Estas últimas son las llamadas de caja negra. Aquí no evaluamos el funcionamiento interno del sistema, sino de nuevo, que funciones realiza este sistema.

Las pruebas de seguridad también son funcionales, y se ve que fallos de seguridad pueden acarrear las funciones del sistema sobre todo cuando este interactúa con otros sistemas, de los que puede haber fallos de compatibilidad.

Aunque siguen siendo pruebas de caja negra, existen las pruebas no funcionales, que se encargan de comprobar el funcionamiento interno del sistema.

Este tipo de pruebas son pruebas de estrés, de fiabilidad, etc.

Hay otro tipo de pruebas, que son de caja blanca, que se basan en técnicas de análisis de código y que constan de pruebas estructurales, donde se examina la aplicación y su arquitectura de manera más profunda.

También existen las pruebas de regresión o repetidas que son las que se realizan repitiendo la prueba, pero cuando se ha modificado algo en la aplicación para comprobar que los cambios no han afectado al software.

Este tipo de pruebas no se usa para el descubrimiento de errores sino para asegurarnos de que no existen ciertos errores.

Las pruebas de regresión suelen estar automatizadas y agrupadas en conjuntos a los que llamamos regression teest suites.

3.3. Pruebas de caja blanca

Estas pruebas, que también se denominan clear box testing, consisten en emplear técnicas de análisis del código fuente del programa para detectar errores en las distintas partes de este.

Hay varias clases de este tipo de pruebas, como pueden ser:

- Pruebas de cubrimiento.
- Pruebas de condiciones.
- Pruebas de bucles.

3.3.1. Pruebas de cubrimiento

Cuando usamos este tipo de pruebas lo que se persigue es ejecutar al menos una vez, todas las sentencias que se encuentren el código del programa.

Esto no siempre es cien por cien posible, porque hay veces en las que por lo que sea, el código no se va a ejecutar nunca debido a la expresión almacenada.

Pero, además, también se puede dar el caso en el que haya excepciones de errores de código que nunca van a fallar. Esto es el llamado código sin errores.

Si queremos realizar estas pruebas, deberemos de poder generar los suficientes casos de prueba como para que todos los caminos del código se puedan albergar.

3.3.2. Prueba de condiciones

Cuando tenemos una condición en las sentencias que conforman un programa informático, son múltiples las elecciones que tenemos, y por lo tanto también tendrán que ser múltiples sus pruebas. De manera lógica, debe de haber una prueba por cada resultado que pueda dar la condición.

3.3.3. Prueba de bucles

Los bucles o las iteraciones son estructuras basadas en la repetición de ciertas sentencias que terminan cuando se llega a un resultado concreto o que se mantienen en el tiempo siempre que cierto valor no cambie. Esto hace que necesitemos que la prueba de bucles se base en la repetición que el bucle contemple.

En el caso de un bucle simple, los casos de prueba deben de contemplar lo siguiente:

- Repetir el máximo, máximo -1 y máximo +1 veces el bucle para comprobar que el resultado del código funciona como queremos.
- Repetir el bucle cero y una vez.
- Repetir el bucle un número determinado de veces.

Cuando tengamos bucles anidados, tendremos bucles internos y externos. Habrá que realizar las pruebas contempladas en un bucle simple para cada bucle, pero dependiendo siempre del bucle externo que indicará la repetición de los bucles internos.

3.4. Pruebas de caja negra

Las pruebas de caja negra, que son las que solo comprueban el funcionamiento de la aplicación sin comprobar el código, son las siguientes:

- Pruebas de cubrimiento.
- Pruebas de clases de equivalencia de datos.
- Pruebas de valores límite.

3.4.1. Pruebas de clases de equivalencia de datos

Mientras que probamos un software, podemos imaginar que hay que generar un usuario y una clave para acceso al sistema.

Por temas de seguridad, tanto usuario como contraseña deben de tener mayúsculas y minúsculas, letras y caracteres alfanuméricos. Además, una extensión mínima de 8 caracteres.

