Capítulo 6 Pruebas de sistema

Cualquier pieza de software completo, desarrollado o adquirido, puede verse como un sistema que debe probarse, ya sea para decidir acerca de su aceptación, para analizar defectos globales o para estudiar aspectos específicos de su comportamiento, tales como seguridad o rendimiento. A éste tipo de pruebas donde se estudia el producto completo se les llama ***Pruebas de Sistema***.

La prueba de sistemas usualmente es de caja negra, especialmente si quien prueba no tiene acceso al código fuente del producto a probar, que es lo más frecuente.

En este capítulo se trata el proceso de prueba de sistema, incluyendo su relación con el proceso de obtención de requerimientos y el resto del proceso de desarrollo de software.

6.1 Visión sistémica de la prueba

Cuando se deben realizar pruebas, debe mantenerse un enfoque sistémico, es decir integral, que está detrás de todo desarrollo de software. Al hablar de enfoque sistémico se indica que:

a) Todo sistema tiene una serie de objetivos que le dan sentido. Esos objetivos están asociados con indicadores de éxito que permiten determinar si los objetivos se cumplen y en qué medida.
b) Todo sistema tiene una serie de elementos que lo forman y la interacción de tales elementos se orienta a satisfacer los objetivos.
c) Todo sistema tiene una frontera que lo separa de un medio ambiente. Los elementos de ese medio ambiente influyen sobre el sistema proporcionándoles una serie de entradas y obteniendo del mismo un conjunto de salidas.
d) Ningún sistema existe en aislamiento; siempre interaccionan con otros sistemas constituyendo un sistema mayor.

Al aplicar esos conceptos a la prueba de software, se obtienen una serie de principios que servirán de base para la prueba:

1. Debe asegurarse de conocer con precisión los objetivos del software a probar, así como sus indicadores de éxito. Éstos elementos se localizan en los documentos obtenidos en la etapa de recolección de requerimientos, así como en las especificaciones del software. Esta información será indispensable para preparar el plan de pruebas y será la base para iniciar el desarrollo de los casos de prueba.
2. Deben determinarse las entradas y salidas del sistema aprobar. Éste aspecto es necesario en la preparación de los casos de prueba y

también en el establecimiento de procedimientos de prueba, orientados especialmente a los casos de prueba que muestran el cumplimiento de los objetivos.

3. Considerar el sistema mayor donde opera el software a probar. Generalmente es un ambiente organizacional, formado por elementos de hardware, de software y personas (usuarios). Todos éstos elementos influyen mucho sobre el sistema y ayudan especialmente en la preparación de casos de prueba de situaciones no deseadas, relacionadas con datos inadecuados, ausencia de elementos necesarios y ocurrencia de excepciones.

6.2 Vista general de la prueba de sistemas

El proceso de prueba de un sistema tiene dos etapas que pueden estar muy separadas en el tiempo: la preparación de las pruebas y la aplicación de las mismas. La primera está muy ligada a la obtención de requerimientos, por lo que ocurre en las primeras etapas del proyecto, mientras que la segunda requiere del sistema completo o al menos una ***integración***, como se denomina a un producto parcial, aún no liberado, para poder aplicar las pruebas, por lo que ocurre en etapas avanzadas del proyecto. La situación exacta de estas partes depende del modelo de ciclo de vida que se haya elegido.

La etapa de preparación de pruebas incluye al menos tres actividades:

a) preparar un plan de pruebas,
b) preparar una lista de verificaciones de los requerimientos y
c) preparar casos de prueba.

Para ejecutar la segunda y la tercera actividades se requiere contar con el documento de requerimientos.

La primera etapa de pruebas provee retroalimentación para el análisis de requerimientos, identificando huecos, ambigüedades y otros problemas. También provee valiosas sugerencias para el diseño y la implementación del sistema, si apenas está desarrollando.

La etapa de aplicación de pruebas requiere del plan de pruebas y de una versión del sistema que sea ejecutable (una integración). Sobre ésta se aplicarán los casos de prueba que se prepararon, se analizarán los resultados y se identificarán posibles defectos.

Esta segunda etapa provee retroalimentación a la implementación y al diseño, mostrando posibles defectos que deben ser corregidos. También provee información que será de utilidad en la liberación del sistema, su aceptación, la estimación de su confiabilidad y para su mantenimiento.

En la Figura 6.1 se muestra el proceso de prueba de sistema y su relación con otros procesos.

