**Capitulo 13 - Estrategias de prueba**

Definición: Integración de métodos de diseño de casos de prueba del software en una serie bien planeada de pasos que desembocará en la eficaz construcción del software. Mapa descriptivo de esos pasos.

Incluye: planeación de pruebas, diseño de casos de prueba, ejecución de pruebas, recolección y evaluación de los datos resultantes.

**Un enfoque estratégico para la prueba del software**

Características Genéricas

* Para realizar pruebas efectivas un equipo software debe efectuar revisiones técnicas formales y efectivas. Esto elimina muchos errores antes de que empiece la prueba.
* La prueba comienza al nivel de componentes y trabaja hacia la integración de todo el sistema de cómputos.
* Diferentes técnicas de prueba son apropiadas en diferentes momentos.
* La prueba la dirige el desarrollador del software y un GIP en caso de proyectos grandes.
* La prueba y depuración son actividades diferentes, pero la segunda debe incluirse en cualquier estrategia de prueba.

Prueba: conjunto de actividades que se planean con anticipación y se realizan de manera sistemática.

Estrategia debe incluir:

* Pruebas de Bajo nivel.
* Pruebas de Alto nivel.

**Verificación y Validación**

Verificación: Es el conjunto de actividades que aseguran que el software implemente correctamente una función específica.

Validación: Es un conjunto diferente de actividades que aseguran que el software construido corresponde con los requisitos del cliente.

Ambas abarcan una amplia lista de actividades de aseguramiento de calidad.

No considerar las pruebas como una red de seguridad.

**Organización para las pruebas del Software**

* Conflicto de intereses.
* Cuando se comienzan las pruebas hay un intento de destruir lo creado.
* Es el cliente quien encuentra los errores, no el ingeniero de software.
* Malas interpretaciones.
* Mayoría de las veces: desarrollador, luego GIP.
* Si no hay GIP, aplicar su punto de vista.

Grupo independiente de prueba (GIP): Consiste en eliminar los problemas propios de dejar que el constructor pruebe lo que el mismo ha construido. La prueba independiente elimina el conflicto de interés que, de otra manera, estaría presente. GIP y desarrollador trabajan unidos, durante el trascurso del proyecto. Por último, el GIP no tiene conflicto de intereses

**Estrategia de prueba para arquitecturas convencionales de software**

**Estrategia de prueba de software para arquitecturas OO**

* Filosofía idéntica (arquitectura convencional.), pero con diferente enfoque. Se va : Pequeño 🡪 Grande
* Unidad 🡪 Clase = atributos, operaciones. Además requiere comunicación y colaboración.
* A medida que se Integran las clases se realizan:
  + Pruebas de regresión.
  + Prueba de sistema.

**Criterios para completar la prueba**

Respuestas pragmáticas, nada empírico:

1. Nunca. Las continúa el cliente.
2. Cuando se acaba el tiempo o dinero.
3. Se afirma que concluyen las pruebas basándose en estadísticas y teoría de confiabilidad del software.

**Aspectos Estratégicos**

* Especificar los requisitos del producto de manera cuantificable mucho antes de que empiecen las pruebas.
* Establecer explícitamente los objetivos de la prueba.
* Comprender cuales son los usuarios del software y desarrollar un perfil para cada categoría de usuario.
* Desarrollar un plan de prueba que destaque la “prueba de ciclo rápido”.
* Construir un software “robusto” diseñado para probarse a si mismo.
* Usar revisiones técnicas formales y efectivas como filtro previo a la prueba.
* Realizar revisiones técnicas formales para evaluar la estrategia de prueba y los propios casos de prueba.
* Desarrollar un enfoque de mejora continua para el proceso de prueba.

**Estrategias de prueba para el software convencional**

Existen dos enfoques:

* El equipo de software espera a que el sistema este totalmente construido para aplicar las pruebas.
* El equipo de software aplica pruebas al sistema diariamente.

Enfoque incremental de pruebas, se ubica dentro de los dos enfoques anteriores:

1. Pruebas de unidades individuales del programa.
2. Pruebas que faciliten la integración de las unidades.
3. Culminar con pruebas sobre el sistema construido.

**Prueba de Unidad**

Se ubican en la verificación de la unidad más pequeña del diseño del software: componente o módulo. Se concentra en la lógica de procesamiento interno y estructuras de datos dentro de los límites del componente. Se puede aplicar en paralelo a varios componentes.

Tarea esencial: prueba selectiva de las rutas de ejecución. Se deben diseñar casos de prueba para descubrir errores debido a cálculos incorrectos, comparaciones erróneas o flujos de control inapropiados.

