***Cobertura de Pruebas***, 1

**Estrategia de las pruebas**, 3

**Estrategias de validación y depuración de Software**, 1

*Testing de Unidad*, 4

*Testing integración*, 4

1. **Estrategias de validación y depuración de Software**

Uno de las primeras estrategias a tomar en cuenta para que el software tenga una mínima posibilidad de fallar, es el verificar que no se presenten los siguientes errores:

* Errores en la referencia de datos
* Errores en la declaración de datos
* Errores de cómputo
* Errores en la Interfaz
* Errores de Entrada/Salida
* Errores de Portabilidad

Para evitar este tipo de errores se harán pruebas de validación mediante Testing (pruebas de software), ya que esto permitirá evitar errores antes mencionados, sin embargon Testing se utiliza para mostrar la presencia de errores, pero nunca su ausencia.

Una parte necesaria de los casos de prueba es la definición del resultado o salida esperada. Siempre compare cuidadosamente el resultado actual con el esperado para todos los casos de prueba.

Los casos de prueba deben ser escritos para condiciones de entrada inválidos e inesperados, al igual que para condiciones válidas y esperadas. Inválido se define como una condición que está fuera del conjunto de condiciones válidas, y deberá ser diagnosticado como tal por el programa.

Los casos de prueba deben ser escritos para generar condiciones de salida esperadas. Los testers con menos experiencia tienden a pensar desde la perspectiva de las entradas. Los testers con más experiencia determinan las entradas requeridas para generar un conjunto prediseñado de salidas. Asegúrese de incluir la salida “Inválida” en ese conjunto.

Con la excepción de las pruebas unitarias y de integración, un programa no deberá ser probado por la persona u organización que lo desarrolló.

El número de errores no descubiertos es directamente proporcional al número de errores descubiertos.

1. ***Cobertura de Pruebas***

* ¿Cómo medimos que tan bien probado fue un producto?
* ¿En que grado nuestros casos de prueba cubren al producto?
* ¿Cómo medimos que tan buen trabajo estamos haciendo como testers?
* Un caso de pruebas . . .
  + Cubre cierta parte de los requerimientos
  + Cubre cierta parte de la funcionalidad (Diseño Funcional)
  + Cubre cierta parte de la lógica interna del programa
* Lo que se debe hacer es generar suficientes casos de prueba para cubrir todos los puntos de cada nivel.
* Caja Negra
  + Se realizan a partir de las especificaciones del diseño funcional sin importar la estructura interna del programa.
  + En la práctica es importante probar, o al menos hacer los planes de prueba, de los requerimientos y el diseño funcional sin tener mucho conocimiento del código.
  + El conocer el código contamina la manera como se ven los requerimientos.
* Caja Blanca
  + Se realizan a partir de las especificaciones del diseño interno y del código.
  + Estas pruebas no detectan funciones faltantes u omisiones.
  + Son necesarias para probar rutas lógicas que no son discernibles en la funcionalidad externa (ejem. una función matemática que tiene dos algoritmos diferentes dependiendo de los datos).
* ¿De dónde sacamos los casos de prueba que validan al código?
  + Documento de los requerimientos
  + Documento del diseño funcional
  + Documento del diseño interno
* Pruebas basadas en requerimientos
  + Estrategia de Caja Negra
* Pruebas funcionales
  + Estrategia de Caja Negra
* Pruebas internas
  + Estrategia de Caja Blanca
* Definición de Resultados
  + **Si el resultado esperado no está especificado, es muy fácil interpretar el resultado obtendio como el correcto**
  + **“El ojo ve lo que quiere ver.” (Myers, 1979).**
* Repetición
  + **Si un error no se puede repetir, no es error.**
  + **Es muy difícil en un ambiente de multiprocesamiento (o multi-hilos) asíncrono.**
* Una *prueba*
  + (i) Una actividad en la cual el sistema o componente es ejecutado bajo condiciones específicas, los resultados son observados o grabados, y se hace una evaluación de algunos aspectos del sistema o componente.
  + Un *caso de* *prueba*
  + (i) Un conjunto de entradas de prueba, condiciones de ejecución, y resultados esperados desarrollados para un objetivo particular.
  + (ii) La entidad más pequeña que siempre es ejecutada como una unidad, de principio a fin.
  + (ii) Un conjunto de uno o más casos de pruebas.
* Un *procedimiento de* *prueba*
  + (i) Las instrucciones detalladas para la implantación, ejecución y evaluación de los resultados para un caso de pruebas dado.
  + (ii) Un caso de pruebas puede ser usado en uno o más procedimientos de prueba.

1. **Estrategia de las pruebas:**

Es imposible e inviable encontrar todos los errores de un programa, y este no es la excepción, es decir, se pueden diseñar y realizar muy buenos test pero aun así no aseguran la posterior inexistencia de errores. Por esto es importante establecer una estrategia que nos permita decidir si un test fue exitoso o no.:

Se realizarán dos tipos de testing estos son de integración y unidad.

* *Testing integración*

En el caso de testing de integración se realizaran pruebas por funcionalidad que a su vez contiene todas las operaciones necesarias, estas serán creadas a partir del modelo **PHASE MODELS,** cada prueba tendrá un nivel de tolerancia.

Las pruebas se realizaran conectando las tres capas de la aplicacion

¿Quién diseña las pruebas para cada funcionalidad para el testing de integración?

El encargado de diseñar las pruebas para cada acción es el mismo encargado de realizar el test, por lo cual se reuquiere un alto nivel de entendimiento sobre entradas y salidas esperadas para la funcionalidad, pero debe saber bien como es la interacción de esta con otras funcionalidades.

¿Cuál será la tolerancia a errores en el testing de integración?

Se considera los errores no pueden ser erradicados totalmente pero en este caso realizaremos caja negra conectando las tres capas, se considera que un 0% de aprobación en los test de las operaciones de cada funcionalidad es una funcionalidad exitosa.

¿Quién es el encargado de realizar cada prueba?

Los encargados se encuentran ya definidos en una planificación, estos para garantizar la correctitud del las pruebas no tienen relación cercana con la acción testeada, esto inhabilita solamente al encargado de la acción.

* *Testing de Unidad*

En el testing de unidad se ocupara una sistema automatizado, es decir, una herramienta que hace el test, esta es JUnit

¿Quién es el encargado de crear las clases para JUnit?

El encargado de crear las clases necesarias para que JUnit trabaje es el mismo que realiza la funcionalidad

¿Quiénes son los encargados de hacer el test de cada funcionalidad (Trabajar con Junit)?

Los encargados son personas que no tienen relación directa con la funcionalidad

¿Cuál será la tolerancia de errores en el test de unidad?

Esta debe tener 0% de error en JUnit