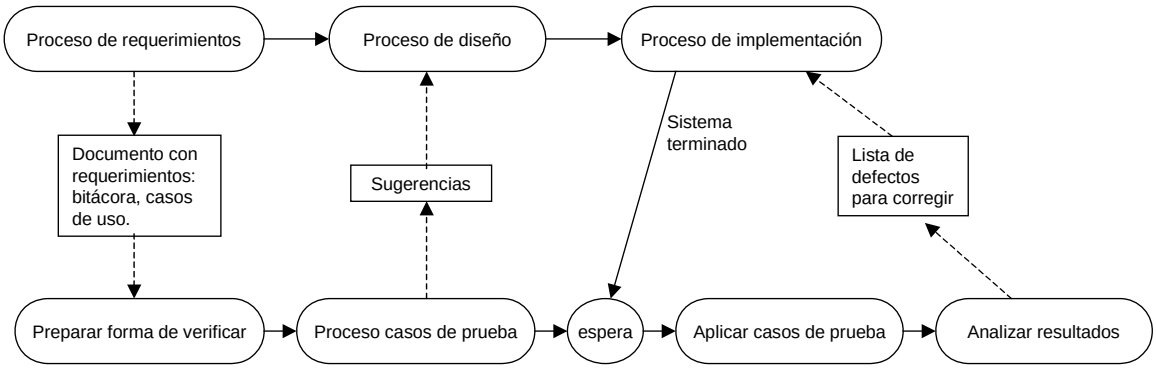

Figura 6.1 Proceso de prueba de sistema

La prueba de sistemas tiene varias suposiciones importantes:

a) Cada unidad que forma el sistema ha sido probada por separado y se han eliminado sus defectos.

b) Las interfaces humano-computadoras (de texto o gráficas) han sido probadas también por separado.

c) Se han realizado pruebas de integración para analizar la interacción entre partes del sistema y se han eliminado los defectos identificados.

El segundo punto es importante, ya que algunas veces se confunde la prueba de sistema con la prueba de la interfaz. La primera verifica la interacción de todas las partes, mientras que la segunda únicamente analiza los elementos de la interfaz y posiblemente el manejo de eventos asociados. Sin embargo, las herramientas que ayudan a la prueba de interfaz pueden utilizarse para iniciar las pruebas de sistema.

6.3 Plan de prueba

En la sección anterior se dio una idea general de lo que es un caso de prueba y cómo se utilizan en la práctica informalmente. Surgen varias preguntas: ¿cuántos casos serán suficientes? ¿cómo generar los menos posibles? ¿qué valores son adecuados?

6.3.1 Plan de pruebas

El plan de pruebas es un documento muy importante dentro del proceso de prueba del software. En él se explican los propósitos y enfoques de las pruebas, el plan de trabajo, los procedimientos operacionales, las herramientas necesarias y las responsabilidades. La extensión y detalle del plan debe adecuarse al proyecto y a las necesidades de la empresa, pudiendo usarse como guía el estándar IEEE 829. A continuación se muestra una propuesta mínima de contenido, para proyectos pequeños y medianos.

a) Identificación del plan de pruebas y el sistema al que se aplica

b) Elementos a probar: qué módulos, clases, casos de uso se van a probar; cuando se emplea desarrollo iterativo, deben especificarse las prestaciones

(funcionalidades) a probar y cuáles no se probarán (ya sea que se probaron antes o que se implementarán después).

c) Enfoque: vista general de la estrategia de prueba.
d) Criterio de aceptación o rechazo de un caso de prueba: criterio para dar por bueno o malo un caso de prueba al ser ejecutado.
e) Criterio de suspensión: ya sea por tiempo o por cobertura.
f) Productos a entregar: desde el propio plan, los casos y procedimientos de prueba, los resultados.
g) Tareas a realizar para satisfacer el proceso.
h) Necesidades ambientales: hardware, software y espacio de trabajo necesarios.
i) Responsabilidades: quién es responsable de cada cosa.
j) Personal necesario y si requieren entrenamiento.
k) Calendario: tiempos e hitos en el proceso.
l) Riesgos y contingencias que pueden ocurrir en el proceso de prueba.

6.4 Lista de verificaciones

Para comenzar el proceso de pruebas del sistema se parte del documento de requerimientos. En caso que no exista y se deba evaluar un sistema para aceptarlo o seleccionarlo de entre un conjunto, habrá que preparar dicho documento, aún cuando sea extemporáneo.

