



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

Índice Tema 12

Introducción.

1. Conceptos generales de la prueba.
 - 1.1. Definición de prueba.
 - 1.2. Principios de la prueba.
 - 1.3. Errores en el software. Tipos y consecuencias.
2. Tipos de pruebas.
3. Las fases del proceso de prueba.
4. Gestión del proceso de pruebas.
 - 4.1. Planificación de las pruebas. El plan de pruebas.
 - 4.2. El recurso humano en el proceso de pruebas.
5. El enfoque de las pruebas unitarias. Métodos de «caja blanca» y de «caja negra».
 - 5.1. El método de «caja blanca».
 - 5.1.1. Prueba de interfaz.
 - 5.1.2. Pruebas de estructuras de los datos locales.
 - 5.1.3. Prueba del camino básico.
 - 5.1.4. Prueba de bucles.
 - 5.2. El método de «caja negra».
 - 5.2.1. Particiones de equivalencia.
 - 5.2.2. Análisis de los valores límite.
 - 5.2.3. Valores típicos de error y valores imposibles.
 - 5.2.4. Ejemplo: aplicación de las técnicas de prueba tipo «caja negra».
 - 5.2.5. Otras técnicas de «caja negra».

- 6. Pruebas de integración.
 - 6.1. Integración incremental.
 - 6.1.1. Integración incremental descendente o de arriba hacia abajo.
 - 6.1.2. Integración incremental ascendente o de abajo hacia arriba.
 - 6.1.3. Comparación de las estrategias de integración incremental.
 - 6.2. Integración no incremental.
- 7. Pruebas del sistema y pruebas de implantación.
- 8. Prueba de aceptación.
- 9. Pruebas de regresión.
- 10. Pruebas estáticas. Revisiones.
 - 10.1. Revisión o «prueba» de requisitos.
 - 10.2. Revisión o «prueba» de diseño.
 - 10.3. Revisión o «prueba» de código.



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

TEMA 12

Pruebas. Planificación y documentación. Pruebas de caja negra. Pruebas de caja blanca. Utilización de datos de prueba. Pruebas de software, hardware, procedimientos y datos.

INTRODUCCIÓN.

Dentro de las actividades que comprende el proceso de desarrollo de un sistema de información un conjunto importante de ellas se orienta a garantizar la calidad del producto que se entrega, esto es la calidad del software. Estas actividades se aplican a lo largo de todo el ciclo de vida del sistema e incluyen, entre otras:

- El uso de una metodología de desarrollo y el uso de normas y estándares de desarrollo.
- La realización de revisiones formales e informales.
- La prueba del software.
- Y la gestión de la configuración, esto es, el control de los cambios realizados en el software y su documentación.

Así pues, las pruebas, objeto de este tema, deben estar orientadas a construir y entregar un producto de calidad, y en este sentido se puede afirmar que son el procedimiento de control de calidad más utilizado.

Asentada la necesidad de la prueba, sucede, sin embargo que ésta es una de las actividades peor entendidas. Esto se debe, entre otras causas, a que la naturaleza de lo que se prueba, el software, es algo intangible, inmaterial, es algo que no se ensambla en el sentido de producción convencional, sino que se desarrolla, su coste se centra más en los procesos de análisis y diseño que en el de fabricación, y no tiene piezas de recambio, es decir, cuando falla, falla todo el producto.

Este tema pretende abordar los aspectos relacionados con el proceso de prueba del software buscando un triple objetivo:

1. Comprender los conceptos generales de la prueba y entender que la prueba no es única y que la prueba total es imposible.
2. Comprender el proceso de pruebas, las fases de que consta y los aspectos relacionados con su gestión.
3. Comprender y distinguir los diferentes tipos de pruebas que se deben realizar al software y cuándo deben realizarse.

Para este cometido, comenzaremos definiendo qué es la prueba, los principios generales de toda prueba y, puesto que el objeto esencial de la prueba es descubrir errores, qué tipos de error se pueden presentar y cuáles son sus consecuencias.

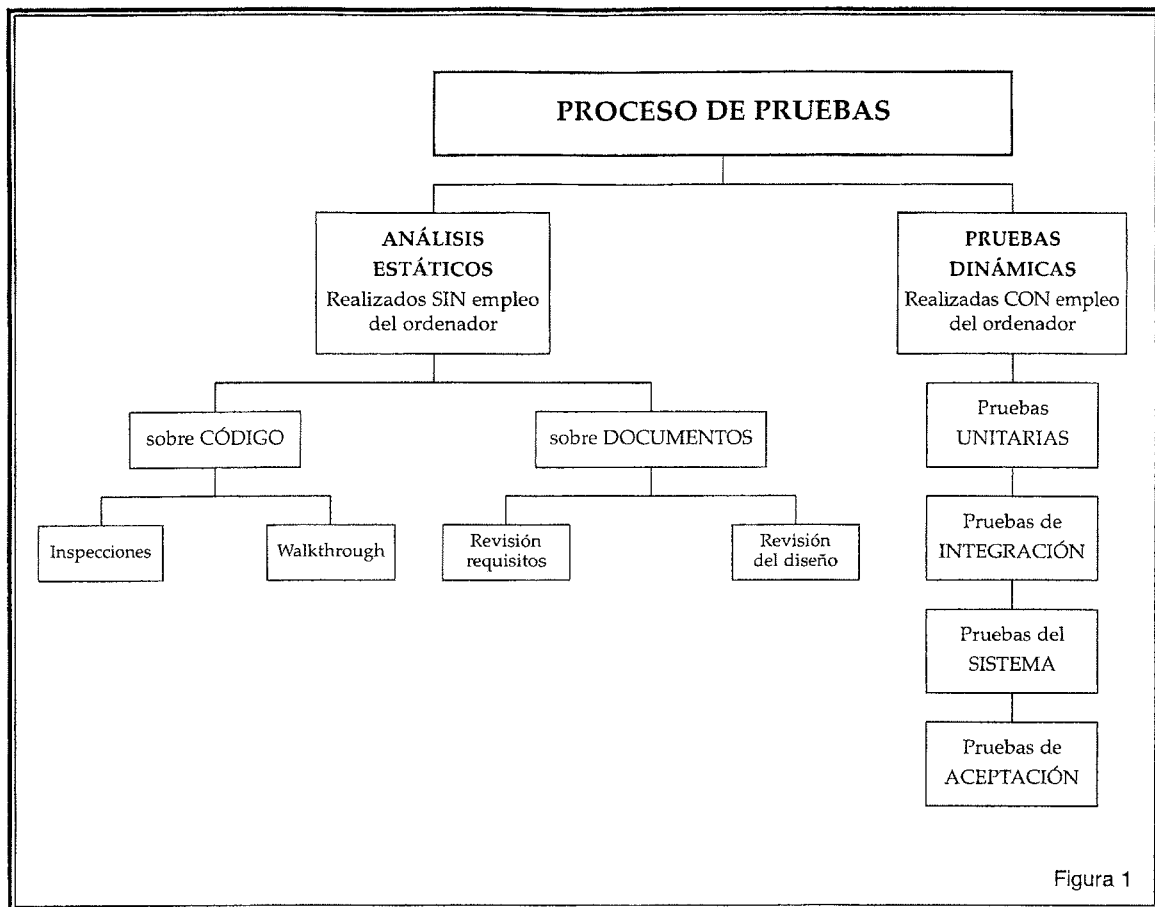
Seguidamente se presentarán los distintos tipos de pruebas que se aplican en el desarrollo de un sistema de información, distinguiendo, puesto que la prueba se dirige a garantizar la calidad del software, entre pruebas estáticas, también llamadas genéricamente revisiones, que no necesitan para su realización el uso del ordenador, y pruebas dinámicas, es decir, las que se aplican al producto software una vez construido y que requieren el empleo del ordenador para su ejecución.