Entonces, si lo que queremos es probar el funcionamiento de esto, debemos de crear clases de equivalencia tanto para el código de usuario como para la contraseña. Pero, también tenemos que ver clases inválidas para asegurarnos que esto no se acepta.

Un ejemplo sería:

- Usuario:
  - Clases válidas: “Pepe123#”
  - Clases inválidas: “Jose”
- Contraseña:
  - Clases válidas: “$Cambiame321”
  - Clases invalidas: “1234567890”

3.4.2. Prueba de valores límites

Estas pruebas se complementan a las pruebas de particiones y tienen como objetivo comprobar que los valores que debe de recibir la aplicación son los correctos para comprobar su funcionamiento. Por ejemplo, en una aplicación de cita médica, si solo nos deja solicitar una cita al día, podemos probar la solicitud de dos citas, y si esto funciona, quiere decir que realmente hay un error. También se podría hacer en una calculadora que solo nos deje introducir números positivos y del uno al mil. Si le insertamos valores negativos o mayores de mil, al estar fuera de rango, también debería de fallar. No obstante, también se deben de probar valores válidos para asegurarnos de que funciona bien.

El objetivo de este tipo de pruebas es comprobar que los límites se hayan establecido correctamente en las comparaciones.

3.4.3. Prueba de interfaces

Las GUI o interfaces de usuario deben de probarse con lo general con la denominada prueba de interfaces.

Por lo general, una interfaz es una serie de objetos con ciertas propiedades en conjunto.

Cada vez que hacemos clic o introducimos valores en las interfaces lo que pasa internamente es que los objetos que la componen van adquiriendo ciertos valores durante la ejecución del programa.

Dependiendo de los valores que la interfaz reciba desde la entrada, la salida será de un modo u otro, pero en estas pruebas debe de darse la salida esperada. Por eso, es importante conocer la funcionalidad del programa previamente al testeo.

Como testear una interfaz

El primer consejo que se da a la hora de testear una interfaz es realizar todo lo que figure en el manual de usuario para su funcionamiento. Lo mejor en este punto es que se trabaje con valores reales para comprobar que las salidas son las esperadas.

Si se llega a este tipo de prueba con el software es común que el testeo sea más serio puesto que está cerca de instalarse en producción.

Testear la usabilidad

En este testeo se busca comprobar que el software que se ha generado cumple con los resultados que espera el usuario final.

Aquí se trabaja como lo va a hacer el usuario final usando datos reales para su comprobación.

Este tipo de testeo tiene, sobre todo, la finalidad de que el usuario se encuentre cómodo en su manejo, pues solo él nos va a indicar si se trata de una aplicación tediosa o dificultosa de usar. Si esto es así, se realizarán los cambios oportunos para su fácil manejo, pues el cliente final es quien tiene que usar dicho software y resultarle lo más cómodo posible.

Los test de usabilidad se encuentran generalmente integrados en el ciclo de vida de desarrollo de software.

La usabilidad no se debe de tener en cuenta únicamente cuando se realicen las pruebas, sino también en el diseño de la interfaz, por eso, es necesario que antes de realizar el diseño nos planteemos las siguientes cuestiones:

- ¿Los usuarios comprenderán cómo funciona la interfaz de una manera sencilla?
- ¿Es la interfaz lo suficiente rápida y eficiente para el trabajo del usuario?

Todo lo que se pueda añadir para mejor la usabilidad como atajos de teclado, autocompletados, guardado de secuencias, etc. Añadirán un valor extra a nuestro software.

Testear la accesibilidad

Cuando hablamos de accesibilidad, no solo nos referimos a que cualquier tipo de usuario pueda acceder, sino que también haya frameworks de test automatizados que cuenten con acceso.

Además, un software cuenta con una amplía accesibilidad si todo tipo de usuarios, independientemente de que cuenten con ciertas discapacidades, pueden acceder a sus funciones de manera correcta y sencilla. Hay ciertos estándares que ya regulan y obligan a que todo software tenga un nivel medio de accesibilidad implementado.