Un documento de requerimientos debe contener la lista de las funciones que se desea realice el software, describiéndolas y priorizándolas; también debe incluir los requerimientos no funcionales, que pueden incluir aspectos organizacionales, de rendimiento y otros.

Un documento de requerimientos bien preparado debe proveer, para cada requerimiento, una forma de verificar que se satisface. En el caso de las funciones, será una descripción y en caso de requerimientos no funcionales pueden ser especificaciones muy precisas, como puede ser el tiempo de respuesta.

Por el momento concentraremos la atención en los requerimientos funcionales, dejando los otros para una sección posterior.

La actividad de preparar lista de verificación incluye los pasos siguientes:

a) asegurarse que para cada requerimiento exista una descripción de la manera en que se verificará; si no existe, debe desarrollarse. Una buena descripción debe contener al menos el funcionamiento típico de la función a que corresponde y los principales comportamientos alternos: variaciones menos frecuentes, respuesta ante datos incompletos y fallas del ambiente.

b) revisión de las descripciones: cada descripción debe revisarse para asegurarse que se entiende claramente, que efectivamente es realizable.

Ejemplos.

1. Una función muy común en los sistemas de información es la de identificar al usuario pidiendo un nombre y una contraseña. Una manera de describir la forma de verificar sería la siguiente:

> "Se introduce el nombre y la contraseña de un usuario registrado y entonces se activa el sistema habilitando las opciones a que tenga derecho".

Una forma más completa podría agregar como casos alternos:

> "Se introduce un nombre y contraseña que no coinciden con un usuario registrado y entonces se activa una ventana donde se muestra un mensaje especificando el error cometido".

Suponga una función donde un cliente de una papelería desea saber que tipos de cuadernos existen y su precio de mayoreo, eligiendo el concepto "cuaderno" en un catálogo. La forma de verificar podría ser como ésta:

> "El cliente selecciona "Cuaderno" en el catálogo de productos y oprime el botón "Buscar"; el sistema mostrará una página con los diferentes tipos y marcas, con su precio al menudeo y su precio de mayoreo. Si no se oprime ningún botón en 30 segundos, aparecerá un mensaje que solicite elegir un producto y que indique que se oprima un botón".

Debe observarse que estas frases, que indican cómo verificar, no constituyen casos de prueba, aunque sí ofrecen una guía para prepararlos. Algunas recomendaciones para estas listas son las siguientes:

a) Comenzar siempre por el funcionamiento típico deseado, el que se considerará un éxito del sistema.
b) Si existen casos alternos aceptables, describirlos después.
c) Finalmente, describir casos que se consideran fracasos, pero que están (o deben estar) considerados en la programación, generalmente en forma de validaciones.
d) Escríbalos de manera clara y concreta, en tiempo presente y evite frases condicionales como: "podría oprimir" o "debería escribir un identificador"; en su lugar, escriba: "oprima ..." o "escriba identif".
e) Si existe algún requerimiento no funcional asociado con la verificación, asegúrese de anotarlo en el documento de requerimientos. Por ejemplo, en el caso de la identificación es común considerar que tiene un máximo de tres intentos.

Si utiliza una metodología que ayude a la recolección y manejo de requerimientos, es posible que esta forma de verificar ya esté escrita. Por ejemplo, la metodología Áncora (Sumano, 2006) utiliza una bitácora donde se exige anotar la manera de verificar cada quinteta que representa una acción iniciada por un papel (usuario). Algunos enfoques de Casos de Uso sugieren describir los pasos más importantes que se siguen en cada caso de uso, incluyendo caminos

alternos. Esta información es bastante similar a la que se indica como verificación. Estos dos casos se tratarán en las secciones siguientes.

Si la metodología que utiliza no incluye éste elemento o si el software ya existe y no se cuenta con un documento de requerimientos adecuado, usted puede preparar esa lista de verificaciones.

6.4.1 Detalle a partir de Áncora

La metodología Áncora permite identificar de manera precisa la funcionalidad requerida por el usuario y ofrece una serie de artefactos útiles en la preparación de pruebas. El principal de éstos es la Bitácora de Desarrollo.

La Bitácora de Desarrollo (ver Figura 6.2) contiene la lista de quintetas[1] diferentes que se identificaron en los Guiones de la propuesta computacional del sistema; para cada quinteta incluye un campo que indica cómo verificar y otro con el tiempo estimado de desarrollo. Para fines de las pruebas interesan los dos primeros.