Centrándonos exclusivamente en lo que hemos llamado «pruebas dinámicas», nos referiremos a continuación al proceso de pruebas. Se presentarán las fases de que consta el proceso y se abordarán los aspectos relativos a la gestión del mismo, aportando una metodología para la gestión del proceso de pruebas. En este punto se hará un especial hincapié en la planificación de las pruebas.

Finalmente se estudiará detalladamente cada tipo o nivel de prueba:

- Las pruebas unitarias, junto con los métodos de «caja blanca» y de «caja negra» para su realización.
- Las pruebas de integración, junto con las distintas estrategias de integración.
- Las pruebas del sistema.
- Las pruebas de implantación.
- Las pruebas de aceptación.
- Y las pruebas de regresión, asociadas al mantenimiento del sistema de información.

Por último se hará una breve referencia a ese tipo particular de pruebas, denominadas «revisiones», aplicables en cualquier momento del desarrollo, distinguiendo entre las pruebas que no utilizan el código (revisiones de requisitos y de diseño) y las que sí lo utilizan (inspecciones y walkthroughs).



1. CONCEPTOS GENERALES DE LA PRUEBA.

De manera general, puede decirse que el objetivo de la prueba es encontrar errores; sin embargo, esto no ha sido así a lo largo del tiempo.

- Durante los años 50, los conceptos de prueba y depuración no estaban separados. Los informáticos analizaban, diseñaban, escribían los programas y, a continuación, probaban y probaban, sucesivamente, hasta que consideraban que el programa estaba libre de errores. Sólo al final de la década, cuando empieza a crecer moderadamente el número de líneas de código, se separan ambos conceptos.
- En la década de los 60 se produce lo que se ha venido en llamar «crisis del software». El número de líneas de código aumenta espectacularmente y el impacto que los fallos del software producen en los recursos, tanto humanos como técnicos, así como en los costes, es dramático. Como consecuencia de esto, se empieza a prestar mayor atención a la prueba, dándosele mayor relevancia en el contexto del desarrollo del software.
- A partir de 1972, en que se define la prueba como conjunto de actividades orientadas a obtener la confianza de que un programa o sistema realiza las funciones que se supone que debe hacer, la prueba surge como elemento con peso propio dentro de la tecnología del software.
- Por último, a partir de los años 80-90, se asienta la necesidad de desarrollar la disciplina de pruebas, creando métodos para asegurar la calidad.

Un aspecto fundamental de la prueba es su coste. Aunque habría que matizar este término, pues una prueba es algo más que probar programas, se acepta, como término medio, que el coste de la prueba total de un producto software oscila entre el 40 y el 50 por 100 del coste total del producto. No obstante, sobre estas cifras medias, los costes varían enormemente en función de la criticidad del producto software, pudiendo existir variaciones desde el 15 por 100 del coste global, para sistemas poco críticos como, por ejemplo, un sistema de control de inventario para un pequeño comercio, hasta el 85 por 100 para sistemas altamente críticos, como, por ejemplo, un sistema de control de tráfico aéreo.

Otro aspecto importante es el que se refiere a la localización de las pruebas, esto es, ¿dónde se realiza el proceso de pruebas? La respuesta es sencilla. Puesto que las pruebas se centran en detectar los errores y corregirlos, hay que realizar pruebas durante el desarrollo del software y durante el mantenimiento, ya que en ambos procesos se producen errores.

La prueba durante el desarrollo del software puede afectar a cualquier fase, pensemos, por ejemplo, en el caso de que se está probando un programa y se detecta la omisión de un requisito. Esto conduce a la idea de que la prueba no es única en el sentido de que siempre se realiza sobre el código, sino que, en aras a garantizar la calidad del producto final, también se han de «probar» los requisitos, las especificaciones del diseño, etc. Dicho en otros términos, además de la prueba del código, se ha de revisar («probar») toda la documentación que se genere a lo largo del proceso de desarrollo del software.

Lo mismo cabe decir en cuanto al mantenimiento, ya sean los cambios en el software debidos a cambios en el entorno o a la inclusión de nuevas funcionalidades. En ambos casos habrá de probarse el código, pero también la documentación.

La parte de la Ingeniería del Software que aborda la prueba se denomina «Técnicas de verificación y validación».

Se entiende por verificación el conjunto de actividades que aseguran que el software implementa correctamente una función específica, y por validación el proceso de evaluación del software al final del proceso de desarrollo, para asegurar una correspondencia del producto con los requisitos. Es decir, la validación se refiere al conjunto de pruebas que aseguran que el software construido se ajusta a los requisitos del cliente.

La verificación y la validación son una parte importante del conjunto de actividades que se realizan para construir software de calidad, por tanto sus objetivos son valorar y mejorar la calidad de los productos intermedios del proceso de desarrollo y de los productos a entregar al usuario del proyecto, una vez desarrollados.

