Diseño y realización de pruebas

Contenido

Filosofía de las pruebas de software

Las pruebas constituyen una de las tareas del ciclo de vida del software y sirven para detectar defectos que se han cometido durante el proceso de construcción. Aunque se trate de una pequeña aplicación, es imposible probar el software completo, es decir, encontrar todos sus errores.
La filosofía de las pruebas es encontrar errores con pruebas que no supongan un esfuerzo excesivo y elevar la posibilidad de encontrar fallos.

- Mediante las pruebas se pretende comprobar si se está creando el producto correcto **(validación)**
- Por otro lado, se mira si el producto se está desarrollando correctamente, si hace lo que tiene que hacer **(verificación)**

En relación con las pruebas, hay dos aspectos estratégicos con su puesta en práctica:

- **Aplicación de las pruebas**, qué partes del software deben someterse con lo que se desarrolla
- **Técnicas del diseño de casos de prueba**, se decide cómo se debe llevar a cabo la prueba de una parte de software

Estrategia de pruebas

Comienzan siendo pruebas pequeñas y se van incrementando a medida que incrementa el código. Se llevan 4 pruebas secuenciales:

Prueba de unidad

Se prueba primero los módulos individuales (las clases en POO). Se comprueba que funcionan correctamente cada uno de los métodos para asegurar que todo funciona como es debido.

Prueba de integración

Se comprueba si las clases de las que consta el software funcionan como deberían cuando se ejecuta el código. Hay dos tipos

- **Basada en hebra:** Integra las clases requeridas para responder a una entrada o evento
- **Basada en uso:** Se prueban las clases independientes para pasar luego a las dependientes

Prueba de sistema

Se comprueba si cumple con los requisitos especificados:

- Cumple todos los requisitos funcionales
- El funcionamiento y rendimiento de las interfaces HW, SW del usuario y operador
- Documentación de usuario
- Ejecución y rencimiento

Se distinguen varias pruebas:

- **Recuperación:** Se fuerza al software a fallar y se verifica que la recuperación funciona bien. Importante para los sistemas tolerantes a fallos
- **Seguridad:** Intentan verificar que los mecanismos de protección funcionan bien poniendo a prueba su seguridad, intentando emular ciberataques. Con esto se evalúa la seguridad de un programa
- **Esfuerzo:** Se expone al sistema a situaciones extremas para comprobar su rendimiento ante mucho estrés.
- **Rendimiento:** Se hace en sistemas que no solo necesitan requisitos funcionales, sino requisitos de rendimiento con respuestas en tiempo limitado.
- **Despliegue:** Se prueba el sistema en distintas plataformas donde debe poderse utilizar. Puede ser necesario probarlo en diferentes sistemas operativos y navegadores, probando el máximo posible de combinaciones navegador-SO

Prueba de validación

También llamada prueba de aceptación es la prueba donde el usuario determina si el software es válido y está preparado para implantarse en el entorno de quienes lo vayan a usar. Se hace por medio de unos requisitos establecidos previamente, que deben estar especificados en los requisitos de software. Participan activamente los usuarios que van a usar la aplicación, y dan su consentimiento para su uso o explotación.

Modelo en V

Define el curso de un proyecto en fases individuales detalladas. La forma en V viene de que los 4 niveles principales se asocian entre ellos (entre los niveles de desarrollo y los de prueba). Cada actividad de desarrollo debe completarse con la prueba

Estudio de requisitos

Hay dos tipos de requisitos que especifican las partes interesadas:

- Requisitos funcionales: **Qué** funciones será capaz de resaltar
- Requisitos no funcionales: **Cómo** esperan que funcione la aplicación. Especifican la forma en la que un sistema debe realizar funciones

Especificaciones

La solución que va a permitir paliar los requisitos que se piden. Se define la tecnología sobre la que se va a implantar, las funciones, los flujos de trabajo, las estructuras de datos, etc.

Diseño del sistema

Se especifica la funcionalidad general de los componentes, las interfaces y las dependencias. Permiten definir cómo van a actuar las soluciones técnicas (Aquí entra el UML)

Diseño de la unidad

Se describe cada módulo y la lógica interna que se va a implementar, una especificación de la interfaz y las tablas de bases de datos. Permite ejecutar el programa y llegar hasta el punto final (el software)

Código

La parte final es el código, el trabajo de codificación en un lenguaje de programación, siguiendo las especificaciones que se han terminado en las fases previas

Técnicas de diseño de casos de prueba

Las técnicas de diseño son las formas que se pueden generar casos de prueba para los softwares. Se pueden aplicar distintos tipos de técnica dependiendo del enfoque que se busque, pero es conveniente hacer siempre los siguientes:

Pruebas de caja blanca o estructurales (rutas)

Permiten revisar detalles internos y derivan en casos de prueba que garantizan que todas las rutas independientes se han revisado, se revisan todas las decisiones lógicas y que los bucles y las estructuras de datos son válidas.