Quinteta	Cómo verificar	Tiempo est.

Figura 6.2 Bitácora de Áncora

Usualmente el campo “cómo verificar” contiene únicamente una vaga descripción de lo que se hace y posiblemente incluye aspectos que corresponden más bien a requerimientos no funcionales, que no fueron identificados por separado. Entre éstos pueden incluirse número máximo de intentos para una operación o alguna costumbre de la organización donde operará el sistema.

Un ejemplo de cómo verificar es el siguiente:

Quinteta	Cómo verificar
Se anota compromiso en libreta	La secretaria selecciona una fecha y hora en la ventana de libreta electrónica y anota el compromiso; si no existen conflictos, el sistema confirma que quedó anotado. Si existe conflicto para esa fecha y hora o se omite algún dato, el sistema avisa y no lo deja registrado.

Como se indicó antes, esta forma de verificar debe revisarse y completarse, de modo que se asegure contar con:

[1] Una quinteta incluye un papel, una acción (verbo), un utensilio principal, un utensilio auxiliar opcional y una periodicidad opcional.

a) descripción de la acción iniciada por el papel (primer elemento de la quinteta) y la respuesta del sistema (lo que será visible y quizá parte no visible, como actualización de una base de datos).
b) descripción de variantes del casó típico, si los hubiera.
c) descripción de al menos un caso fracasado.

Los casos fracasados pueden incluir algunos de los siguientes:

1. Equivocación en los datos proporcionados por el papel
2. Falta de disponibilidad de algún componente del sistema (pieza de software, base de datos, etc)
3. Falla del hardware
4. Otro problema del ambiente del sistema

6.4.2 Detalle a partir de Casos de Uso

Una alternativa muy popular para definir la funcionalidad de un sistema es el empleo de casos de uso. Para éstos existe una notación semiformal dentro del estándar UML, orientada a su representación gráfica, pero existen muchas variantes acerca de los detalles que acompañan a dicha notación. En esta sección usaremos algunas ideas tomadas del método ICONIX [Rosemberg, 1999], del Proceso Unificado [Jacobson et al, 2000] y del enfoque de Select Perspective [Select, 2005], que se describen brevemente.

En todos los métodos que utilizan el Lenguaje Unificado de Modelado (UML), la descripción general de lo que hace el sistema se representa en un **Modelo de Casos de Uso**, el cual contiene actores, casos de uso y relaciones entre ellos.

Un Caso de Uso corresponde a una forma en que un actor utilizará el sistema, es decir, a una funcionalidad o prestación. Cada caso de uso se representa gráficamente como un óvalo con un texto que indica su función, usualmente un verbo o una frase verbal. Usualmente la forma gráfica no es suficiente para documentar las prestaciones de un sistema y se acostumbra agregar una descripción textual, que no es parte del estándar de UML. Algunas formas de realización son las siguientes:

a) **Proceso Unificado**: la forma gráfica del caso de uso se complementa con el **Flujo de Sucesos**, que es una descripción textual de la secuencia de acciones que forman el caso de uso. Puede haber varios niveles de detalle.
b) **ICONIX:** se recomienda describir en forma de texto la serie de acciones que indican la función que realiza el caso de uso. El autor recomienda iniciar con una sola frase eneral y luego detallarla.

c) **Select Perspective**: en este método (y su herramienta correspondiente) se describe cada caso de uso por medio de un diálogo donde se indican las acciones del actor y las respuestas del sistema, de manera alternada

Como puede verse, en realidad los tres constituyen una misma forma de texto, con diferentes detalles pero con el mismo concepto.

Ejemplos.

El siguiente es un ejemplo tomado de [Jacobson et al, 2000]

> "Caso de uso **Pagar Factura**:
>
> El comprador estudia la factura a pagar y verifica que se corresponde con el pedido original.
>
> El comprador planifica el pago de la factura por banco.
>
> Cuando vence el día de pago, el sistema revisa si hay suficiente dinero en la cuenta del Comprador. Si tiene suficiente dinero disponible, se realiza la transacción."

La descripción textual de los casos de uso contiene, en cierta medida, la información necesaria para verificar su cumplimiento. Si no fuera suficiente, deberán detallarse más.

6.5 Casos de prueba

La forma de verificar de las diversas funcionalidades de un producto de software, descritas en el formato de Áncora o en Casos de Uso, son el punto de partida para la preparación de casos de prueba y, en ocasiones, de procedimientos de prueba.