1.1. DEFINICIÓN DE PRUEBA.

Antes de dar una definición de «prueba del software», vamos a mostrar algunas definiciones incorrectas de la misma, con el fin de evitar malos entendidos.

- La prueba es el proceso de demostrar que no hay errores en el software. La definición es incorrecta, primero, porque es imposible realizar todas las pruebas para demostrar que no hay errores, y segundo porque la definición implica un cambio de objetivo en la prueba (descubrir errores), lo que conduce a seleccionar casos de prueba que demuestren que no hay errores.

- La prueba es el proceso de establecer la confianza de que un programa realiza lo que se supone que debe hacer. Definición incorrecta puesto que aunque un programa haga lo que se supone que debe hacer, puede tener errores.

Existen muchas definiciones correctas de «prueba del software». Algunas de ellas son:

- La prueba es el proceso de ejecución de un programa con la intención de descubrir errores (Glenford Myers).
- La prueba es cualquier actividad dirigida a evaluar la capacidad de un programa y determinar que alcanza los resultados requeridos.

Puesto que la prueba sirve para detectar y prevenir errores, diremos que una prueba tiene éxito o es satisfactoria cuando detecta algún error no encontrado hasta ese momento.

Siguiendo la metodología Métrica v.3, las pruebas son prácticas a realizar en diversos momentos de la vida del sistema de información al objeto de verificar:

- El correcto funcionamiento de los componentes del sistema.
- El correcto ensamblaje entre los distintos componentes.
- El funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de los sistemas de información con los que se comunica.
- El funcionamiento correcto del sistema integrado de hardware y software en el entorno de operación.
- Que el sistema cumple con el funcionamiento esperado y permite al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento.
- Que los cambios sobre un componente del sistema de información, no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.

1.2. PRINCIPIOS DE LA PRUEBA.

Conocida la definición de prueba, se enuncian a continuación algunos principios que deben ser considerados como una guía cuando se realiza el proceso de prueba:

- La prueba es el proceso de ejecutar un programa con la intención de descubrir errores, por lo que un buen caso de prueba es aquel que tiene una alta probabilidad de descubrir un error no encontrado hasta el momento.
- La prueba completa no es posible.
- Los casos de prueba se deben escribir para condiciones de entrada válidas e inválidas y esperadas e inesperadas; y se deben definir las salidas o resultados esperados de los casos de prueba. Asimismo, se deben conservar los casos de prueba.

- La prueba de software se realiza tanto para ver si hace lo que se supone que debe hacer, como para ver si hace lo que se supone que no debe hacer.
- Un programador o un equipo, en su caso, no debe probar sus propios programas.
- No se debe planificar el esfuerzo de la prueba bajo la creencia de que no se encontrarán errores.
- La probabilidad de la existencia de más errores en una parte del software es proporcional al número de errores ya encontrados en dicha parte.

1.3. ERRORES EN EL SOFTWARE. TIPOS Y CONSECUENCIAS.

Los objetivos de la prueba son descubrir y prevenir errores, pero, ahora bien, ¿son iguales todos los errores y tienen la misma importancia?, ¿todos los errores afectan de igual forma? Evidentemente, no. Los errores o defectos en el software no son iguales, no tienen la misma importancia, la cual se mide en su coste monetario, y tienen diferentes consecuencias.

Los tipos de errores más comunes que se detectan en el software son:

- Errores lógicos. Son la manifestación de asunciones erróneas, y su distribución depende de la lógica de la aplicación.
- Errores de codificación. Son los cometidos al crear la estructura de interconexión de los módulos o interfaces.
- Errores tipográficos y de mecanografía. Son los que se producen en la escritura de las sentencias que constituyen el código. Estos errores se distribuyen al azar.

La importancia de los errores depende de los siguientes factores:

- Frecuencia, esto es, el número de veces que ocurre el error.
- Coste de corrección, coste en tiempo y dinero de corregir el error, una vez encontrado.
- Coste de instalación, que se refiere al número de instalaciones o puestos que soportan o ejecutan el software.
- Coste de consecuencias, que se refiere a una medida de las consecuencias ocasionadas por el error al usuario (no es lo mismo un error que desajuste el inventario de una pequeña tienda, que un error en un sistema de control de tráfico aéreo).

Una medida de la importancia del error puede venir dada por la expresión:

$$\begin{aligned} \text{IMPORTANCIA (en dinero)} &= \\ &= \text{Frecuencia} \times (\text{coste de corrección} + \text{coste de instalación} + \text{coste de consecuencias}) \end{aligned}$$

Así como no todos los errores tienen la misma importancia, tampoco afectan al usuario o al sistema de la misma manera, es decir, no tienen las mismas consecuencias. En los diversos textos de Ingeniería del Software existen múltiples rangos para calibrar las consecuencias. Uno de ellos, muy usual, es el que cataloga la consecuencia de los errores en una escala o rango del 1 al 10, de la siguiente manera:

1. Leve. El efecto del error es meramente estético. Por ejemplo, un encolumnado incorrecto en un listado.
2. Moderado. El efecto puede afectar al rendimiento del sistema. Por ejemplo, salidas redundantes.
3. Incómodo. Deshumaniza el comportamiento del sistema. Por ejemplo, nombres truncados.
4. Trastorno. No realiza de forma correcta el tratamiento de las transacciones.
5. Serio. Pierde la pista de la transacción.
6. Muy serio. El sistema realiza una transacción errónea.
7. Extremo. El problema no se limita a unos pocos usuarios o transacciones tipo, sino que la frecuencia y arbitrariedad de los errores es la norma de comportamiento del sistema.
8. Intolerable. Provoca la decisión de parar el sistema. Por ejemplo, corrupción de la base de datos.
9. Catastrófico. La parada del sistema se produce porque éste falla. Por ejemplo, falla el núcleo del sistema operativo.
10. Infeccioso. Se corrompen otros sistemas o el entorno, incluso sin provocar parada del sistema.

