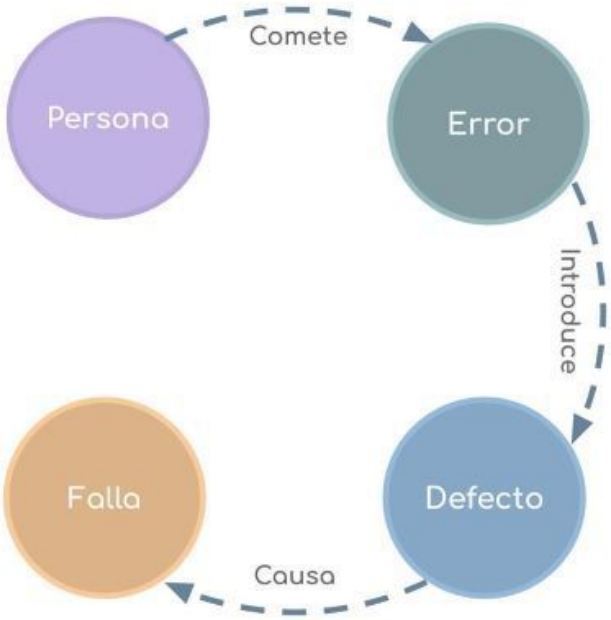
INFORMATORIO

Taller de Programación Web

Testing: Conceptos iniciales



Error *o bug*

es provocado por una acción humana.
Por ejemplo: Quien programa provoca un error el cual producirá un resultado no esperado.

Defecto

es provocado por un error de implementación.
Por ejemplo: el defecto será por haber utilizado el operador "x+y<z" en lugar de "x+y =<z".

Fallo

se produce por ejecutar un programa con defectos.
Por ejemplo, retomando el ejemplo anterior al hacer las suma de ambos componentes no obtendríamos los mismos resultados y esto a nivel de sistema muy complejo puede llegar a producir efectos catastróficos.

Practiquemos entonces

Un módulo de registro de usuarios tiene mala configuración en la función de conexión a base de datos

ERROR ____ DEFECTO ____ FALLO ____

Visualización de un mensaje de alerta que no fue definido previamente por el desarrollador.

ERROR ____ DEFECTO ____ FALLO ____

Testing: Validación y Verificación

El **testing de software** pertenece a una actividad del proceso de producción de software denominada Verificación y Validación –usualmente abreviada como V&V.

- **Verificación:**
  "¿Estamos construyendo el producto corréctamente?".
- El software debería ajustarse a su especificación
- **Validación:**
  "¿estamos construyendo el producto correcto?".
- El software debería hacer lo que el cliente realmente reclama.

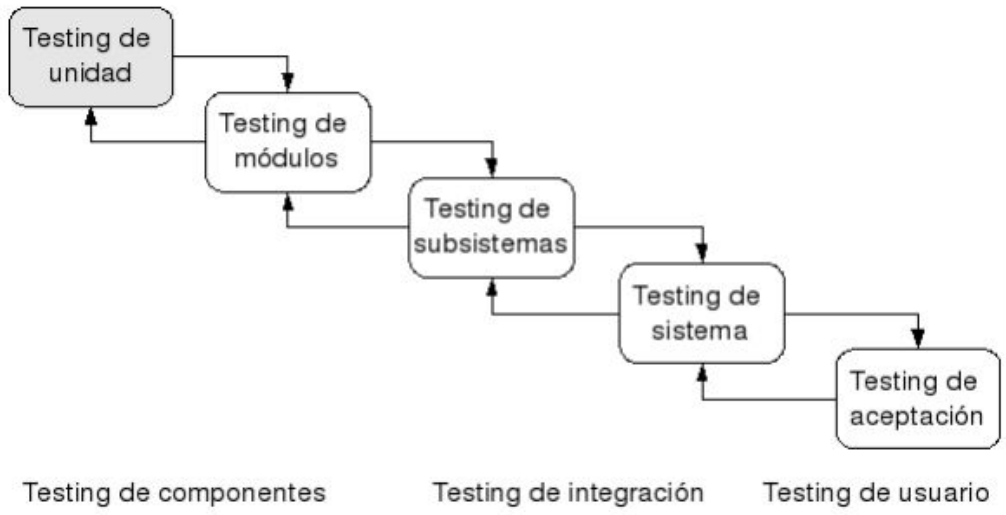
V&V es el nombre genérico dado a las actividades de comprobación que aseguran que el software respeta su especificación y satisface las necesidades de sus usuarios. El sistema debe ser verificado y validado en cada etapa del proceso de desarrollo.

Aunque el testing se puede aplicar tanto en verificación como en validación, en este curso nos concentraremos casi exclusivamente en el testing de programas como parte de la verificación.

Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.
E. W. Dijkstra

El proceso de testing

Es casi imposible, excepto para los programas mas pequeños, testear un software como si fuera una única entidad monolítica.



Cuál es el objetivo del testing: ***aportar calidad al software que se está verificando***. ¿Cómo? Detectando posibles incidencias de diversa índole que pueda tener cuando esté en uso.

Y la forma en la que estaremos aportando calidad es principalmente buscando fallos. Entonces, si no encontramos fallos todo el costo del testing se puede haber ahorrado. ¿No? ¡Nooo! Porque si no encontramos fallos entonces tenemos más confianza en que los usuarios encontrarán menos problemas.