Las funcionalidades o prestaciones de un sistema pueden separarse en dos grupos:

a) aquellas que reciben un conjunto de entradas más o menos simultáneas y a partir de ellas generan un resultado y

b) las que requieren una serie de interacciones con el actor(usuario), en cada una de las cuales éste introduce una serie de datos.

El primer caso ocurre cuando las entradas deben completarse antes de que el sistema se lance a realizar una función, básicamente sin retroalimentación que pueda influir en el usuario. Es el caso de una sola variable, una sola acción a través de un botón o de un *Enter*, pero también incluye la lectura de listas de datos cuyo proceso se realiza cuando la lista ha terminado. Un ejemplo típico, en interfaces gráficas, donde resulta muy común que deban llenarse varios campos (por ejemplo nombre, dirección, teléfono y rfc) y al final se oprime un botón el cual indica al sistema que la entrada está completa y puede aplicar la función seleccionada.

El segundo caso ocurre cuando la entrada ocurre en varias etapas donde una de ellas genera una respuesta parcial del sistema, la cual influye en la

siguiente, ya que el usuario no puede indicar las entradas sin tomar en cuanta la salida intermedia. Sin embargo la función completa requiere de terminar todas las etapas. Un ejemplo de éste tipo es la siguiente entrada de una función de venta:

> "El empleado selecciona la opción "Consultar" y el sistema le muestra una lista de productos;
>
> El empleado marca los productos que le interesan y al terminar oprime el botón "Cotizar"; El sistema le muestra la lista de productos seleccionados y los precios de cada uno, así como la disponibilidad; también deja un espacio a la derecha de cada producto para anotar la cantidad deseada;
>
> El empleado marca la cantidad deseada de cada uno; los no deseados los marca con un cero. Al concluir oprime el botón "Listo"; el sistema genera una orden y le solicita que indique la forma de pago;
>
> El empleado marca "Tarjeta" y escribe el número y la compañía y oprime "Termina"; el sistema verifica los datos con el banco y, si está de acuerdo, envía mensaje y un muestra un botón "Imprimir factura";
>
> El empleado oprime el botón "Imprimir factura" y el sistema lo hace."

En el ejemplo es claro que existen cinco etapas de entrada de datos para completar una sola función. Lo que el empleado puede ingresar en cada una depende de la anterior.

En las tres subsecciones siguientes se describen los aspectos generales de la preparación de casos de prueba y los detalles cuando ocurre cada uno de los casos mencionados.

6.5.1 Forma general de los casos de prueba

La idea básica para la preparación de casos de prueba a partir de la forma de verificar que se trató en la sección anterior, es como sigue:

1) Se toma la parte típica de la forma de verificar, es decir la parte exitosa de la función. Para esa parte se genera un grupo de casos de prueba, empleando algunos de los métodos descritos en el Capítulo 2 (dominios y valores a la frontera) o bien por otros que se tratarán en capítulos posteriores, como el de grafo causa-efecto o el de máquinas de estados.
2) Si existen caminos alternos exitosos, se hace lo mismo que en el apartado anterior.
3) Si existen caminos alternos que terminen en fracaso, se aplica el mismo procedimiento del primer apartado.
4) Considerando el enfoque de sistemas para prueba (ver Sección 6.1), se agregan casos de prueba para situaciones no previstas en los apartados anteriores, que incluirán:
    - Ausencia de algún elemento del sistema (biblioteca, clase, archivo)
    - Falta de disponibilidad de recursos necesarios (acceso a sitio remoto, acceso a base de datos, etc.) o errores en su direccionamiento *(path*)

- Equivocaciones del usuario (datos críticos vacíos, llaves duplicadas, tipos de datos erróneos, etc.)
- Situaciones del medio ambiente que afecten negativamente la operación del sistema (sobrecarga del equipo o la red, dispositivos en problemas (no listos, sin papel, etc.)

Muchas de las situaciones incluidas en los apartados 3 y 4 corresponden a problemas de validación y manejo de excepciones que deben tomarse en cuenta y deben producir salidas en forma de mensajes de aviso o toma de valores por omisión.

6.5.2 Entrada simultánea

Cuando las entradas son simultáneas, usualmente la generación de casos de prueba no tiene mayores problemas y no es necesario hacer más. Recuerde que los casos de prueba utilizan valores específicos de las variables de entrada o salida, lo que los distingue totalmente de las formas de verificar, que pueden ser más genéricas.