2. TIPOS DE PRUEBAS.

Puesto que el objetivo de la prueba es descubrir errores en aras a garantizar la calidad del software, además de probar el producto terminado es necesario, también, hacer pruebas durante el proceso de desarrollo del software. En este sentido decíamos que la prueba no es única; esto es, además de las pruebas que podríamos llamar «de código», es necesario probar, o mejor, revisar, la documentación que se va generando durante el proceso de desarrollo, fundamentalmente la de requisitos.

Esto nos lleva a afirmar que existe más de un tipo de pruebas, las cuales pueden clasificarse atendiendo a varios criterios.

Atendiendo a la necesidad de ejecutar el objeto probado se distingue entre:

1. Análisis estáticos o Pruebas estáticas, que analizan el objeto sin necesidad de ejecutarlo. Genéricamente se conocen por el nombre de «revisiones» y el objeto probado es la documentación asociada a las distintas fases del desarrollo. Dentro de las revisiones se distingue entre:
 - Inspecciones, en las que los participantes van leyendo paso a paso el documento, comprobando el cumplimiento de los criterios de la lista de comprobación.

- Walkthroughs, en las que se demuestra la funcionalidad del objeto revisado mediante la simulación de su funcionamiento con casos de prueba y ejemplos.
2. Pruebas dinámicas, que son aquellas que requieren la ejecución del objeto que se está probando. Es decir, son pruebas en las que para su realización se requiere el empleo del ordenador.

Refiriéndonos a las pruebas dinámicas, que son nuestro objeto de estudio, se pueden establecer los siguientes criterios de clasificación.

Atendiendo a su ámbito de realización, las pruebas pueden ser:

- Pruebas unitarias, en las que se prueban de forma individual todos los componentes del sistema que se desarrollen al objeto de comprobar su correcto funcionamiento.
- Pruebas de integración, en las que se prueba la integración entre los componentes del sistema para demostrar que se pueden encajar correctamente.
- Pruebas del sistema, en las que se prueba globalmente todo el sistema. Dentro de esta categoría tienen nombre propio las Pruebas de Aceptación, que son las que realiza el usuario para comprobar si el sistema cumple los requisitos expresados por él.

Si consideramos las pruebas a realizar en la etapa de mantenimiento del software, se habla de Pruebas de Regresión, que son aquellas a realizar después de la modificación de cualquier parte del sistema. Este tipo de pruebas incluye todas las anteriores, esto es, las unitarias, las de integración, las del sistema y las de aceptación.

Atendiendo a la estrategia o enfoque a seguir en el diseño de la prueba se distinguen dos tipos:

- Pruebas de Caja Blanca, que permiten examinar la estructura interna de los programas.
- Pruebas de Caja Negra, donde los casos de prueba se diseñan considerando exclusivamente las entradas y salidas del sistema, sin preocuparse por la estructura interna del mismo.

Finalmente, desde la metodología empleada en la realización de las pruebas, algunos autores distinguen entre:

- Pruebas informales de desarrollo, que empiezan tras la codificación y la depuración y son llevadas a cabo por el propio equipo de desarrollo.
- Pruebas formales de validación, que deben ser realizadas por un grupo independiente, después de que el equipo de desarrollo haya finalizado las pruebas informales.

3. LAS FASES DEL PROCESO DE PRUEBA.

El proceso de prueba de un sistema software es inverso al proceso de creación. Esto es, mientras que el desarrollo de un sistema software transcurre desde las especificaciones (lo abstracto) hasta el programa (lo concreto u operativo), la prueba avanza desde el programa hasta el sistema en su totalidad. En otras palabras, la prueba tiene una estructura de abajo hacia arriba o bottom-up que contrasta con la estructura top-down del desarrollo.

La mayoría de los autores de Ingeniería del Software considera que las fases de que consta todo proceso de pruebas son, de manera secuencial, las siguientes:

1. Prueba de unidad. Es la primera prueba a realizar en todo proceso de pruebas y corresponde a la prueba de cada módulo del programa, por lo que también se la denomina «prueba modular». Centra sus actividades en ejercitar la lógica del módulo y los distintos aspectos de la especificación del mismo, y la realiza el propio programador en su entorno de trabajo.

Un error encontrado en esta fase provocaría un retroceso a la fase de diseño detallado e implicaría revisar la lógica y/o las especificaciones del módulo.

2. Prueba de integración. Una vez realizada y superada la prueba de unidad, se integran los diferentes módulos con el fin de probar sus interfaces. Esta prueba tiene en cuenta la agrupación de los módulos y el flujo de información en las interfaces entre dos módulos.

Un error localizado en esta prueba llevaría a revisar la estructura del programa y/o del sistema realizada en el diseño de alto nivel.

3. Prueba de validación. Una vez ensamblado el software se comprueba que cumple las especificaciones validando el mismo frente a la especificación de requisitos. El objeto de esta prueba es comprobar los posibles desajustes respecto a los requisitos del software. Normalmente la prueba de validación se suele integrar en la prueba del sistema.

Como es natural, un error en esta fase llevaría a revisar la especificación de requisitos.

4. Prueba del sistema. Esta prueba centra sus comprobaciones en el cumplimiento de los objetivos indicados para el sistema del que el software forma parte. Consiste en que el software, una vez validado, se relacione con los restantes elementos que conforman el sistema global (hardware, otro software existente, bases de datos, etc.) y que el conjunto total cumpla con los requisitos impuestos en el análisis de sistemas.

Si se localizara un error en esta fase, se tendrían que revisar los objetivos fijados para el software desde el punto de vista de ser un componente de un sistema mayor en el que está inmerso.

5. Prueba de aceptación. Es la última fase del proceso de pruebas y constituye el último paso antes de la entrega formal del software al cliente. La prueba de aceptación consiste en comprobar que el producto se ajusta a los criterios fijados por el usuario y en la aceptación por éste (cliente) del software desarrollado. Esta prueba se realiza, normalmente, en entornos de usuario.

Un error encontrado en esta prueba llevaría a revisar los requisitos del usuario.

4. GESTIÓN DEL PROCESO DE PRUEBAS.

En el proceso de desarrollo de un sistema de información, además de las actividades propias de desarrollo, mediante las que se crean los productos que componen el sistema, es necesario contemplar también un conjunto de actividades de gestión, que se refieren a la ejecución, administración y supervisión del proceso e incluyen: la planificación, la asignación de recursos, la organización del equipo de trabajo, la estimación de costes, etc. La fase de pruebas, como cualquier otra fase del desarrollo, tiene, además de la vertiente técnica una vertiente de gestión, inseparable de aquella. El éxito del proceso de pruebas viene dado por un desarrollo de éste bien gestionado siguiendo una metodología.

