Tema 3

Las pruebas se hacen en todo momento, pero para ello se seguirán unas reglas

1º Pruebas unitarias: Son las realizadas por cada desarrollador en el código implementado

2º Las de integración, donde se prestan atención al diseño y la construcción de la arquitectura

3º Las de validación, donde se comprueba que el programa cumple con lo establecido

4º Pruebas de sistema se verfica el funcionamiento global del sistema

Tipos:

Caja negra: También llamadas funcionales, se trata de probar si las salidas, a partir de unas entradas, devuelve lo esperado, sin mirar el resto

Particion equivalente: Consiste en dividir y separar los campos de entrada según el tipo de datos y las restricciones que conllevas (código postal)

Analisis de valores límite: En este tipo de prueba, se eligen valores que se encuentran en el límite delas clases de equivalencia

Pruebas aleatorias

Caja blanca: También llamadas estructurales, sirven para ver como el programa se comporta al ejecutarse, se utilizan pruebas estructurales, que se fijan anteriormente, se pretende:

Ejecutar todas las instrucciones del programa y así remover el código no usado

Ver los caminos lógicos del programa va a recorrer

Tipos:

Cobertura de decisiones: se trata de crear los suficientes casos de prueba para que cada opción resultado de una decisión 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.

Pruebas de regresión

Durante el proceso de prueba, tendremos éxito si detectamos un posible fallo o error, con lo que conlleva la modificación de componente donde se ha detectado

Esto puede desencadenar más fallos, por ello obliga a repetir las pruebas, a dichas pruebas se les denominan pruebas de regresión

Para que no aumenten mucho las pruebas:

Se hace hincapié en lo que ha sido modificado y sus componentes involucrados.

Herramientas de depuración

Cada IDE incluye herramientas de depuración como:

- Puntos de ruptura
- Ejecución paso a paso de cada instrucción
- Ejecución por procedimiento
- Inspección de variables…

Nos ayudan a encontrar errores o bugs que, aunque el programa compile bien, salten o hagan que este mismo no vaya bien

Debugger:

- Un programa debe compilarse con éxito para poder utilizarlo en el depurador.
- El depurador nos permite analizar todo el programa, mientras éste se ejecuta.
- **Permiten**:
  - Suspender la ejecución de un programa
  - Examinar y establecer los valores de las variables
  - Comprobar los valores devueltos por un determinado método
  - Comprobar el resultado de una comparación lógica o relacional

Puntos de ruptura:

Son marcadores que pueden establecerse en cualquier línea de código ejecutable, cuando se pone y se ejecuta el programa, se hara hasta donde esta el punto.

Tipos de ejecución:

Paso a paso Puede ayudarnos a verificar que el código de un método se ejecute en forma correcta

Paso a paso por procedimientos: Nos permite introducir los parámetros que queramos en un método

Hasta una instrucción Es lo que pasa al poner cursor o punto de ruptura, seguidamente, deberemos hacer o un paso a paso o por procedimientos

Hasta el final

Pruebas unitarias:

Tienen por objetivo probar el correcto funcionamiento de un módulo de código, sirve para ver que cada módulo funciona por separado. **Deben ser**

- Automática
- Completa: cubre la mayor cantidad del código
- Repetibles
- Independientes: no afecta a otras pruebas
- Profesionales

Los documentos que se generan son:

- Plan de pruebas
- Registro de pruebas
- Informe de incidente de pruebas
- Informe sumario de pruebas

Tema 4

La refactorización consiste en realizar pequeñas transformaciones en el código para mejorar la estructura sin que cambien el comportamiento ni funcionalidad del mismo

Ventajas

- Hacer que el software sea mas fácil de entender
- Hace que el mantenimiento sea mas sencillo
- Ayuda a encontrar errores
- Mejora el diseño

Hay que diferenciarlo de la optimización, esta busca que el programa se ejecute mas rápido, pero esto puede provocar que el código sea mas difícil de entender

Limitaciones de refactorizar

- Estamos modificando la estructura interna de un programa o de un método
- Si renombramos un método, hay que cambiar todas las referencias que tengamos sobre el
- No es recomendable cuando la estructura es de vital importancia en el diseño de la aplicación
- Hay veces que no hace falta, ya que puede estar tan mal que deberías de hacerlo de nuevo

Patrones

- Tabulacion
- Renombrado
- Extraer método
- Encapsular métodos (getters and setters)
- Extraer la clase
- Mover metodo
- Borrado seguro
- Cambiar los parametros de un metodo
- Extraer la interfaz

Documentación

Para explicar su funcionamiento

En caso de que se use por varias personas, detección de errores y su posterior mantenimiento

Para todo esto se usa los comentarios

Si el código es modificado, el comentario también deberías

Problemas y soluciones:

Si el código no está documentado, puede resultar bastante costoso de entender

Usar herramientas que permiten automatizar, completar y enriquecer nuestra documentación