Hay distintos casos de prueba de caja blanca:

- **Prueba de camino básico** o prueba de ruta básica: Permite generar una métrica del módulo probado, llamado también Complejidad Ciclomática. Esta complejidad indica el número de casos de prueba mínimos que hay que hacer para comprobar la funcionalidad de todo el método en cuestión.
    - **Se asgina un número a cada sentencia (stackeable)**
    - **Se unen los nodos mediante aristas según su estructura de control (if, if-else, if-else-if, for, while, do-while, try-catch)**
    - Se calcula de la siguiente manera:
        - V(G): Numero de regiones del esquema
        - V(G) = Numero de aristas – numero de nodos + 2´
        - V(G) = Numero de nodos predicados + 1
    - Cabe recalcar que **es un grafo más bien subjetivo**, no tienen que ser todos iguales para un mismo código, ya que se pueden simplificar los nodos
    - Una vez se saca la complejidad ciclomática, se tienen que calcular todos los posibles caminos y describir como comprobar cada uno.
- **Prueba de bucles:** La prueba de bucles se centra en este tipo de estructuras, y se debe generar dependiendo del tipo de bucle:
    - **Simples:** Si el máximo de iteraciones es n, se debe comprobar que se salga del bucle, que realice n iteraciones (cada caso), y cuando salga del bucle
    - **Anidados:** Se hace la prueba de bucles, pero se empieza del más interno al exterior, manteniendo los valores mínimos
    - **Bucles concatenados:** Si son independientes, se usa la estrategia de bucles simples. Si son concatenados, el segundo usa al primero, así que se usa la de anidados
    - **Bucles no deseados:** No se analizan, directamente fuera si suceden
- **Prueba de flujo de datos:** Siguen las variables a lo largo de las rutas de ejecución para garantizar que se declaran, se inician y se usan bien.

Pruebas de caja negra o funcionales (tablas y esas cosas)

Crean casos de prueba que revisan todos los requisitos funcionales. Existen distintas técnicas de diseño de pruebas

- **Particiones o clases de equivalencias (Validas/No validas):** Identifican un conjunto de tipos de valores que se deben asignar a los datos de entrada. Hay una serie de reglas para identificar las clases de equivalencia en función del tipo de condición:
    - **Valor especifico:** Una clase válida y una no válida
    - **Rango de valores:** Una clase válida y dos no válidas
    - **Una serie de valores permitidos:** Una clase válida por cada valor permitido y una no válida
    - **Si es una booleana o condición lógica:** Una válida y una no válida

  Una vez se ha creado todo correctamente, se asigna un número a cada clase (válida y no válida), y se hacen el número mínimo de pruebas para todas las clases válidas y no válidas (en las no válidas, tiene que ser una no válida y el resto de valores en válido)
- **Análisis de valores límite:** Es una extensión de las clases de equivalencia. En vez de seleccionar cualquier elemento, se selecciona aquel o aquellos que se hallan en esos límites de la clase. En rango de valores, se calcula el caso extremo y el caso justo por encima/debajo. En conjunto de valores: Se da un caso para cada extremo, un caso menos que el mínimo y uno más que el máximo
- **Conjetura de errores:** Genera una lista de errores que se suelen cometer, y luego genera casos de prueba basados en esa lista, son errores comunes.

Documentación de las pruebas

Documentación del diseño

Es conveniente documentar las pruebas para organizarlas correctamente y asegurar su reutilización. En cuanto al diseño de pruebas, es conveniente crear los siguientes documentos:

- **Plan de Pruebas:** Planificación general de las pruebas para la aplicación creada. Se enfocan de forma general las pruebas, los recursos requeridos, las actividades en las pruebas, los elementos, el personal y los riesgos. Se desglosa en varias especificaciones, cada documento con varias especificaciones de los casos de prueba y los procedimientos.
- **Especificación del diseño:** Detalla el plan de pruebas e indica cada uno de los elementos que se van a probar:
    - **Especificaciones de casos de prueba:** Se indican los datos de entrada y salida, los resultados y las dependencias
    - **Especificaciones de procedimientos de prueba:** Especifica los pasos para ejecutar los conjuntos de casos de prueba.

Documentación de la ejecución

Es importante para la eficacia en detección y corrección de defectos del programa. Por cada ejecución de pruebas, se genera un histórico con todos los hechos relevantes ocurridos durante las pruebas y un informe por cada incidente. Se recoge en un informe de resumen de pruebas.

- **Histórico de pruebas:** Registra los hechos relevantes ocurridos durante la ejecución: Entornos de pruebas, resultados, fecha y hora del informe, elementos probados…
- **Informe de incidentes:** Registra cada incidente ocurrido durante la prueba

Informe resumen de pruebas

La ejecución de las pruebas genera un informe de resumen de pruebas, que incluye el resumen de la evaluación de los elementos, las valoraciones del software y la cobertura, los resultados obtenidos y el consumo de recurso por las actividades de prueba.

Es importante validar los procesos de prueba porque puede ayudar en proyectos posteriores. Es conveniente realizar un análisis causal para proporcionar información sobre los defectos encontrados. De estos se quiere saber cuñando se cometieron, quién lo hizo, cómo se podría haber detectado y cómo se encontró el error.

Herramientas de depuración

A veces no es fácil ver el fallo del código, por lo que las herramientas de depuración pueden ayudar a localizar esos errores y corregirlos. Así mismo, cuando se corrige un defecto, puede que el resto del código se vea afectado por dicho fallo. Por ello, se deben hacer pruebas de regresión y valorar qué se debería cambiar y evitar fallos de software.

Las herramientas de depuración se encargan de ejecutar el código paso a paso, estableciendo puntos de ruptura donde interrumpir el código, y examinar los valores a lo largo de la ejecución.