UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

METODOLOGIA DE PRUEBAS DE SISTEMAS

T E S I S

QUE PARA OBTENER EL TITULO DE

A C T U A R I O

P R E S E N T A

MA. DEL CARMEN TERESA CADENA ARIAS

MEXICO, D. F. NOVIEMBRE DE 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

'As long as there were no machines, programming was no problem at all, when we had a few weak computers, a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem'

E. W. Dijstra (1972)

Prólogo

A medida que el hardware es más flexible surgen nuevos problemas de programación; las restricciones físicas del hardware se ven reemplazadas por las limitaciones de la capacidad de la mente humana. El rechazo psicológico a éstas, unido a un optimismo desquiciado, han conducido al diseño de sistemas de software enormes.

Existe una buena razón para justificar esta idea. Se ha demostrado que la complejidad de los sistemas de software rebasa la destreza del intelecto humano. En efecto, la gigantesca cantidad de errores que albergan los sistemas grandes, nos recuerdan lo incompleto e inconsistente que es el intelecto del género humano.

Cada uno de estos errores puede considerarse como un simple flujo lógico sin relación con el todo; i.e., lo particular nos conduce al olvido de la consideración del todo. Sin embargo la experiencia muestra que estos errores, vistos como un todo, describen un fenómeno general comunmente denominado:

'Crisis de software'

La complejidad de muchos sistemas de software ha llegado a ser inmanejable y las consecuencias naturales son retrasos e inconfiabilidad. Existen varios métodos para producir sistemas de software más confiables:

1 Seguir una metodología de desarrollo que conduzca a productos altamente confiables.

2 Probar y depurar, considerando que sólo puede ejecutarse una fracción de todos los casos de de entrada posibles.

3 Incluir redundancia a fin de detectar y corregir los errores que se muestren durante el uso de un sistema de software.

La combinación de estos tres métodos lleva al desarrollo de software más confiable.

A finales de 1983 IEEE (Institute of Electrical and Electronics Engineers) publicó las bases sobre las cuales autorizó el desarrollo de un estándar para probar software, denominado:

'Standard for Software Unit Testing'

Estas fueron:

1 Identificar las características a probar.

2 Planear el enfoque, recursos y calendario.

3 Diseñar los casos de prueba y los procedimientos.

4 Realizar el diseño.

5 Ejecutar los procedimientos de prueba.

6 Examinar los resultados.

7 Reportar la evaluación.

Hasta el momento no se ha publicado el resultado de la convocatoria, sin embargo se han hecho algunos trabajos como el publicado por la editorial Auerbach titulado:

'Standard for Software Aplication Testing'

que cubre prácticamente las bases antes mencionadas, con la limitante de enfocarse a la realización de las pruebas cuando el software ya está programado.

Este pequeño análisis motiva la elaboración de este trabajo. La autora, en su experiencia profesional, se ha enfrentado a un sinúmero de estos problemas.

'And simplicity is the inavoidable price we must pay for reliability!'

C. A. Hoare (1975)

Introducción

Las pruebas de software son un componente de la certificación de calidad de sistemas, su objetivo es detectar errores.

Prevenir un error es más deseable que detectar y corregir uno cuando el sistema ya esta programado, ya que de esta forma no hay código que corregir, no hay que volver a probar para confirmar que la corrección es buena, nadie se siente ofendido, no se consumen recursos y el calendario establecido no se ve afectado de manera adversa.

Más que la ejecución de las pruebas ,el diseño de las mismas; constituye la prevención más efectiva de los errores. El razonamiento y el análisis que deben hacerse para crear una prueba útil pueden descubrir y eliminar muchos de ellos, antes de que sean codificados.

La actividad ideal sería que no fuese necesario correr las pruebas, porque todos los errores serían encontrados y eliminados durante el diseño de las mismas. Sin embargo esto no es posible, los errores no siempre son obvios y gran parte de los sistemas se producen gracias a varios años de esfuerzo de muchos analistas, diseñadores y programadores.

Generalmente un sistema no puede ser comprendido en su totalidad por alguien, en consecuencia los estándares de calidad sólo pueden lograrse empleando métodos de control. Sin embargo no importa cuán elegantes sean los métodos que se utilicen para probar un sistema, que tan completa esté la documentación, lo estructurada que sea la arquitectura, lo detallado de los planes de desarrollo, el número de revisiones que se hagan del proyecto, lo poderoso que sea el manejador de la base de datos que se emplee, lo cuidadoso que sea el control de la configuración, lo avanzado de las técnicas, etc.; todo será inútil y el proyecto fallará si el nivel unitario del software, i.e., si las rutinas individuales no han sido probadas.

Considerando que el ciclo de desarrollo de un sistema se divide en tres fases:

(A): Fase de análisis

(D): Fase de diseño

(P): Fase de programación,

que existe una metodología de desarrollo; i.e., un enfoque formal de desarrollo que define los objetivos precisos para cada una de las fases, así como los resultados que se requieren de una fase para que la siguiente pueda comenzar, proporcionando formas especiales para la documentación a lo largo de cada fase, y que la inclusión de redundancia se llevará a cabo en el mantenimiento. El presente trabajo presenta una metodología para probar los sistemas de software.

La metodología que se presenta está escrita con la intención de cerrar, en la medida de lo posible, el abismo que existe entre teoría y práctica; la experiencia obtenida trabajando en el área de sistemas, particularmente la que se logró al haber colaborado en el Departamento de Certificación de Calidad de Sistemas en Banco Nacional de México, y el estudio en el campo de la ingeniería de software son los elementos empleados. Se visualiza a sí misma totalmente integrada al ciclo de desarrollo de sistemas, ya que probar cuando un sistema está terminado empobrece mucho su alcance.

La tesis está dividida en cinco capítulos.El primer capítulo introduce las nociones fundamentales de la metodología.El segundo muestra los métodos que se utilizan para probar un sistema de software. El tercero ejemplica la manera de realizar las pruebas de software.El penúltimo capítulo da una clasificación de los errores conocidos por la autora.El último intenta dar un marco de referencia formal para la aplicación de la metodología.

Cuando se consideró que algún enunciado no era suficientemente claro, se ejemplificó. El texto enmarcado en un rectángulo tiene esta función. El trabajo finaliza con unas conclusiones y la bibliografía consultada por la autora.