Prueba de límites: Con frecuencia el software falla en sus límites.

Enfoque antierror: Se proveen condiciones de error y se configuran rutas de manejo de errores para modificar la ruta o terminar limpiamente el procesamiento cuando ocurra un error.

Procedimientos de prueba de unidad: diseño de pruebas de unidad, se realizan antes (enfoque más ágil) o al finalizar la codificación.

Módulo = no es un programa independiente. Para cada prueba se debe desarrollar software:

Controlador = “programa principal” que acepta los datos del caso de prueba, los pasa al componente e imprime resultados.

Resguardo = “subprograma simulado” utiliza interfaz de módulo subordinado; realiza mínima manipulación de datos, proporciona verificación de entrada y devuelve el control al módulo de prueba.

* Ambos = sobrecarga de trabajo. No se entregan con producto final.
* Prueba de unidad se simplifica cuando se diseñan componentes con cohesión elevada.

**Prueba de Integración**

Técnica sistemática para construir la arquitectura del software mientras, al mismo tiempo, se aplican las pruebas para descubrir errores asociados a la interfaz.

Enfoques:

1. BIG BANG: Se combinan todos los componentes por anticipado y se prueba el programa como un todo. Se encuentra una gran cantidad de errores. ¡Caos!
2. Incremental: El programa se construye y prueba en pequeños incrementos, lo cual resulta más fácil para aislar y corregir errores.

**Prueba de Integración Descendente:** Los módulos se integran al descender por la jerarquía de control, empezando con el modulo de control principal. Los módulos subordinados al modulo de control principal se incorporan de dos maneras: primero-en-profundidad o primero-en-anchura.

**Prueba de Integración Ascendente**: Empieza la construcción y prueba con módulos atómicos. Debido a que se integran de abajo hacia arriba, siempre está disponible el procesamiento requerido para los componentes subordinados a un determinado nivel y se elimina la necesidad de resguardos.

**Prueba de Regresión:** Consiste en ejecutar el mismo subconjunto de pruebas que ya se han aplicado para asegurarse de que los cambios no han propagado efectos colaterales indeseables.

El conjunto de pruebas de regresión contiene 3 clases diferentes de casos de prueba:

* Muestra representativa de pruebas que ejercerán las funciones del software.
* Pruebas adicionales que se concentran en las funciones que el cambio afectaría
* Pruebas que se concentran en los componentes que han cambiado.

**Prueba de Humo:** Esta diseñado como mecanismo para marcar el ritmo en proyectos en los cuales el tiempo es crítico, permitiendo al equipo evaluar su proyecto con frecuencia. Es una estrategia de integración continua mientras se desarrolla el producto de software.

Abarca:

* Los componentes de software traducidos a código se integran en una construcción.
* Se diseña una serie de pruebas para exponer errores que impedirán que la construcción realice apropiadamente su función.
* La construcción se integra con otras construcciones y diariamente se aplica una prueba de humo a todo el producto. EL enfoque de la integración puede ser ascendente o descendente.

Beneficios:

* Se minimiza el riesgo en la integración
* Se mejora la calidad del producto final
* Se simplifican el diagnostico y la corrección de errores
* El progreso es más fácil de evaluar

**Opciones estratégicas**

En general, las Desventajas de un enfoque son ventajas del otro y viceversa. La Mejor elección es un enfoque combinado denominado Prueba sándwich, el cual utiliza Pruebas descendentes para los niveles superiores y Pruebas ascendentes para los niveles subordinados.

Se deben identificar módulos críticos, estos tienen las siguientes características:

* Atiende varios requisitos de software
* Tiene un alto grado de control
* Es complejo o propenso a errores
* Tiene requisitos de desempeño bien definidos

Las pruebas de regresión se deben concentrar en el funcionamiento de los módulos críticos.

**Documentación de la prueba de integración**

Un plan general para la integración del software y una descripción de pruebas específicas se documentan en la especificación de la prueba. Este documento que contiene un plan de prueba, un procedimiento de prueba, es un producto de trabajo del proceso de software y se vuelve parte de la configuración del software.

**Prueba de integración en el contexto OO**

Prueba basada en subprocesos: integra el conjunto de clases requerido para responder a una entrada o un evento del sistema. Cada subproceso se integra y prueba individualmente.

Prueba basada en el uso: empieza la construcción del sistema con la prueba de las clases independientes que usan muy pocas clases de servidor. Después se prueba la capa de clases dependientes, que usan las clases independientes.