Como se trata de prueba de todo un sistema que rara vez trabajará en aislamiento, los casos de prueba que toman en cuenta el medio ambiente son muy importantes y a veces no basta representar los casos de prueba como se hizo en ejemplos de los capítulos anteriores. En muchos casos se requiere considerar las **condiciones** (mencionadas en el Capítulo 1) en que se da la prueba y en ocasiones debe considerarse la manera de lograr dichas condiciones, a través de un **procedimiento de prueba**.

Las condiciones describen el contexto deseado al realizar una prueba, como puede ser la presencia de ciertos datos en la base de datos, la ausencia de otras aplicaciones que compitan por los recursos o el estado específico de un objeto o una estructura de datos (por ejemplo una pila vacía o llena al tope). Las condiciones deben expresarse en forma de predicados específicos, sin indefiniciones.

Ejemplos de condiciones:

El usuario “Gonzalo” está registrado en la base de datos.

La base de datos está en el path “c:\unaaplicacion\bd”

La red está desconectada.

El producto “Jabón” no está registrado en la base de datos.

Las condiciones pueden ser de entrada y de salida, siendo más comunes las primeras. Puede haber varias condiciones para un solo caso de prueba.

Los casos de prueba con condiciones queda como el ejemplo de la Figura 6.5.

Condiciones entrada	Entradas	Salidas esperadas	Condiciones salida
El usuario “Gonzalo” está registrado en la	usuario=”Gonzalo” contra=”W3fY74”	“Bienvenido, Gonzalo”	El menú principal se activa

registrado en la BD con contraseña “W3fY74”			
El usuario “Gonzalo” está registrado en la BD con contraseña “W3fY47”	usuario=”Gonzalo” contra=”W3fY74”	“Error en contraseña”	El sistema termina ejecución.
La red está desconectada	url=”http:www/Elsitio.org”	“Sitio inaccesible”	
La red está desconectada	url=”http:www.Elsitio.org”	(se muestra la página www.Elsitio.org)	

Figura 6.5 Casos de prueba con condiciones

En algunos casos las condiciones resultan complejas de expresar o no resulta fácil verificar su cumplimiento. También puede suceder que las pruebas propuestas no sean inmediatamente identificables para el probador o para una persona que revise los casos de prueba. Esto puede suceder cuando no hay una manera directa de probar un resultado y deben aplicarse varios pasos. En estas dos situaciones se hace necesario un **procedimiento de prueba**. Éste consiste en una serie de pasos que deben realizarse para aplicar un grupo de casos de prueba.

Un procedimiento de prueba puede tener tres secciones:

a) preparativos para la prueba
b) instrucciones especiales para aplicar los casos de prueba
c) instrucciones para verificar los resultados y restaurar el estado original.

Los preparativos para la prueba incluyen la creación de elementos auxiliares (como archivos de datos y objetos), carga de registros de prueba a una base de datos y aplicación de otras funciones del sistema como antecedente para la función que se desea probar.

Las instrucciones para la prueba pueden incluir aviso de reiniciar el estado antes de cada caso de prueba o dejar que trabajen de manera sucesiva, sin reiniciar el estado.

La parte final puede incluir la verificación de resultados insertados en la base de datos o revisión de documentos impresos, así como retirar de la base de datos los registros usados para las pruebas.

Ejemplo.

1. Salve la base de datos.
2. Utilice una copia vacía de la base de datos.
3. Inserte los registros que se indican en el archivo “datosPrueba.txt”

4. Asegúrese de tener deshabilitadas las conexiones de red.
5. Aplique todos los casos de prueba, uno a uno, sin reiniciar la base de datos.
6. Verifique que en la base de datos haya sido eliminado el registro del producto “Café en polvo 250 gr.”
7. Verifique que en la base de datos la cantidad del producto “Azúcar cubos” indique 47.
8. Elimine la base de datos de prueba y restaure la original.

Estos procedimientos de prueba pueden realizarse manualmente, programarse en forma de script o dentro de un programa auxiliar para pruebas. El uso de scripts es muy común cuando se realizan pruebas automáticas.

6.5.3 Entradas en varias etapas

La sección anterior trató el caso en que se tiene un conjunto de entradas más o menos simultáneas. En ésta se tratará el segundo caso, cuando la entrada a un sistema se da en etapas que no resultan independientes.

Este caso puede probarse con el mismo procedimiento del caso sencillo, siempre y cuando se separe cada etapa en un caso de prueba y todos los pasos anteriores se incluyan en un procedimiento de prueba. Sin embargo éste enfoque resulta poco práctico, ya que se repetirán muchos pasos innecesariamente.