En cualquier fase del proceso de pruebas está claro que la prueba a realizar se corresponde con un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente, esto es, bajo una metodología. En este sentido, se denomina estrategia de pruebas a la organización de la prueba a realizar en una serie de pasos o actividades bien planificadas, al objeto de conseguir una correcta construcción del sistema.

A grandes rasgos, los pasos o actividades que debe incorporar cualquier estrategia de pruebas son:

1. La planificación de la prueba a realizar.
2. El diseño de los casos de prueba.
3. La ejecución de la prueba.
4. La agrupación y evaluación de los datos resultantes de las actividades de prueba.

Sin embargo, a fin de concretar más la gestión del proceso de pruebas, se indican en el siguiente cuadro las fases, actividades, objetivos de la actividad y documentos de salida que debe comprender cualquier metodología para la gestión del proceso de pruebas.

METODOLOGÍA PARA LA GESTIÓN DEL PROCESO DE PRUEBAS

FASES	ACTIVIDADES	OBJETIVOS DE LA ACTIVIDAD	DOCUMENTO DE SALIDA
PLANIFICACIÓN	PLANIFICACIÓN DE LAS PRUEBAS	<ul style="list-style-type: none"> • Objetivos del proceso de pruebas. • Relación de objetos a probar. • Alcance de las pruebas. • Métodos a utilizar. • Recursos a emplear. • Cronograma de las pruebas. • Productos a generar. • Reparto de responsabilidades. 	PLAN DE PRUEBAS
PARA CADA PRUEBA CONCRETA			
PREPARACIÓN	DISEÑO DE LA PRUEBA	<ul style="list-style-type: none"> • Descripción de la prueba. • Métodos de prueba a aplicar. • Objetos a probar. • Criterios para determinar el cierre de la prueba. 	DOCUMENTO DE DISEÑO DE LA PRUEBA

.../...

PREPARACIÓN	ESPECIFICACIÓN DE LA PRUEBA	<ul style="list-style-type: none"> • Objetos a probar. • Entradas propuestas. • Salidas esperadas. 	ESPECIFICACIÓN DE LOS CASOS DE PRUEBA
REALIZACIÓN	PLANIFICACIÓN DEL PROCEDIMIENTO DE PRUEBA	<ul style="list-style-type: none"> • Pasos a realizar para la ejecución de los casos de prueba. • Requisitos para la ejecución de la prueba. • Condiciones de terminación de la prueba. 	ESPECIFICACIÓN DEL PROCEDIMIENTO DE PRUEBA
	EJECUCIÓN DE LA PRUEBA	<ul style="list-style-type: none"> • Aplicar cada caso de prueba especificado y registrar los problemas detectados. 	INFORME DE LOS CASOS DE PRUEBA
	ANÁLISIS Y EVALUACIÓN DE LA PRUEBA	<ul style="list-style-type: none"> • Comparar los resultados de la prueba con las salidas esperadas y los objetivos de la misma. • En su caso, cierre de la prueba o revisión de la misma con vistas a su repetición. 	INFORME DE LA PRUEBA

4.1. PLANIFICACIÓN DE LAS PRUEBAS: EL PLAN DE PRUEBAS.

De la misma forma que el proceso de desarrollo de un sistema de información debe estar planificado, así también debe planificarse el proceso de pruebas. La planificación de las pruebas, reflejada en el documento denominado «Plan de Pruebas», es la primera y la más importante actividad para gestionar adecuadamente el proceso de pruebas y sobre la que se apoyará toda la realización del mismo.

La prueba debe planificarse, y un buen Plan de Pruebas es aquel que descubre el máximo número de errores en el menor tiempo y al menor coste posible. Ahora bien, diseñar un plan de pruebas que obtenga un software sin errores es imposible. La prueba demuestra que existen errores en el software, pero al no poder demostrar lo contrario, en la planificación de las pruebas hay que considerar tanto los aspectos técnicos como los económicos.

Los aspectos que debe contener el Plan de Pruebas, resultado de la planificación de las mismas, son los siguientes:

- **Objetivos.** Esto es, definir los objetivos de cada fase de las pruebas.
- **Criterios de terminación.** Existen varios criterios para determinar cuándo deben cerrarse las pruebas. Los más comunes son:
 - Terminar la prueba cuando el tiempo establecido para la misma haya concluido. Método no muy útil ya que el tiempo establecido puede ser muy corto, en cuyo caso el objeto de la prueba, que es detectar errores, queda insatisfecho, o puede ser excesivo, en cuyo caso se estarían dilapidando recursos y aumentando innecesariamente el coste de la prueba.

– Terminar cuando todos los casos de prueba se ejecuten sin detectar errores. Método tampoco óptimo porque predispone a generar casos de prueba sencillos, para que el programa los pase sin errores y finalizar la prueba cuanto antes.

– Otros métodos más complicados de aplicar, pero más efectivos son:

1. Estimar el número total de errores del programa.
2. Estimar el porcentaje de errores que pueden encontrarse fácilmente.
3. Estimar qué fracción de errores se origina en procesos particulares de diseño.

El problema de utilizar métodos de estimación es que ésta sea demasiado alta y el programa tenga menos errores de los estimados, en cuyo caso nunca se terminarían las pruebas porque nunca se llegaría al límite establecido. El problema se solucionaría poniendo también un límite temporal.

- **Cronología.** Es la planificación temporal de la prueba. Se deben fijar los tiempos necesarios para cada actividad de prueba: diseño de la prueba, determinación de los casos de prueba, ejecución de la prueba, etc.
- **Responsabilidades.** Es la asignación de recursos humanos a la prueba. Para cada fase y actividad de prueba se han de especificar las personas que intervienen y su cometido.
- **Bibliotecas de casos de prueba y normas.** La prueba debe realizarse según una metodología de trabajo, por tanto, debe crearse una sistemática de identificación, escritura y almacenamiento de casos de prueba.
- **Herramientas.** Es la asignación de recursos software a la prueba. El plan de pruebas debe establecer las herramientas automáticas de prueba que se van a utilizar.
- **Tiempo de máquina.** Es la asignación de recursos hardware a la prueba. Debe determinarse el tiempo de computación que se necesita en cada fase del proceso de prueba, así como la necesidad de configuraciones especiales de equipo o de algún período concreto.
- **Integración.** Se deberá describir el plan de integración del sistema.
- **Métodos de seguimiento.** Se refiere al control del proceso de pruebas. El plan deberá especificar los métodos de seguimiento de las actividades del proceso de pruebas.
- **Depuración.** Por último, el plan debe definir un mecanismo para informar sobre los errores detectados, para seguir el proceso de correcciones y para incorporar éstas al sistema.