**Pruebas de Validación**

Las pruebas de validación empiezan tras la culminación de la prueba de integración. En el nivel de validación o sistema desaparece la distinción entre software convencional y OO. La prueba se concentra en las acciones visibles para el usuario y en la salida del sistema que este puede reconocer.

Se alcanza la validación cuando el SW funciona de tal manera que satisface las expectativas razonables del cliente 🡪 definidas en la Especificación de requisitos del SW (criterios de validación).

**Revisión de la configuración**

Es un elemento importante en el proceso de validación. Tiene como objetivo asegurar que todos los elementos de la configuración del software se hayan desarrollado apropiadamente.

**Pruebas Alfa y Beta**

Los usuarios finales son quienes aplican la prueba alfa en el lugar de trabajo del desarrollador. El software se utiliza en un entorno natural mientras el desarrollador supervisa y registra los errores y problemas de uso. Las pruebas alfa se realizan en un entorno controlado.

Las pruebas beta se aplican en el lugar de trabajo de los usuarios finales, por lo general el desarrollador no está y se dice que es una aplicación “en vivo” del software en su entorno que no controla el desarrollador.

**Prueba del Sistema**

Abarca una serie de pruebas diferentes, cuyo propósito es:

* Ejercitar profundamente el sistema de cómputo.
* Verificar que se hayan integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

Tipos de prueba de sistema:

* *Prueba de recuperación:* Obliga al software a fallar de varias maneras y a verificar que la recuperación se realice apropiadamente. Existen dos formas de recuperación:
  + Recuperación Automática (la realiza el propio sistema)
  + Recuperación que requiere intervención humana
* *Prueba de seguridad:* Comprueba que los mecanismos de protección integrados en el sistema realmente protejan de irrupciones inapropiadas.
* *Prueba de resistencia:* Ejecuta un sistema de tal manera que requiera una cantidad, una frecuencia o un volumen anormal de recursos. El fin es sobrecargar al programa para ver cómo se comporta.
  + *Prueba de sensibilidad:* tratan de descubrir combinaciones de datos dentro de las clases de entrada validas que causen inestabilidad o procesamiento inapropiado.
* *Prueba de desempeño:* Prueba el desempeño del software en tiempo de ejecución dentro del contexto de un sistema integrado.

**El arte de la depuración**

No es una prueba, pero ocurre como consecuencia de una. Una depuración realizada con éxito conlleva a un Caso de prueba que descubre un error y la depuración elimina ese error.

La depuración relaciona síntoma con causa, lo que conduce a la corrección del error.

Dos posibles resultados:

* Se encuentra y corrige la causa.
* No se localiza la causa.

¿Depuración difícil? -> Psicología humana (programadores con habilidad y otros sin habilidad).Algunas características adicionales:

* El síntoma y la causa pueden estar separados geográficamente.
* Es posible que el síntoma desaparezca al corregir otro error.
* Es probable que el síntoma no lo cause algún error.
* El síntoma podría deberse a un error humano difícil de localizar.
* El síntoma podría deberse a problemas de tiempo y no de procesamiento.
* Dificultad de reproducir con exactitud las condiciones de entrada.
* El síntoma podría presentarse intermitentemente.
* El síntoma se debe a causas distribuidas entre varias tareas que se ejecutan en diferentes procesadores.

Durante la depuración se encuentran errores son múltiples, y van de:

* Medianamente molestos (ej. formatos de salida incorrectos.)
* Catastróficos (ej. falla total del sistema).

**Estrategias de depuración**

*Fuerza bruta:* Es el método más común y menos eficiente para aislar la causa de un error del software. Comúnmente se aplica cuando todo lo demás falla. Consume tiempo y esfuerzo.

*Rastreo hacia atrás:* Es un método muy común que se utiliza con éxito en programas pequeños. Empezando donde se encuentra un síntoma, se recorre hacia atrás el código fuente hasta hallar el sitio de la causa.

+ código + caminos hacia atrás= cantidad de información inmanejable.

*Eliminación de la causa:* Los datos relacionados con el error se organizan para aislar causas posibles. Hipótesis de la causa: se aprueba o desecha.

Depuración Automatizada: herramientas de SW incluidos en el entorno de desarrollo.

**Corrección del error**

Antes de llevar a cabo la corrección del error se deben hacer las siguientes preguntas:

* ¿La causa del error se repite en otra parte del programa?
* ¿Cuál es el “siguiente error” qué podría introducirse con la corrección que está a punto de realizarse?
* ¿Qué debió hacerse para evitar este error desde el principio?