Ahora bien, la prueba de varias etapas de manera encadenada obliga a realizar un procedimiento de prueba donde se van detallando los datos de entrada en cada etapa y las salidas intermedias esperadas. Si cualquier paso de una salida intermedia difiere de lo esperado, se suspende la ejecución y se marca una falla.

Esta manera de probar usando un procedimiento que cubra las etapas requiere cuidados especiales en la selección de valores. A diferencia de un caso sencillo, donde se conocen los diversos dominios (o su equivalente, según el método), en este caso deben considerarse valores que obliguen a seguir un camino de ejecución determinado, que permita cubrir las diferentes etapas. Existirán valores que pasan la primera etapa, pero que no sigan adelante con la segunda y otros que pasen tres etapas y fracasen en la cuarta.

Por ejemplo, considere un sistema de inscripciones a diversos cursos, con diferentes precios, de acuerdo al siguiente procedimiento de prueba:

1. Se registra el nombre del alumno y otros datos del mismo, terminando al oprimir “Registra”; el sistema lo registra y muestra un número de control en un campo de la ventana, habilitando el botón “Arancel”. Si ya existe un alumno con el mismo nombre, rechaza el registro enviando un mensaje.
2. Cuando aparece el número de control se anota el tipo de curso y se oprime “Arancel”; el sistema muestra la cantidad a pagar y habilita el botón “Pago”.
3. El alumno indica si el pago es en efectivo o con tarjeta (y da el número) y oprime “Pago”; el sistema verifica y genera un recibo.

Si se desea probar la primera etapa por separado, se pueden dar nombres de alumnos no registrados y otros ya registrados. Sin embargo, si se desea probar todo el procedimiento hasta generar el recibo, sólo se pueden elegir nombres no repetidos; los otros producen salidas antes de generar el recibo. Igualmente, si no se elige un curso válido o ninguno le interesa al alumno, el procedimiento se quedará en la segunda etapa, sin concluir.

Así pues, cuando se prueba en este tipo de secuencias, debe tenerse cuidado con la selección de valores que lleguen hasta el final.

Recuerde que cada unidad ya fue probada por separado, así que aquí no se trata de volver a probar cada parte, sino el sistema como un todo.

6.6 Beneficios a partir de la preparación de casos de prueba

Hasta aquí se ha descrito como preparar listas de verificación de los requerimientos y la preparación de casos de prueba a partir de ellos, lo que llevará a asegurar el cumplimiento de todos los requerimientos.

Como se dijo al principio, la preparación de casos de prueba ocurre en las etapas tempranas del desarrollo del software, cuando todavía no se ha diseñado ni escrito el mismo, mientras que su ejecución ocurrirá en etapas más avanzadas.

Sin embargo, desde la preparación de los casos de prueba y los procedimientos asociados ya se obtienen ganancias para la calidad del software. Entre otras se tienen:

a) Las listas de verificación del cumplimiento de los requerimientos ayudan a realizar revisiones técnicas al software antes de iniciar su diseño, identificando los aspectos más débiles y los que pueden generar problemas; como deben estar completos, ayuda a asegurarse que no existan requerimientos que no se pueden comprobar. Por ejemplo, si hubiese un requerimiento “El sistema debe ser totalmente amigable”, sin una forma de verificar tal cosa, debe revisarse qué se entiende por “amigable” y, más aún, qué debe entenderse por “totalmente amigable”.

b) La preparación de los casos de prueba obliga a reflexionar acerca de los valores aceptables y los no aceptables, así como las situaciones no deseadas que podrían ocurrir, como los errores del usuario o la falla de la base de datos. El tener presentes estas situaciones permite diseñar una buena validación de las entradas y aseguramiento del ambiente, en vez de darlo por hecho. Por ejemplo, en Delphi es común hacer programas que abren automáticamente la base de datos, desde el inicio; si por alguna razón la base de datos tiene problemas, tales programas se cierran sin más trámite, dejando al usuario frustrado. De haberlo enido en cuenta, se habría dejado la base cerrada para abrirla más adelante, asegurándose de su buen funcionamiento.

c) Al preparar los casos de prueba se pueden identificar la funciones más complejas, con más casos posibles, lo que

hará que se ponga cuidado especial en su desarrollo, ya que muy probablemente serán las funciones que más defectos tendrán.