Objetivo

El objetivo de la Metodología de pruebas de sistemas es:

Prevenir errores

CAP 1. Conceptos básicos

Probar de manera efectiva es una actividad crítica para el desarrollo de productos de buena calidad.

El ambiente de pruebas y los recursos requeridos para su realización son dos aspectos clave.

El examen del código fuente y de la documentación deben realizarse antes de probar la ejecución.

En algunos casos, una prueba puede validar varias funciones y en otros varias pruebas son necesarias para validar una función.

Las pruebas de condiciones de error, condiciones frontera y acoplamientos requieren atención especial.

Las pruebas de funciones en tiempo real son más complejas que las pruebas de funciones en lote (batch).

Estos conceptos se desarrollan en los capítulos siguientes.

'Nada es absoluto,
he aquí el único
principio absoluto'

A. Comte

1.1 Axiomas

NO ES POSIBLE PROBAR TOTALMENTE UN SISTEMA

NO SE PUEDE ASEGURAR LA AUSENCIA DE ERRORES

ES POSIBLE MINIMIZAR LA CANTIDAD DE ERRORES

Utilice un proceso finito para probar

1.2 Definiciones

Probar es el proceso de revisar un sistema con la intención de detectar errores.

Un criterio es consistente cuando los conjuntos de prueba cp1 y cp2 elegidos mediante éste, son tales que todas las instancias de prueba con cp1 son exitosas exactamente cuando lo son con cp2.

Un criterio es completo cuando produce conjuntos de datos de prueba que descubren todos los errores.

1.3 Proposiciones

Trabajar especialmente con elementos representativos del dominio funcional.

Generar conjuntos de datos de prueba adecuados, i.e., aquellos que cubren el dominio funcional siendo suficientemente pequeños para llevar a cabo el proceso de prueba con cada elemento del conjunto.

Seleccionar tales conjuntos mediante un criterio consistente y completo.

1.4 Teorema fundamental de pruebas

SI EXISTE UN CRITERIO COMPLETO Y CONSISTENTE PARA SELECCIONAR UN CONJUNTO DE DATOS DE PRUEBA PARA UN SISTEMA S, Y SI EXISTE UN CONJUNTO DE DATOS DE PRUEBA QUE SATISFACE EL CRITERIO DE MODO QUE TODAS LAS INSTANCIAS DE PRUEBA CON ESTE CONJUNTO SON EXITOSAS, ENTONCES S ES CORRECTO.

Para lograr una buena aproximación a sistemas correctos es indispensable dar atención prioritaria a la generación de los datos de prueba mediante este criterio. Actualmente existen algunos generadores en el mercado y su uso agiliza esta labor.

Data Test Generator (DTG) de UNISYS Corporation.

También da buenos resultados hacer un generador de datos de prueba de acuerdo a los siguientes criterios:

- LOS ELEMENTOS DEL CONJUNTO DEBEN REFLEJAR PROPIEDADES ESPECIALES DEL DOMINIO.

  elementos frontera

- LOS ELEMENTOS DEL CONJUNTO DEBEN EJERCITAR LA ESTRUCTURA DEL SISTEMA.

  ejecución de las ramificaciones

- LOS ELEMENTOS DEL CONJUNTO DEBEN REFLEJAR PROPIEDADES ESPECIALES DE LA DESCRIPCION FUNCIONAL.

  valores del dominio que lleven a valores frontera

CAP. 2 Verificación y validación

Todo sistema se desarrolla con la intención de realizar una función particular (aplicación). Dictaminar que un sistema recientemente desarrollado la ejecuta de forma adecuada se considera la última responsabilidad de las pruebas. Existen dos procedimientos para probar:

Verificación : Procedimiento sistemático de revisión que debe emplearse durante el ciclo de desarrollo de un sistema, iniciando en la fase de análisis ((A)) y concluyendo en el momento de la implantación.

Validación : Procedimiento que asegura que el sistema funciona como fué planteado. Su duración garantiza que el sistema cubrió las espectativas.

La verificación asegurará la calidad del software en producción y mantenimiento, ya que examina que el producto del diseño sea exacto y completo. La validación examina los resultados obtenidos del mismo.

Un sistema se considera verificado cuando se ha probado que conoce la especificación de sus requerimientos, y validado cuando las pruebas muestran conformar estas especificaciones.

El término certificación se emplea para referirse al proceso de crear programas correctos mediante verificación y validación.

2.1 Enfoques

Existen dos enfoques diferentes para demostrar que un sistema es correcto:

Estructural: se basa solamente en la estructura del sistema, se trata de ejercitar todos los caminos de control (trayectorias) del sistema.

Funcional : se basa solamente en la función del sistema, se trata de validar la ejecución correcta de la función del sistema, ejecutando éste con entradas controladas.

2.2 Pruebas estructurales y funcionales

Pruebas estructurales

Su diseño se realiza en base a la estructura del sistema (D), por tanto el conjunto de casos de prueba debe escogerse basado en la estructura. La selección de los casos de prueba debe hacerse de acuerdo a los siguientes criterios:

- toda instrucción del sistema debe ejecutarse al menos una vez.
- toda ramificación debe probarse en cada dirección al menos una vez.
- toda trayectoria debe ejercitarse al menos una vez.

Para ello es conveniente contar con alguna herramienta automatizada.

un sistema pequeño puede tener billones de trayectorias

Pruebas funcionales

Su diseño se realiza en base a las especificaciones del sistema. La especificación de entradas es la extensión de acciones requeridas por el usuario (A).

Cada sistema opera sobre un número finito de entradas que pueden ser interpretadas como cuerdas de bits. Una prueba funcional consiste en someter el sistema a todas las cuerdas de entrada posibles, para ello es necesario contar con un generador de entradas y un comparador de salidas.

> una cuerda de diez caracteres tiene 2 cuerdas de entrada posibles con sus respectivas salidas, si se ejecutara una prueba cadà microsegundo, realizarlas todas tomaría en tiempo, aproximadamente cuatro veces la edad del universo calculada actualmente.