En ciertos casos, incluso expertos en accesibilidad deben de testear este software y realizar una auditoría para comprobar que los mínimos exigidos se encuentran presentes.

3.5. Herramientas de depuración de código

Las herramientas de depuración de código pueden usarse de diferente manera, por ejemplo, existe JUnIt que se trata de un framework que da la oportunidad de que se realicen pruebas de repetición. Esto consiste generalmente en que se diseña una prueba en concreto para cierta aplicación que se ejecuta unas varias veces en busca de distintos errores. Como hemos visto antes, estas pruebas deben de ejecutarse cada vez que se haya modificado el código para ir verificando que los cambios no alteran su funcionamiento.

3.6. Planificación de pruebas

Nada más que se empieza con el diseño de software es necesario que nos planteemos que tipo de pruebas se van a realizar y en qué punto van a ser realizadas.

Cuando se trata de un proyecto de gran envergadura, seguiremos las pruebas que vamos a tratar a continuación. Si, por otro lado, el proyecto es pequeño o de bajo presupuesto, será decisión de los encargados del proyecto determinar las pruebas a seguir.

3.6.1. Pruebas unitarias

Estas pruebas normalmente se suelen dar durante las primeras fases del desarrollo del software. Es importante no retrasar las pruebas mucho porque luego deberemos de integrar todas las demás partes del proyecto, y en caso de fallos, costará más encontrarlos y repararlos.

Cuando trabajamos con programación orientada a objetos, las pruebas unitarias se realizan a nivel de objeto y después, a nivel de librerías. Realmente un paquete o librería no tendría efecto de comprobación si no se han comprobado los objetos individualmente primero.

3.6.2. Pruebas de integración

Si ya se han probado individualmente todos los componentes, es el momento de pasar a la integración de los módulos. Estas pruebas se realizan siempre en el final del diseño y en el final de la codificación del programa o aplicación.

Hay tanto pruebas de integración ascendentes como pruebas de integración descendentes. Se pueden probar los módulos de más general a más específico, o, al contrario.

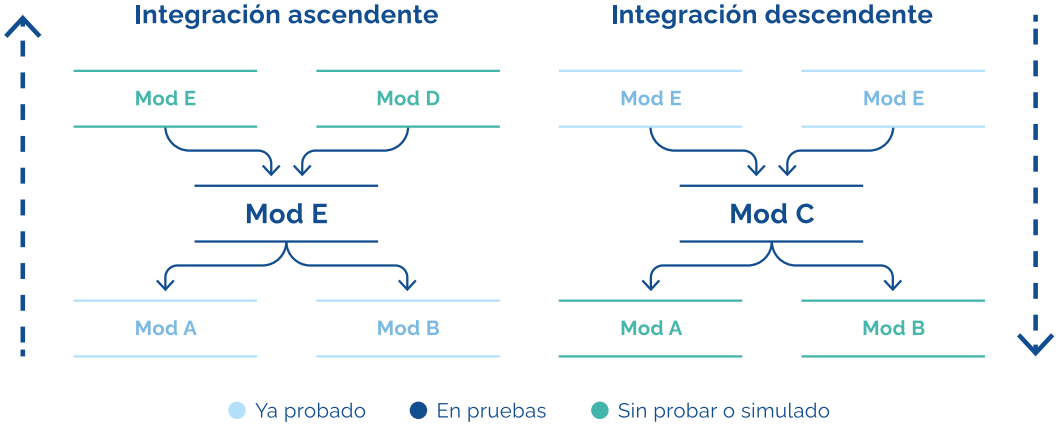

Imagen 1. Pruebas de integración ascendente y descendente.

Como podemos ver en la anterior imagen, ambas filosofías son contrarias.

No obstante, aunque se contradicen, el objetivo siempre va a ser que el sistema en conjunto funcione de manera correcta.