¡Ojo!, ***el objetivo no tiene por qué ser el mismo a lo largo del tiempo***, pero sí es importante que se tenga claro en cada momento cuál es. Puede ser válido en cierto momento tener como objetivo del testing encontrar la mayor cantidad de errores posibles, entonces seguramente el testing se enfoque en las áreas más complejas del software o menos logradas. Si el objetivo es dar seguridad a los usuarios, entonces seguramente se enfoque el testing en los escenarios más usados por clientes.

CASO DE PRUEBA

¿De qué hablamos cuando nos referimos a un Caso de Prueba? Veamos algunas definiciones:

De acuerdo al Estándar IEEE 610 (1990) un caso de prueba se define como:

> *"Un conjunto de entradas de prueba, condiciones de ejecución, y resultados esperados desarrollados con un objetivo particular, tal como el de ejercitar un camino en particular de un programa o el verificar que cumple con un requerimiento específico."*

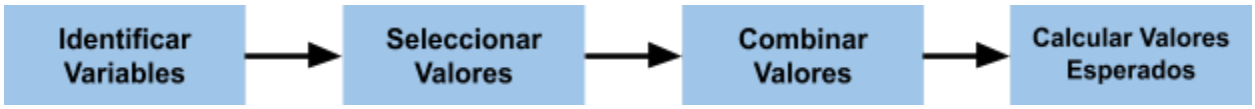
Brian Marick utiliza un término relacionado para describir un caso de prueba que está ligeramente documentado, referido como "La Idea de Prueba":

> *"Una idea de prueba es una breve declaración de algo que debería ser probado. Por ejemplo, si se está probando la función de la raíz cuadrada, una idea de prueba sería el 'probar un número que sea menor que cero'. La idea es chequear si el código maneja un caso de error."*

Por último, una definición provista por Cem Kaner:

> *"Un caso de prueba es una pregunta que se le hace al programa. La intención de ejecutar un caso de prueba es la de obtener información, por ejemplo sea que el programa va a pasar o fallar la prueba. Puede o no contar con gran detalle en cuanto a procedimiento, en tanto que sea clara cuál es la idea de la prueba y cómo se debe aplicar esa idea a algunos aspectos específicos (característica, por ejemplo) del producto."*

¿Cómo diseñar un buen caso de prueba?



a) Identificar Variables

El comportamiento esperado de cada funcionalidad a probar es afectado por una serie de variables que deben ser identificadas para el diseño de pruebas. Tal como indica el nombre, estamos hablando de variables, de cualquier entrada que al cambiar su valor hace que cambie el comportamiento o resultado del sistema bajo pruebas.
Qué puede conformar las variables de una funcionalidad:

- Entradas del usuario por pantalla.
- Parámetros del sistema (en archivos de configuración, parámetros de invocación del programa, tablas de parámetros, etc.).
- Estado de la base de datos (existencia –o no– de registros con determinados valores o propiedades).

Se debe identificar qué variables son las que nos interesan, para focalizar el diseño de las pruebas y de los datos de prueba considerando estas variables, y no focalizar en otras pues esto nos hará diseñar y ejecutar más pruebas que quizá no aporten valor al conjunto de pruebas.

b) Seleccionar Valores

Una vez que se identifican las variables en juego en la funcionalidad bajo pruebas, para cada una hay que seleccionar los valores que se le asignarán en las pruebas.

Para cada variable se decidirá utilizar distintos datos, los cuales sean interesantes desde el punto de vista del testing.

Por ejemplo, si tenemos una variable para ingresar un identificador que acepta cadenas de entre 5 y 10 caracteres podremos pensar en “Fede”, que es muy corto por lo que debe dar un error, “Federico” que es una entrada válida, “Federico_Toledo” que es muy largo, por lo que debe dar error. En cambio, “Federico” y “Andrés” son ambas válidas, no son “significativamente diferentes”, ambas deberían generar el mismo comportamiento en el sistema.

c) Combinar Valores

Cuando diseñamos casos de prueba se utilizan datos para sus entradas y salidas esperadas, y por lo tanto también diseñamos datos de prueba para cada variable que está en juego. Cada caso de prueba se ejecutará con distintos juegos de datos, pero ¿es necesario utilizar todas las combinaciones posibles? Con casos muy simples, que manejan pocas variables y pocos valores, ya podemos observar que la cantidad de ejecuciones que necesitamos serían muchas, por lo que tenemos que intentar encontrar las combinaciones de datos que tienen más valor para el testing, o sea, las que tienen más probabilidad de encontrar errores.

d) Calcular Valores Esperados

Para cada prueba diseñada, se calculan los valores esperados de acuerdo a las especificaciones del sistema, o a nuestro conocimiento del negocio. Esta es la única forma de determinar si la aplicación funciona como es esperado o no. Es importante para que al momento de ejecutar las pruebas no se tengan que analizar los resultados, sino que sea posible comparar directamente con lo que ya estaba previsto.

Con valores esperados estamos incluyendo a lo que el sistema muestra en pantalla, archivos que se tengan que modificar, estados de la base de datos, o cualquier otro tipo de acción que deba realizar el software que estamos probando. Así mismo, una buena práctica es incluir pruebas que deban registrar un fallo, que su resultado esperado sea el manejo de una excepción y su correcto procesamiento. Esto también es conocido como ***Testing Negativo***, que consiste en incluir escenarios con aquellas cosas que el sistema debe estar preparado para no hacer, o no fallar por tratarse de una situación incorrecta.