Durante estas pruebas las funciones especificadas deben probarse individualmente y en combinación, para ello es muy útil construir una matriz de pruebas que tenga en un eje las funciones que serán probadas y los casos de prueba en el otro. Si un caso ejercita una función particular se marca en la intersección. De esta manera las funciones individuales y sus combinaciones se muestran de una forma manejable. La elección de casos de prueba relevantes sobre la base de funcional representa el método mas efectivo de probar un sistema.

funciones	casos de prueba			
	c1	c2	c3	c4
f1				
f2				
f2 y f1				

Dicotomías:

Estructura vs. Función

Las pruebas pueden diseñarse desde un punto de vista estructural o funcional.

En las pruebas funcionales el sistema es tratado como una caja negra. Está sujeto a entradas y sus salidas son verificadas para saber si conforman los objetivos establecidos. Intuitivamente representan el punto de vista del usuario final.

Inversamente las pruebas estructurales se dirigen a los detalles de realización, como el estilo de programación, el método de control, el lenguaje fuente, el diseño de la base de datos, etc.

Los buenos sistemas están construidos en capas, del exterior al interior, el usuario final sólo ve la capa más externa, la capa de la función pura. El límite entre estructura y función es borroso, cada capa interior está menos relacionada con la función del sistema y más comprometida con su estructura, por lo tanto, lo que es función en una capa es estructura para la siguiente.

Las pruebas funcionales pueden en principio detectar todos los errores, pero tomaría tiempo infinito; las estructurales son inherentemente finitas pero no pueden detectar todos los errores. El arte de probar radica en parte en hacer elecciones juiciosas entre pruebas estructurales y funcionales.

2.3 Tipos de pruebas

Las pruebas se dividen en tres tipos:

Unitarias	:	las que verifican y validan la ejecución correcta de las funciones atómicas del sistema.
Modulares o de cadena	:	las que verifican y validan la ejecución correcta del enlace de unidades.
Integrales o de sistema	:	las que verifican y validan la ejecución correcta del enlace de módulos.

Pruebas unitarias

Las que prueban las rutinas individuales que realizan una función completa y su tamaño no excede de un cuarto de página. Actualmente estas rutinas se presentan como bibliotecas, macroinstrucciones o subrutinas en los lenguajes de acoplamiento.

una rutina que abre un archivo

Estas pruebas se enfocan a lo siguiente:

información de entrada	:	identificar el conjunto de unidades elementales, las relaciones entre éstas, y su asociación con variables del sistema.
verificación de unidades	:	identificar las unidades en las cuales se interpretan los resultados de los cálculos de los programas del sistema (metros, watts, litros,etc.).
información de salida	:	detectar todas las operaciones que involucran variables que no son conmensurables, esto depende de las posibilidades de especificación del lenguaje de programación que se use.

Pruebas modulares

Desde el punto de vista del análisis estructurado, el diseño y la programación deben hacerse de manera estructurada, lo que significa trabajar en base a partes lógicamente autocontenidas (módulos), dividiendo así de manera clara funciones distintas.

Un módulo bien construído sólo tiene un punto de entrada y uno de salida. En varios lenguajes una subrutina es equivalente a un módulo, en algunos lenguajes se permiten múltiples puntos de entrada y salida.

El propósito de la programación modular es descomponer una tarea compleja en subtareas más pequeñas y simples, lo cual facilita la escritura de sistemas correctos. El tamaño de un módulo debe ser a lo más de dos páginas, un sistema construído a base de módulos es mucho más simple de diseñar, programar, probar y mantener que uno que no es modular.

Las pruebas modulares contemplan los mismos puntos que las unitarias e incluyen uno más:

acoplamientos intramodulares : Las interacciones de un módulo requieren de supervisión estricta. Con frecuencia se abusa de las interacciones entre módulos (intermodulares), consideradas como una transferencia de control en el espacio de instrucción o de los datos, aumentando así la complejidad del software, estas pruebas deben cerciorarse de que el acoplamiento intramodular sea máximo.

Los puntos clave son:

Asegurarse de que todos los datos sean generados y usados por algún proceso y que todos los procesos usen datos.

Analizar los acoplamientos de los módulos en base al diseño que contiene la descripción de cada módulo en cuanto a la naturaleza de las entradas y salidas.

Esta información se especifica empleando declaraciones para indicar el número y orden de las entradas, tipos de datos, unidades y rangos.

La verificación de acoplamientos es efectiva para detectar esta clase de errores que en la práctica son muy difíciles de aislar.

Un paquete estadístico escrito en Fortran usa un sistema de acceso de archivos para recuperar los registros que contienen los datos que son necesarios para el realizar el análisis.

La recuperación del registro primario la realiza una subrutina sin fin, un número de línea en el programa que la llama recibirá el control en caso de error, sin embargo el programa falla al momento de suministrar el argumento requerido.

El compilador no puede detectar el error, más aún, la realización en Fortran es tal que el error no ocurre hasta que se solicita el regreso a la línea que no existe, en este momento el sistema 'truena'

Este error puede ser fácilmente detectado usando un acoplamiento en el diseño, ya que de esta forma es posible detectar que el número de parámetros es incorrecto.

Pruebas de integración

Estas pruebas se centran en los acoplamientos intermodulares buscando que éstos sean mínimos, se enfocan al paso de argumentos, tipos y parámetros. Se trata de detectar los siguientes casos:

- módulos que se usan pero que no se definen
- módulos que se definen pero que no se usan
- número incorrecto de argumentos
- tipos de datos que no concuerdan; i.e., el parámetro real y el formal
- restricciones que no concuerdan
- datos anómalos

Ya que estas pruebas involucran tanto el análisis de la consistencia y completez de la información como del control de flujo entre los módulos, se requiere de lo siguiente:

- una representación formal de los requerimientos del sistema
- una representación formal del diseño del sistema
- la codificación de los programas

2.4 Técnicas para obtener los objetivos

Existen dos técnicas para obtener los objetivos de una prueba:

1. Relacionar objetivos del sistema a las funciones del mismo, a partir de las funciones clave ((A)).

una consulta de un saldo debe realizarse en dos milisegundos

2. Relacionar objetivos del sistema a grupos de funciones (transacciones) del mismo, a partir de las transacciones clave del sistema ((D)).

un retiro de efectivo debe realizarse en medio segundo

En ambos casos se establece la lista de funciones y/o transacciones, así como los objetivos priorizados de cada una de ellas. Cada objetivo del sistema se define como una prueba.

Dicotomías:

Modularidad vs. Eficiencia

Pruebas y sistemas pueden ser modulares, un módulo es discreto, bien definido y relativamente pequeño.