La planificación de las pruebas y su concreción en el Plan de Pruebas se va realizando a lo largo de las distintas fases que comprende el desarrollo del sistema.

- En la fase de Análisis del Sistema se inicia la definición del Plan de Pruebas especificando y justificando los tipos o niveles de pruebas a realizar y el marco general de cada nivel de prueba (participantes implicados, planificación temporal, criterios de verificación y aceptación de las pruebas, productos a entregar como resultado de la ejecución de las pruebas, etc.). Asimismo, se definen los requisitos del entorno de pruebas (hardware, software, herramientas, librerías, etc.) y también se definen las pruebas de aceptación.

- En la fase de Diseño del Sistema se especifica detalladamente el Plan de Pruebas para cada uno de los tipos o niveles de prueba establecidos (unitarias, de integración, del sistema, de implantación y de aceptación). En este sentido, se detalla y se completa la especificación del entorno de pruebas y se diseña detalladamente cada tipo de prueba (casos de prueba, procedimientos de prueba, criterios de aceptación, análisis y evaluación de los resultados, etc.). Por último, en esta fase se procede a revisar el Plan de Pruebas, completando o detallando algún aspecto, por ejemplo, perfiles de los participantes o cronograma de las pruebas, si fuera necesario.
- Siguiendo la planificación especificada en el Plan de Pruebas, en la fase de Construcción del Sistema se realizan las pruebas unitarias, de integración y del sistema, y en la fase de Implantación y Aceptación del Sistema, las pruebas de implantación y de aceptación. En ambos casos se revisan las especificaciones del Plan de Pruebas y se completan, si fuera necesario, con casos de prueba adicionales.

4.2. EL RECURSO HUMANO EN EL PROCESO DE PRUEBAS.

Una de las claves del éxito del proceso de pruebas es contar con los recursos adecuados que asegurarán la efectividad del proceso y la calidad del producto que se está probando. Entre ellos, el recurso humano es fundamental.

Aunque son muchas las personas que participan en el proceso de prueba, cada una con su perfil característico y su misión concreta, nos referiremos particularmente a dos, por el peso específico que tienen dentro del proceso. Son, en concreto, los especialistas de prueba y el director de pruebas.

Los especialistas de prueba tienen como función principal asegurar que se lleva a cabo un proceso de pruebas efectivo. Esta figura es necesaria por el conocimiento especializado de éstos y por su experiencia. Los especialistas de pruebas deben tener un perfil que responda a una buena capacidad de control, espíritu crítico, comprensión y consideración. Las actividades principales que llevan a cabo los especialistas de prueba son:

- Asegurar que la prueba se lleva a cabo.
- Asegurar que la prueba se documenta.
- Asegurar que las técnicas de prueba están definidas y se utilizan.

La función del director de pruebas es la de coordinar y dar soporte de los distintos grupos que conforman el desarrollo de un proyecto en lo referido al proceso de pruebas. Las responsabilidades del director de pruebas abarcan:

- La organización de las pruebas, definiendo las políticas de prueba.
- La planificación de las pruebas.
- La preparación de las pruebas, indicando herramientas de soporte y ayuda, desarrollando la base de datos de prueba y preparando las especificaciones de prueba.
- El control de las pruebas, asegurando que se siguen los planes previstos y realizando revisiones de los procedimientos de prueba.

- El seguimiento de la prueba, esto es, la revisión de la efectividad del proceso y su coste.
- La documentación de la prueba, es decir, registrar los resultados y determinar formalmente las pruebas que se han llevado a cabo.

5. EL ENFOQUE DE LAS PRUEBAS UNITARIAS. MÉTODOS DE «CAJA BLANCA» Y DE «CAJA NEGRA».

Tal como se puso de manifiesto al señalar las fases de que consta el proceso de prueba, las pruebas unitarias o pruebas de unidad constituyen la prueba inicial de un sistema y las demás deben apoyarse sobre ellas. Su objetivo es verificar la funcionalidad y estructura de cada componente (módulo) individualmente, una vez que ha sido codificado. Dicho de otra forma, las pruebas unitarias pretenden comprobar que el módulo, entendido éste como una unidad funcional de un programa completamente independiente, está correctamente codificado.

Probar un módulo o programa significa en la práctica la ejecución del mismo, a priori un número indeterminado de veces, para comprobar que ante unos datos de entrada (casos de prueba) se producen unos resultados esperados y acordes con la especificación del programa.

Dentro de la estrategia de pruebas existen dos enfoques o métodos diferentes para la construcción de casos de prueba, ambos no excluyentes sino complementarios. Su diferencia estriba en la visión que van a poseer de la estructura interna del programa y, en general, no aseguran que el programa esté completamente probado. Estos métodos son los denominados «caja blanca» y «caja negra», a los que nos referiremos seguidamente.

Aunque no es lo óptimo, normalmente las pruebas unitarias son realizadas por el programador que codificó el programa, en vez de, como sería deseable, ser realizadas por un programador diferente. Asimismo, la mayor parte de las veces estas pruebas son realizadas de un modo informal, es decir, sin una planificación rigurosa ni un registro de los casos de prueba utilizados y de los resultados obtenidos, por lo que con esta filosofía, las pruebas se completarían cuando el programador considere que a su juicio el programa está suficientemente probado y consiguientemente, el software tiene un nivel de confianza aceptable.

Una de las claves del éxito de las pruebas unitarias o de unidad consiste en elegir adecuadamente los módulos. Para ello, aunque no hay una definición concreta y completamente aceptada acerca del significado de este término, se puede considerar que un módulo es un bloque básico de un programa, con un tamaño que puede oscilar entre las 50 y 300 líneas y que realiza, al menos idealmente, una función simple e independiente.