6.7 Ejecución de casos de prueba

Cuando se tiene lista una **integración** de un sistema, es decir, el sistema completo o una parte del mismo que ya satisface un cierto número de requerimientos y ha sido probado a nivel de unidades y de integración de partes, se puede pasar a ejecutar los casos de prueba que se prepararon con anterioridad. Esta prueba puede realizarse en varias etapas o maneras:

a) prueba de humo
b) pruebas de regresión
c) pruebas alfa realizadas por el equipo desarrollador en sus instalaciones
d) pruebas beta realizadas por el cliente en sus instalaciones, bajo supervisión del equipo desarrollador

Las pruebas de humo son de tipo muy tosco y preliminar; su nombre viene del campo de la electrónica antigua, donde se probaba un circuito nuevo conectándolo y observando si no se quemaba, es decir, si no lanzaba humo. El software no puede producir humo, pero si puede fallar estrepitosamente. Un sistema recién integrado se pone en ejecución sin preocuparse mucho de los casos de prueba, sólo para observar si al menos comienza a trabajar. Como se mencionó en alguna sección, existen programas que al abrirse de inmediato terminan sin avisar qué sucede (echan “humo”)[2].

A cada iteración de un proyecto se logra una integración del software, la cual debe probarse. Los defectos que se identifique se corrigen y se agregan nuevas funcionalidades antes de llegar a otra integración, repitiéndose el proceso.

En cada iteración habrá muchas pruebas que ya se cumplían en la iteración anterior, pero que deben volver a probarse, debido a los cambios, correcciones y modificaciones. Estas pruebas que se repiten se denominan pruebas de **regresión**, y definitivamente deben automatizarse, para lo cual se escriben los procedimientos de prueba a manera de scripts.

Cuando el sistema está completo, se realizan pruebas intensivas para ganar confianza en su funcionamiento. Inicialmente las pruebas se realizan en el ambiente del equipo desarrollador, aunque buscando su parecido con el ambiente del cliente. La duración de ésta etapa es uno de los asuntos más delicados que debe resolver el responsable del proyecto: si libera el software muy tarde, quizá pierda sus ganancias y aún al cliente; si libera antes de tiempo puede dejar demasiados defectos. Para auxiliar a quien debe tomar la decisión existen técnicas estadísticas que permiten estimar los defectos remanentes, aún no descubiertos, y decidir cuando ese número es inferior a un umbral dado (ver trabajo de [Lambuth, 2002]). Por ejemplo, ciertas compañías japonesas liberan el

[2] Ver descripción en la Wikipedia: http://wikipedia.org

software cuando se estiman 0.2 defectos por millar de líneas de código (ver trabajo de [Yamaura, 1996]).

Una vez concluidas las pruebas alfa, se pasa a pruebas en el ambiente del cliente, pero bajo supervisión del desarrollador, para validar el software contra las necesidades reales. Es probable que aparezcan nuevos defectos, ya que los usuarios reales no se comportan igual a los probadores de software.

6.8 Otras pruebas de sistema

Hasta ahora se han tratado pruebas de tipo funcional, es decir, que verifican los requerimientos funcionales o prestaciones deseadas del sistema. Sin embargo en muchos casos no son suficientes; apenas son una condición necesaria pero no suficiente.

Otras pruebas que pueden resultar necesarias son:

a) pruebas de rendimiento: donde se evalúa el número de transacciones por unidad de tiempo o el tiempo de respuesta a determinadas funcionalidades
b) pruebas de estrés: donde se somete el sistema a casos extremos, no esperados en situación normal, exagerando la cantidad de datos o la frecuencia de transacciones.

6.9 Referencias

Jacobson, Ivar, Booch, Grady, Rumbaugh, James, "*El Proceso Unificado de Desarrollo de Software"*, Addison Wesley, 2000

Lambuth, Mark, "*Bug fishing*" Embedded System Programming, 28/02/2002, localizable en Internet en http://embedded.com

Rosenberg, Doug: "*Use Case Driven Object Modeling with UML: a practical approach"*, Addison-Wesley, EUA, 1999.

Select "*Vista general de Select Perspective y Select Component Architect*", SELECT Business Solutions, 2003

Sumano, Ángeles "*Áncora: Análisis de requerimientos de software conducente al reuso de artefactos*", Universidad Veracruzana, 2006.

Yamaura, Tsuneo "*How to design practical test cases*", IEEE Software v 15, n 6, november/december 1998, pp 30-36.