Cada módulo implica acoplamientos con otros, lo que constituye fuentes de confusión. Entre más pequeños son los módulos, mayor es la verosimilitud de los errores de acoplamiento.

Los módulos grandes reducen los acoplamientos externos, pero contienen lógica complicada que puede ser difícil o imposible de entender. Los casos de prueba independientes y pequeños tienen la virtud de ser repetibles con facilidad, lo cual es muy útil, ya que si se encuentra un error como resultado de una prueba, sólo la prueba que se realizó debe repetirse para asegurar que éste ha desaparecido.

Parte de la maestría del desarrollo de software de alta calidad, radica en establecer tamaños de módulo y cotas de acoplamiento intermodular para balancear la complejidad interna y la de los acoplamientos externos. Se logra de esta forma una minimización global de la complejidad.

CAP. 3 Realización de las pruebas

Las pruebas son procedimientos formales, por lo cual las entradas deben prepararse, las salidas predecirse, los comandos ejecutarse y los resultados compararse y registrarse. Resultados inesperados indicarán la necesidad de revisar las pruebas, ya que éstas son un ambiente.

Probar es un proceso continuo en el cual creamos modelos mentales del ambiente, rutina o módulo (programa), naturaleza humana y de las pruebas mismas.

Del mismo modo que el software debe ser analizado, diseñado, programado y probado, las pruebas deben serlo.

El problema de probar radica en garantizar que un programa represente una función específica; el proceso de prueba consiste en obtener valores del dominio de la función o mensajes de error (si está fuera del dominio) que determinen el comportamiento esperado.

La descripción del comportamiento esperado es sin duda el componente más difícil de obtener, en casi todos los casos es necesario recurrir a métodos adecuados como son el cálculo manual, la simulación o el modelado.

3.1 Estrategias de prueba

Dependiendo de la fase de desarrollo se aplican distintas estrategias:

Pruebas de escritorio (Análisis y Diseño)

La revisión del análisis y la verificación del diseño deben tratarse en primer lugar en las pruebas. Las revisiones detalladas de cada uno de los requerimientos y la verificación cuidadosa de la lógica que los representan, por medio del procesamiento de datos de entrada supuestos (de prueba), puede conducir al descubrimiento de errores que de otro modo podrían aparecer durante las pruebas de los programas.

El intervalo de tiempo que transcurre, normalmente largo, entre el desarrollo de la lógica del sistema y la posibilidad de probarla en la computadora, justifican ampliamente este escrutinio considerando las serias consecuencias de un error de interpretación.

Pruebas unitarias y modulares (Programación)

Las pruebas de los programas aislados del resto del sistema deben hacerse en condiciones similares a las que se encontrarán en el sistema completo. Sólo aquellas modificaciones que sean estrictamente necesarias deben hacerse, ya que los cambios pueden introducir nuevos errores.

hacer de los módulos programas autosuficientes

Si el módulo a probar llama a otros módulos que aún no están listos, se crean módulos 'huecos' que regresen valores en el rango esperado. Para evitar la necesidad de escribir un programa de prueba para cada módulo, resulta muy útil desarrollar un módulo de prueba que genere valores y despliegue los resultados.

Los errores detectados durante las pruebas unitarias y modulares pueden ser fácilmente eliminados. Los detectados en etapas siguientes tienen consecuencias más serias, por esta razón estas pruebas deben ser ampliamente trabajadas.

Los elementos del dominio funcional se denominan entradas válidas, los programas deben probarse también con entradas inválidas. Los componentes esenciales para realizar estas pruebas son:

- la descripción del dominio funcional
- el programa en forma ejecutable
- la descripción del comportamiento esperado
- una forma de observar el comportamiento
- un método para detectar si el comportamiento observado corresponde al esperado

Pruebas de integración (Programación)

Estas pruebas deben llevarse a cabo en varios pasos; después de la introducción de un módulo nuevo debe establecerse que la ejecución probada previamente no sea alterada.

Esta fase en las pruebas de integración se conoce como prueba de regresión. Sólo cuando esta prueba de regresión ha sido realizada con éxito, las pruebas de las funciones nuevas deben iniciarse.

Al finalizar las pruebas de integración, la reacción del sistema a entradas incorrectas debe ser examinada.

Para lograr una cobertura más amplia de las pruebas es muy conveniente utilizar dos tipos de análisis:

Análisis estático : se basa en el exámen de la estructura del código fuente. Este análisis se compone de métodos que hacen uso de herramientas automáticas capaces de detectar anomalías en el flujo de datos o de concurrencia.

> PATHVU es un analizador estático comercial que mide complejidad y estructuración. Actualmente, únicamente trabaja sobre programas en COBOL 74 (ANSI).

Análisis dinámico : se basa en el exámen del comportamiento del código fuente en ejecución; combina pruebas de trayectorias y de evaluación simbólica.

> CHECKOUT es un analizador dinámico que se encuentra en el mercado y reporta los porcentajes de ramificaciones y trayectorias probadas (sólo para COBOL ANSI 74).

El análisis estático se enfoca sobre la forma y la estructura de la solución (sintaxis, acoplamiento de parámetros intermodulares, tipos, ilegibilidad, falta de estructuración, etc.) y no a los aspectos funcionales. Se usa especialmente para verificación.

El análisis dinámico se enfoca tanto a los aspectos funcionales como estructurales del programa, y determina además la completez de la prueba. Las métricas más usadas son número y porcentaje de líneas ejecutables, ramificaciones, y trayectorias. Se usa principalmente para validación y puede aplicarse manualmente a los requerimientos y a las especificaciones del sistema (Análisis y Diseño).

Debido a que cada tipo de análisis provee distinta información es indispensable hacer uso de ambos. La estrategia para integrarlos consiste en aplicar primero el análisis estático; ya que al hacer esto, se previenen errores al análisis dinámico, cuyo enfoque es el significado funcional. Aunque indirectamente puede detectar errores en la especificación.

3.2 Areas de responsabilidad

Durante el desarrollo de un sistema deben existir tres areas de responsabilidad para la ejecución de las pruebas, éstas son:

Equipo de desarrollo : se compone del grupo humano que que desarrollará la aplicación.

Equipo de certificación: se compone del grupo humano que desarrollará las pruebas.