5.1. EL MÉTODO DE «CAJA BLANCA».

El enfoque de la estrategia de pruebas conocido por el nombre de método de «caja blanca» o, también, «conducido por la lógica» o «logic driven», se centra en probar el comportamiento interno y la estructura del programa, examinando la lógica interna del mismo y sin considerar los aspectos de rendimiento.

El objetivo de este enfoque es ejecutar, al menos una vez, todas las sentencias y ejecutar todas las condiciones tanto en su vertiente verdadera como falsa. Teniendo en cuenta que la única información de

entrada con que se cuenta es el diseño del programa y el código fuente. A tal fin, las pruebas del tipo «caja blanca» deben responder a dos cuestiones básicas: ¿es correcta la lógica del programa? y ¿está completa la lógica del programa?, es decir, ¿está todo correctamente especificado sin faltar ninguna función?

Una vez realizadas las pruebas del tipo «caja blanca» podría pensarse que el programa está completamente probado; sin embargo, esto no es así. El enfoque de la prueba según el método de «caja blanca» no puede considerarse una prueba exhaustiva de los caminos lógicos del programa por dos razones:

1. Porque excepto que se trate de programas triviales, el número de caminos de control, es decir, de posibles alternativas a ejecutar, es demasiado grande.
2. Porque aunque se pudiesen comprobar todos los caminos lógicos, dado que en este enfoque de pruebas la información de entrada sólo es el diseño del programa y el código fuente, todavía podrían existir errores por varios motivos:
 - Porque podría ocurrir que el código fuente fuera incorrecto respecto a las especificaciones de usuario. En este caso, el programa haría bien una cosa equivocada.
 - Porque podría ocurrir que el código fuente fuera incompleto respecto a las especificaciones de usuario. En este caso, el programa todo lo que hace lo haría bien, pero no haría las cosas que tendría que hacer.
 - Porque podría ocurrir un error que sólo se produjera en intervalos concretos de dos valores de dos variables, por lo que pasar por ese punto del programa con otros valores no resolvería nada.

Así pues, como la exhaustividad es imposible e incluso, aunque fuera posible, sería insuficiente, hay que diseñar una serie de estrategias que optimicen los resultados. A este respecto, las técnicas específicas más usuales que siguen el método de «caja blanca» son:

- Prueba de interfaz.
- Pruebas de estructuras de los datos locales.
- Prueba del camino básico: cobertura de sentencias y cobertura de condiciones.
- Prueba de bucles.

5.1.1. Prueba de interfaz.

Este tipo de prueba debe ser la primera a realizar. Se basa en analizar el flujo de datos que pasa a través de la interfaz del módulo, tanto externa como interna, para asegurar que la información fluye de forma adecuada tanto hacia el interior como hacia el exterior del módulo que se está probando.

5.1.2. Pruebas de estructuras de los datos locales.

Las estructuras de datos locales son una fuente potencial de errores. Estas pruebas, que están asociadas a las pruebas de interfaz ya comentadas, tienen como objetivo asegurar la integridad de los datos durante todos los pasos de la ejecución del módulo. Se distinguen cuatro categorías de pruebas en los datos:

- Referencias de datos. Se refieren a los accesos que se realizan a los mismos.
- Declaración de datos. Su propósito es comprobar que todas las definiciones de los datos locales son correctas.
- Cálculo. Intenta localizar errores derivados del uso de variables.
- Comparación. Intenta localizar errores en las comparaciones realizadas en instrucciones tipo If, While, etc.

5.1.3. Prueba del camino básico.

La prueba del camino básico es una técnica de «caja blanca» que permite al diseñador de casos de prueba obtener una medida de la complejidad lógica del diseño procedimental de la estructura de control, o lo que es lo mismo, del código y utilizar esa medida como guía para definir un conjunto básico de caminos de ejecución.

De la técnica del camino básico se derivan dos pruebas complementarias y no excluyentes:

- Prueba de cobertura de sentencias. Consiste en generar casos de prueba que permitan probar todas y cada una de las sentencias de un módulo una vez. Esta prueba es necesaria pero no es suficiente.
- Prueba de cobertura de condiciones. Consiste en diseñar juegos de prueba que consideren todos los valores posibles de cada una de las condiciones. Esta prueba también es necesaria pero no es suficiente ya que no garantiza que todos los caminos sean cubiertos, por lo que debe ser complementada con la anterior.

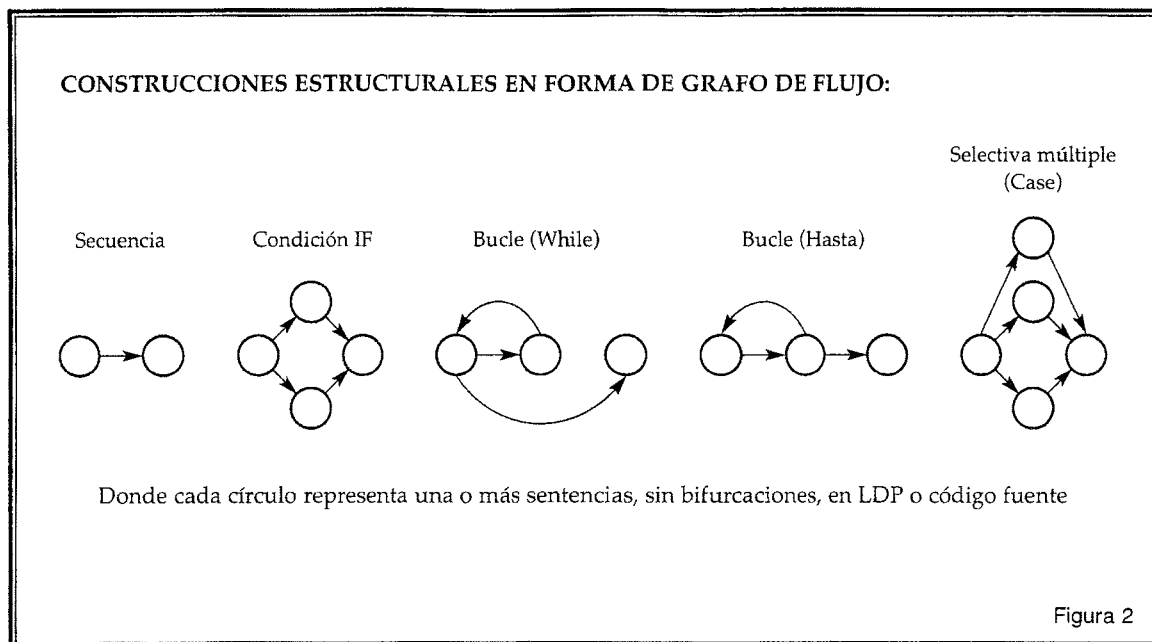
El probar todos los caminos es un objetivo deseable, pero en muchas ocasiones no se puede conseguir, bien porque el número de caminos sea muy grande, lo que sucede cuando hay bucles o repeticiones, o bien porque alguno de los caminos sea imposible de ejecutar debido a alguna condición.