3.6.3. Pruebas de aceptación o validación

En las pruebas de aceptación se comprueba el programa al completo. No solo se comprobarán los requisitos del programa y su funcionamiento individualmente, también se observará si se cuenta con algún fallo técnico y que el funcionamiento del programa en general se mantiene en el tiempo.

En esta fase, se realizarán sobre el software pruebas de carga y de estrés para observar el rendimiento del sistema en casos en los que el pico de trabajo sea muy alto, esperando siempre una buena aceptación de estos casos.

Aquí tenemos dos tipos de pruebas:

- **Alfa.** Se comprueba el funcionamiento del sistema en un entorno cerrado con unas ciertas características predefinidas.
- **Beta.** Los usuarios finales testean el sistema sin el control del desarrollador.

3.6.4. Automatización de pruebas

Cuando las pruebas son de regresión, es decir, las pruebas se repiten en varias ocasiones, es recomendable que estas se automaticen para no perder mucho tiempo y asegurarnos que siempre se realiza la misma prueba.

Para poder realizar estas automatizaciones es necesario que se cuente con una documentación bien estructura que nos ayude a saber que datos se han ido usando y que tipo de pruebas deben de repetirse además de su periodo de repetición.

3.7. Calidad del software

La calidad que tenga un software es un aspecto fundamental que tratar cuando lo desarrollamos, pues dependiendo de más o menos calidad, será mejor o peor competidos nuestro software de cara al mercado.

Debido a la estructura con la que cuenta un software, es mucho más difícil determinar su calidad y su proceso de garantía es igual de complicado.

Los sistemas que se desarrollen no deben de tener fallos y además deben de cumplir con las garantías que el cliente solicita. Existen multitud de tipos de software, desde sistemas empotrados hasta software en electrodomésticos de uso diario.

Dentro de lo que, determinada la calidad de un sistema, no solo se va a tener en cuenta con su funcionamiento, si no, por ejemplo, el soporte que ofrezca una aplicación con respecto a futuros problemas también es muy importante, así como el mantenimiento que se vaya a hacer.

Para la valoración total de la calidad de un sistema se evalúa la aplicación al completo y su rendimiento.

Las mediciones de como rinde un sistema se pueden hacer desde el usuario, evaluando los tiempos de respuesta o desde el sistema en que se aloje comprobando su gasto de recursos. Varias medidas que se tienen en cuenta son, por ejemplo, la capacidad de ejecución o la carga de trabajo que conlleva en el sistema anfitrión.

Es necesario que para que la evaluación sea correcta, nos atengamos a diferentes puntos de vista para poder verlo desde varios ámbitos.

Las llamadas pruebas de carga las realizamos sobre el sistema lanzando a este una serie de simulaciones de carga reales, bajas, medias y altas, para comprobar cómo responde en tiempos de ejecución y en gasto de recursos el software.

Las otras pruebas que podemos realizar son las pruebas de estrés, que consisten en elevar la carga del equipo poco a poco para ver cómo va funcionando la aplicación conforme esta carga aumenta hasta niveles extremos.

Existen también las pruebas de estabilidad o pruebas de picos, en estas, o se mantiene una carga constante o se va cambiando drásticamente, de manera respectiva.

Estas pruebas se pueden realizar también con software de terceros.

3.7.1. Medidas o métricas de calidad del software

Los criterios de calidad con los que debe de contar un software se definen cuando se comienza el proyecto y no deben de moverse.

Todos y cada uno de los criterios que especifiquemos deben de poder medirse, y los dos criterios más comunes son:

- Número de errores por un número determinado de líneas de código.
- Número medio de revisiones realizadas a una función o módulo de programa.

Para ir evaluando estos criterios se suelen realizar revisiones técnicas formales o RTF.

Las medidas para evaluar los principales criterios de calidad son las siguientes:

- Tolerancia a errores.
- Facilidad de expansión.
- Independida de plataforma del hardware.
- Modularidad
- Estandarización de los datos.

www.universae.com