Usuario : se compone del grupo humano que operará el sistema.

3.3 Etapas de prueba

Las pruebas se dividen en tres etapas:

Planeación : consta de las actividades siguientes:

- definición del guión
- determinación de las herramientas a usar y sus salidas útiles
- selección de formas

Ejecución : considera:

- la realización del guión
- el llenado de matrices y formas

Reporte : contempla:

- la elaboración del reporte
- el registro de errores.

Los momentos de prueba se dan en forma simultánea al ciclo de desarrollo de un sistema, considerando puntos distintos para cada fase, algunos puntos a considerar serían:

Durante la fase de análisis :

- la revisión de las especificaciones
- la identificación de controles internos
- la agrupación de los datos en categorías lógicas

Durante la fase de diseño :

- la revisión de descripciones y estimaciones
- la revisión de las matrices de módulos
- las estimaciones de consumo de recursos

Durante la fase de programación :

- los niveles de recuperación de errores
- los tiempos de respuesta
- el consumo de recursos

3.4 El plan de pruebas

Elaborar un plan de pruebas detallado y flexible es extremadamente útil durante el desarrollo de un sistema. Es tarea del plan de pruebas definir quiénes intervendrán en cada una de ellas, las funciones que serán probadas - y en que extensión -, en qué fase y con cuáles estrategias.

El plan debe ser elaborado por el equipo de certificación en la fase de análisis, ya que es necesario que en la siguiente ((D)), se tomen precauciones para que la filosofía de prueba pueda adherirse fácilmente.

La flexibilidad del plan debe demostrarse durante la marcha. El diseño de algunos casos de prueba se verá en ocaciones modificado por los resultados de las primeras pruebas.

Un plan de pruebas completo incluye lo siguiente:

- una lista con los nombres de las personas que intervendrán en cada prueba
- una lista con los objetivos de las pruebas
- las formas necesarias para documentar los resultados de las pruebas
- una lista con las funciones y transacciones a probar
- la preparación de las matrices de prueba
- una descripción de las estrategias que se usarán

- una lista de las espectativas que se cubrirán (porcentaje de ramificaciones y trayectorias, rangos de variables, etc.)
- una lista de los requerimientos de rendimiento esperados para el sistema completo y sus componentes (tiempos de respuesta, utilización de procesador, etc.)
- las tasas de error que deben alcanzarse antes de clasificar al sistema como liberado
- una lista de todos los errores que el sistema debe ser capaz de detectar, con la descripción de cómo activarlos y sus consecuencias
- una definición de cómo los resultados de prueba serán documentados así como un esquema clasificado para los errores
- una lista de todos los valores de entrada a usar (datos de prueba) y la formulación de los resultados esperados
- una estimación de la duración que tendrá cada una de las pruebas

Ejecución de las Pruebas:

La realización del plan debe hacerse de acuerdo al tipo de desarrollo; i.e., si el desarrollo es de arriba hacia abajo (top down), las pruebas deben realizarse de ese modo.

Los datos de prueba deben demostrar y ejercitar externamente las funciones visibles, las estructuras del programa, los datos y las funciones internas, incluyendo casos de prueba imposibles o improbables.

Las actividades clave son:

- reunir al equipo de pruebas
- agotar la lista de funciones y transacciones de acuerdo a sus objetivos
- llenar las matrices de prueba y las formas

Reporte

Para analizar las salidas, deben compararse los resultados obtenidos con los resultados esperados. Esto puede hacerse manual o automáticamente si se tiene un comparador.

El registro de errores es de gran importancia, ya que las pruebas podrán perfeccionarse de forma real gracias a la información obtenida.

Idealmente la creación de un sistema de información con este propósito tendría un fuerte impacto en el diseño de pruebas cada vez más adecuadas, siguiendo este orden sería posible tipificarlas estimando su cobertura.

CAP. 4 Errores

Errores de hardware y software

Los errores de software son aquellos que resultan del desarrollo de un sistema.

Los errores de hardware son el resultado del funcionamiento incorrecto de la máquina.

Aunque en principio esta distinción es clara, existen casos frontera que cada vez son más difíciles de clasificar debido a que los límites entre hardware y software ya llegan a confundirse.

> un error en un microprograma de una memoria sólo para lectura (ROM) del procesador central causa en algunos casos la ejecución incorrecta del hardware.

4.1 Clasificación de errores

Un análisis y clasificación exactos de los tipos de de error que pueden aparecer en los sistemas son de gran importancia para el desarrollo de sistemas confiables.

Después de tal análisis y clasificación es posible introducir métricas. De esta forma se reduce la verosimilitud de ciertos tipos de error, o se minimizan sus efectos. Con objeto de evitar problemas semánticos se da la definición siguiente:

Error : desviación entre el comportamiento intentado y el observado.

Un error puede ser reconocido sólo cuando las condiciones se desvían de una norma predefinida, por tanto pierde su significado en ausencia de una norma. Cuando normas diferentes se aplican durante la evaluación de las condiciones, es posible clasificar como correcto o erróneo en función de la norma dada.

Para distinguir entre el efecto y la causa de un error hablaremos de síntomas y causas. Saber que un programa es incorrecto no implica conocer el error: errores diferentes pueden tener las mismas manifestaciones y un error puede tener muchos síntomas. Los síntomas y las causas pueden ser confundidos, sólo mediante el uso de pruebas unitarias podrán diferenciarse.

Se intenta clasificar desde varios puntos de vista a los errores que pueden aparecer en un sistema de software, como base para un tratamiento unificado de los diferentes tipos de error que pueden establecerse.

Clasificación de acuerdo a los síntomas:

Esta clasificación es particularmente significativa ya que puede ser llevada a la práctica directamente; en lugar de utilizar diagnósticos de error difíciles y complicados. Cuando se hace en conjunción con el dominio de entrada de un sistema de software se ilustra así:

reacción del sistema	entrada fuera del dominio	- entrada dentro del - dominio
entrada rechazada	correcto	error Tipo I
resultados erróneos	error tipo II	error tipo III
caída del sistema	error tipo IV	error tipo V

Las consecuencias de un tipo particular de error dependen en gran medida de los requerimientos de la aplicación en particular. En general, los errores tipo I son los menos serios, una excepción se encuentra en los sistemas en tiempo real; donde se debe reaccionar a una entrada dentro de un período de tiempo predefinido, en tales sistemas el rechazo puede producir resultados catastróficos.

Los errores tipo II a tipo V son más serios, en el mejor de los casos generan resultados equivocados, volviendo inservible la aplicación.

Las pruebas estructurales son un método muy efectivo para detectar errores tipo IV y V, ya que tienen su causa en efectos laterales no intentados. Aparecen como resultado de la ejecución de todas las instrucciones del sistema.

Clasificación de acuerdo a sus causas:

Cualquier error en un sistema es determinado por una de las siguientes causas:

desarrollo : cuando independientemente de la operación correcta del hardware y de entradas válidas los resultados no convergen a donde se esperaba.

degradación del hardware: cuando un componente del hardware no conforma la especificación relevante debido al tiempo (edad) o al ambiente.

datos de entrada : cuando el dato de entrada no es correcto o el usuario se equivoca.

En un espacio de error, el desarrollo del sistema, la degradación del hardware y los datos de entrada forman un sistema independiente. Esta clasificación presupone un diagnóstico completo y adecuado de los errores.

Una de las tareas del desarrollo de un sistema, es la definición de la reacción del mismo en el evento de datos de entrada inválidos.

4.2 Tendencias del error

Los estudios sobre el tema muestran que entre el 45% y el 60% de los errores de software, se originan en la fase de especificación de requerimientos ((A)). Por tanto, se justifica que el proceso de pruebas a las especificaciones sea formal.

Los mismos estudios muestran que los errores tienden a encerrarse dentro del sistema de software, los módulos propensos a errores requieren de un cuidadoso escrutinio.

Un corolario a esta observacion es que el número de errores detectados en una pieza de software incrementa la probabilidad de la existencia de más errores no detectados. "La probabilidad de errores adicionales en una sección es proporcional al número de errores encontrados en la misma".

Los datos deben ser separados y marcados cuando producen errores para identificar los puntos débiles y estar en posición de brindar atención especial a la identificación de aquellos que detrimenten el sistema global.

A continuación se listan los errores más frecuentes en un sistema de software, de acuerdo a la fase del desarrollo en que se generan.

Errores que ocurren en la fase A:

- reglas de operación inadecuadas o incompletas
- criterios de ejecución inadecuados o incompletos
- ambiente de información inadecuado o incompleto
- información de la misión del sistema inadecuada o incompleta.
- requerimientos incompatibles

Se reporta que el 89% de los errores en los requerimientos caen en alguna de estas categorías. Otros errores son:

. requerimientos incompletos u omitidos
. requerimientos que no se puedan probar
. requerimientos incorrectos
. descripción inadecuada del ambiente de los datos
. definiciones erróneas de los acoplamientos externos
. inconsideración del entrenamiento del usuario
. especificaciones vagas de las funciones
. comprensión inadecuada de las necesidades del usuario

Errores que ocurren en la fase D:

- secuencia lógica erronea
- resultados requeridos, procesados de forma inexacta
- falla en las rutinas de entrada o salida de parámetros
- falla en las rutinas para aceptar todos los datos dentro del rango posible
- falta de límites y validación para los datos de entrada
- procedimientos de recuperación inadecuados o no considerados
- procesamiento requerido insatisfecho

Esto constituye el 85% de los defectos de diseño.

Errores que ocurren en la fase P:

. secuencia o decisión lógica inadecuada o errónea

. cálculos aritméticos-lógicos inadecuados o erróneos

. ramificaciones erróneas

. pruebas realizadas de forma incorrecta

Se estableció que el 70% de los errores de codificación caen en alguna de las mencionadas categorías. Adicionalmente existen los siguientes:

. indefinición del término de un ciclo

. violación de las reglas del lenguaje de programación

. violación de los estándares de programación

. interpretación parcial de las construcciones del programa por el programador

. errores tipográficos

. errores de almacenamiento

. esquemas de iteración fallidos

. formatos de entrada y salida

. violación de parámetros o índices

. subrutinas sin término

. errores en el procesamiento de mensajes de error

. errores de acoplamiento con otro software, base de datos,etc.

. errores de sintaxis

. errores de inicialización

. errores en los contadores

. definiciones múltiples y variables indefinidas

. errores de dimensionamiento

. errores de compilación

. errores en operaciones de punto flotante

. errores de decisión

Errores en los datos

. valores antiguos o erróneos

. almacenamiento de datos inadecuado o erróneo

. variables olvidadas

Esto constituye el 89% de éstos errores

Los errores se ubican en los siguientes tipos de problemas:

tecnológicos	: definición del problema, información disponible, recursos y facilidades de comunicación de datos.
organizacionales	: division del trabajo, información disponible, recursos y facilidades de comunicación interpersonal.
históricos	: historia del proyecto y del sistema, situaciones especiales y participación de asesores externos.
dinámica de grupo	: capacidad y distribución de roles.
individuales	: experiencia, capacidad y formación de cada analista, diseñador y programador.
otros	: causas no clarificadas.

CAP. 5 Medición de las pruebas

El primer paso en el desarrollo de productos de software de alta calidad es reconocer los defectos que se tienen.

Con objeto de reducir la inconfiabilidad y la complejidad del software, y por tanto el costo del desarrollo y mantenimiento de los sistemas es necesario reemplazar juicios subjetivos por métricas cuantitativas.

Los principales tipos de medición se definen de acuerdo a su base en:

funcional : se enfoca al patrón de datos de entrada y salida.

caracteristica : se enfoca al comportamiento.

estructual : se enfoca al patrón del control de flujo. Como el patrón puede representarse como una gráfica dirigida, la mayoría de las métricas se obtienen a partir de la teoría de gráficas.

5.1 Factores y criterios

A partir de los tipos de medición se proponen tres factores:

Funcionalidad (F1): se orienta a la satisfacción de los requerimientos del usuario.

Eficiencia (F2): se orienta al rendimiento de las funciones y al consumo de recursos.

Facilidad de mantenimiento (F3): se orienta al esfuerzo requerido para localizar y corregir errores, o para efectuar modificaciones.

Para el factor funcionalidad (F1), se consideran los siguientes criterios:

F1C1 satisfacción de los requerimientos del usuario

F1C2 tolerancia a errores

Para el factor eficiencia (F2), se consideran:

F2C1 comportamiento

F2C2 consumo de recursos

Para el factor facilidad de mantenimiento (F3), se consideran:

F3C1 estructuración

F3C2 generalidad

5.2 Métricas

De acuerdo a las fases de desarrollo, las métricas consideradas están asociadas a los factores y criterios antes mencionados, éstas son:

Para la fase de análisis, el factor funcionalidad y el criterio satisfacción de los requerimientos del usuario (AF1C1):

M0 resumen de los principales requerimientos del sistema

M1 categorización de los requerimientos

M2 descripción narrativa de las funciones principales

M3 diagrama de estructura que muestre la división lógica en módulos

M4 identificación y análisis de controles internos

M5 agrupación de datos en categorías lógicas

M6 definición de procesos manuales y automatizados

M7 definición de la cobertura geográfica (local, regional, nacional, etc.)

M8 especificación de requerimientos de hardware

M9 visto bueno del usuario

Para Análisis, Funcionalidad, Tolerancia a errores (AF1C2):

M0 especificación de tolerancia a errores en las entradas

M1 especificación de los requerimientos de la recuperación de errores de hardware

M2 especificación de los requerimientos de la recuperación de errores de software

M3 especificación de los requerimientos de la recuperación de errores de operación

Para Análisis, Eficiencia, Comportamiento
(AF2C1):

M0 definición de las transacciones en línea del sistema

M1 definición de las transacciones 'batch'

M2 especificación de requerimientos de tiempos de respuesta por transacción en línea

M3 especificación de requerimientos de duración de procesos

M4 definición de estructuras de datos del sistema y sus relaciones

Para Análisis, Eficiencia, Consumo de recursos
(AF2C2):

M0 identificación del volumen de transacciones que el sistema atenderá

M1 identificación del volumen de registros

M2 proyección del volumen de operaciones que atenderá

M3 proyección del volumen de registros por estructura de datos

M4 identificación del volumen de impresión del sistema

Para Análisis, Facilidad de mantenimiento, Estructuración (AF3C1):

M0 uso de análisis estructurado

M1 uso de diagramas de flujo de datos

M2 uso de diccionario de datos

M3 uso de miniespecificaciones de funciones

M4 uso de la técnica de análisis de controles internos

M5 integración de la documentación resultante

Para Análisis, Facilidad de mantenimiento, Generalidad (AF3C2):

M0 identificación de funciones del sistema que puedan hacer uso de módulos generales (bibliotecas)

M1 identificación de módulos que puedan ser usados en diferentes partes del sistema

M2 identificación de módulos que puedan ser usados en diferentes partes de un programa

M3 identificación de bases de datos generales que puedan ser útiles

Para Diseño, Funcionalidad, Satisfacción de requerimientos del usuario (DF1C1):

M0 definición del alcance, objetivos y requerimientos del sistema

M1 especificación de procesos manuales y automatizados

M2 para cada función del sistema:

definición de entradas, procesos, salidas y pruebas

M3 definición de formatos de reportes y pantallas

M4 definición de las transacciones asociadas a cada pantalla

M5 definición de formatos de registros para entradas y salidas batch

M6 especificación de requerimientos de hardware nuevo

M7 definición de los procedimientos de integración con con otros sistemas

M8 matriz que relaciona los requerimientos del usuario a los módulos que los instrumentan

M9 visto bueno del usuario

Para Diseño, Funcionalidad, Tolerancia a errores (DF1C2):

M0 definición de control de condiciones de error en el sistema

M1 definición de la forma de recuperación de errores de hardware

M2 definición de la forma de recuperación de errores de software

M3 definición de la forma de recuperación de errores de operación

Para Diseño, Eficiencia, Comportamiento
(DF2C1):

M0 especificación de volúmenes de información del sistema

M1 especificación de tiempos de respuesta

M2 evaluación del uso de archivos convencionales vs. el uso de base de datos

M3 modelado del comportamiento de la base de datos

Para Diseño, Eficiencia, Consumo de recursos
(DF2C2):

M0 estimación del consumo de procesador

M1 estimación del consumo de memoria

M2 estimación de la actividad de 'I/O'

M3 estimación de la duración de procesos

M4 especificación de volúmenes de impresión

M5 especificación del espacio requerido en cinta

M6 especificación del espacio requerido en disco

Para Diseño, Facilidad de mantenimiento, Estructuración (DF3C1):

M0 diagrama que muestre los procesos que integran el sistema

M1 diagrama que muestre los archivos, bases de datos y programas

M2 definición del diccionario de datos

M3 definición detallada de cada programa

M4 diseño de formatos

M5 clasificación de procesos de acuerdo al tipo de de función

M6 definición de módulos diferentes para cada función sistema

M7 estimación del tamaño de los módulos

M8 definición del paso de parámetros entre módulos

Para Diseño, Facilidad de mantenimiento, Generalidad (DF3C2):

M0 diseño de funciones del sistema haciendo uso de módulos generales (bibliotecas)

M1 diseño de módulos que puedan ser usados en diferentes partes del sistema

M2 diseño de módulos que puedan ser usados en diferentes partes de un programa

M3 diseño que haga uso de bases de datos generales

Para Programación, Funcionalidad, Satisfacción de los requerimientos del usuario (PF1C1):

M0 nivel de integración del manual del usuario

M1 para cada función:

- funcionamiento de entradas y salidas interactivas

M3 para cada función:

funcionamiento de entradas y salidas batch

M4 funcionamiento de procesos

M5 visto bueno del usuario

Para Programación, Funcionalidad, Tolerancia a errores (PF1C2):

M0 nivel de control de condiciones de error en el sistema

M1 nivel de recuperación de errores de entradas

M2 nivel de recuperación de errores de hardware

M3 nivel de recuperación de errores de software

M4 nivel de recuperación de errores de operación

Para Programación, Eficiencia, Comportamiento
(PF2C1):

M0 tiempo de proceso por transacción

M1 tiempo de 'I/O' por transacción

M2 tiempo transcurrido por transacción

M3 bases de datos:

.relación entre búsquedas y lecturas
.porcentaje de superposiciones (overlays)
.promedio de espera en escrituras
.promedio de espera por sincronía

M4 tamaños de bloque de los archivos

M6 ejecución de funciones adecuadas para procesos en línea

M7 ordenamientos (sorts)

Para Programación, Eficiencia, Consumo de recursos
(PF2C2):

M0 uso de procesador en línea

M1 uso de memoria en línea

M2 uso de 'I/O' en línea

M3 uso de memoria en 'batch'

M4 uso de procesador en 'batch'

M5 uso de 'I/O' en 'batch'

M6 uso de periféricos

(disco, impresora, cinta magnética)

Para Programación, Facilidad de mantenimiento, Estructuración (PF3C1):

M0 nivel de uso de nomenclatura externa

M1 nivel de uso de nomenclatura interna

M2 nivel de uso del lenguaje establecido

M3 codificación de formatos de pantallas

M4 uso del manejador de base de datos

M5 uso del 'header' de comunicaciones

M6 documentación

Para Programación, Facilidad de mantenimiento, Generalidad (PF3C2):

M0 nivel de uso de bibliotecas generales

M1 nivel de uso de bibliotecas del sistema

M2 nivel de uso de módulos en diferentes partes de un programa

M3 nivel de uso de bases de datos generales

5.3 Pesos y marcas

Esta técnica de evaluación consiste en lo siguiente:

Dado un conjunto de n elementos, se construye una matriz cuyos renglones y columnas son los elementos del mismo. Cada renglón se compara contra todas las columnas (por razones obvias se excluye la diagonal) y se marca en la intersección de la siguiente manera:

renglón-columna: si se prefiere al elemento del renglón que se está comparando.

columna-renglón: si se prefiere al elemento de la columna con el cual se compara.

La comparación se realiza de acuerdo a un criterio dado y se van anotando los totales (ti) de cada renglón (i=1,...,n). Al terminar de comparar el renglón n-1 se obtienen los pesos (en porcentaje), considerando que n(n-1)/2 = 100%. Todos los elementos del conjunto deben elegirse al menos una vez, de ot: modo alguno quedaría eliminado.

DF3C2	M0	M1	M2	M3
M0		*	*	
M1			*	
M2				*
M3	*	*		
Total=10	6	———		100%
ti's:	(1,2)	———		x

METRICA	(%) PESO	EVALUACION (E,B,S,I,N)	MINIMO ACEPTABLE	(%) PESO ASIGNADO
AF1C1M0				
M1				
M2				
M3				
M4				
M5				
M6				
M7				
M8				
M9				
C2M0				
M1				
M2				
M3				
F2C1M0				
M1				
M2				
M3				
M4				
C2M0				
M1				
M2				
M3				
M4				
F3C1M0				
M1				
M2				
M3				
M4				
M5				
C2M0				
M1				
M2				
M3				

Forma 1

E = EXELENTE
B = BUENO
S = SUFICIENTE
I = INSUFICIENTE
N = NO APLICA

FASE FACTOR Y CRITERIO	OBSERVACIONES	(%) PESO	MIN.	EVAL.	(%) PESO ASIG.
AF1C1					
C2					
F2C1					
C2					
F3C1					
C2					

Forma 1'

METRICA	(%) PESO	EVALUACION (E,B,S,I,N)	MINIMO ACEPTABLE	(%) PESO ASIGNADO
DF1C1M0				
M1				
M2				
M3				
M4				
M5				
M6				
M7				
M8				
M9				
C2M0				
M1				
M2				
M3				
F2C1M0				
M1				
M2				
M3				
M4				
C2M0				
M1				
M2				
M3				
M4				
M5				
M6				
F3C1M0				
M1				
M2				
M3				
M4				
M5				
M6				
M7				
M8				
C2M0				
M1				
M2				
M3				

Forma 2

FASE FACTOR Y CRITERIO	OBSERVACIONES	(%) PESO	MIN.	EVAL.	(%) PESO ASIG.
DF1C1					
C2					
F2C1					
C2					
F3C1					
C2					

Forma 2'

METRICA	(%) PESO	EVALUACION (E,B,S,I,N)	MINIMO ACEPTABLE	(%) PESO ASIGNADO
PF1C1M0				
M1				
M2				
M3				
M4				
M5				
C2M0				
M1				
M2				
M3				
M4				
F2C1M0				
M1				
M2				
M3				
M4				
M5				
M6				
M7				
C2M0				
M1				
M2				
M3				
M4				
M5				
M6				
F3C1M0				
M1				
M2				
M3				
M4				
M5				
M6				
C2M0				
M1				
M2				
M3				

Forma 3

FASE FACTOR Y CRITERIO	OBSERVACIONES	(%) PESO	MIN.	EVAL.	(%) PESO ASIG.
PF1C1					
C2					
F2C1					
C2					
F3C1					
C2					

Forma 3'

Conclusiones

El lector que ha seguido con detalle el desarrollo de este trabajo, quedará convencido que el emplear una metodología correcta, reduce de manera significativa, el número de errores que podrían aparecer al implantar un sistema de software.

Emplear las herramientas automatizadas tales como: DTG, PATHVU y CHECKOUT -descritas anteriormente-, hacen posible la detección de los errores en fases, donde es mas sencillo eliminarlos.

El registro y la clasificación de los errores, permite tener un conocimiento más amplio del sistema y ayuda a la correción de los mismos, cuando aparecen durante su ciclo de vida.

Resta comentar un poco sobre los sistemas.La definición de un conjunto inicial de métricas para medir la calidad de un sistema, es el punto de partida que conduce a estándares de desarrollo.

Podría aventurarse que esta metodología ha sido creada sólo en el escritorio de trabajo ¡Sin embargo, se ha demostrado que la aplicación de la misma (al menos en el Banco Nacional de México), ha mejorado la calidad de los sistemas hasta en un 70%.Desde luego la búsqueda de una metodología ideal está muy lejos de nuestro alcance;acaso esto sea también una muestra de cómo el conocimiento humano supera al mismo ser humano.Nuestro paso por la vida es lo suficientemente insignificante para resolver la magnitud de nuestras limitaciones.

Bibliografía

Software Reliability
H. Kopetz
Macmillan Computer Science Series (1979)

Software System Testing and Quality Assurance
Boris Beizer
Van Nostrand Reinhold Company (1985)

Software Testing Techniques
Boris Beizer
Van Nostrand Reinhold Company (1983)

Standard for Testing Aplication Software
William E. Perry
Auerbach Publishers (1985)

Techniques of Program and System Maintenance
Girish Parikh
Winthrop Computer System Series (1982)