Por consiguiente, la prueba del camino básico se orienta a cubrir la ejecución de cada una de las sentencias, cada una de las decisiones y cada una de las condiciones en las decisiones, tanto en su vertiente verdadera como falsa.

Esta técnica conocida como prueba del camino básico utiliza una convención de representación denominada «grafos de flujo» para la determinación de caminos y para ello se apoya en el concepto de «complejidad ciclomática» propuesto por Tom McCabe.

La notación de «grafo de flujo», propuesta por McCabe, se utiliza para simplificar el desarrollo del conjunto básico de caminos de ejecución. Su objetivo es ejemplificar el flujo de control del módulo que se está probando y para ello utiliza tres elementos:

- Nodos o tareas de procesamiento. Representan cero, una o varias sentencias procedimentales. Cada nodo comprende como máximo una sentencia de decisión (bifurcación).
- Aristas, flujo de control o conexiones. Unen dos nodos, incluso aunque el nodo no represente ninguna sentencia procedimental.
- Regiones. Son las áreas delimitadas por las aristas y nodos. Cuando se contabilizan las regiones debe incluirse el área externa como una región más.



La complejidad ciclomática es una métrica del software que aporta una medición cuantitativa de la complejidad lógica de un programa, aunque cuando se usa en el contexto de la prueba del camino básico su resultado define el número de caminos independientes del módulo.

Un camino independiente es cualquier camino del módulo que introduce, por lo menos, un conjunto de sentencias de procesamiento o una condición no probada, es decir, incluye una arista del grafo de flujo que no ha sido recorrida anteriormente durante la prueba.

El valor obtenido por la complejidad ciclomática proporciona un límite superior en el número de caminos a recorrer y, consiguientemente en el número de pruebas a realizar. La complejidad ciclomática se puede calcular de varias maneras, por ejemplo:

- Complejidad ciclomática = Número de regiones
- Complejidad ciclomática = Número de nodos predicados + 1, entendiendo por nodo predicado aquel nodo independiente que se genera para cada una de las condiciones simples de una sentencia de decisión compuesta (véase figura)

Pero la forma más habitual es calcularla según la expresión del Número Ciclométrico de McCabe, que se define como:

$$NC = \text{Número de aristas} - \text{Número de nodos} + 2$$

Está demostrado que cuanto mayor es el número ciclométrico, mayor es la complejidad del programa. McCabe sugiere como criterio de cobertura de caminos lógicos el añadir al criterio de cobertura de las condiciones el requisito de que, para un programa cuya complejidad ciclométrica sea «NC», al menos se deban seleccionar un número de caminos diferentes igual a «NC».

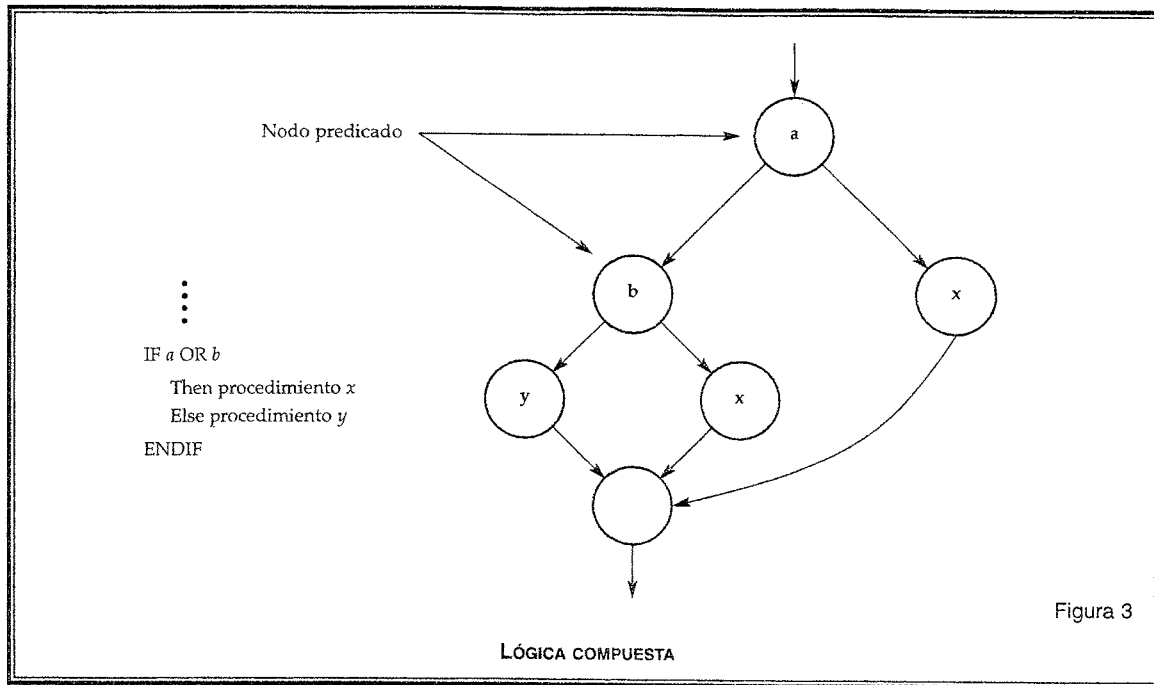


Figura 3

El siguiente ejemplo pretende aclarar los conceptos expuestos. Para ello se partirá del organigrama o diagrama de flujo de un programa y se obtendrá su representación como grafo de flujo a efectos de calcular la complejidad ciclomática.

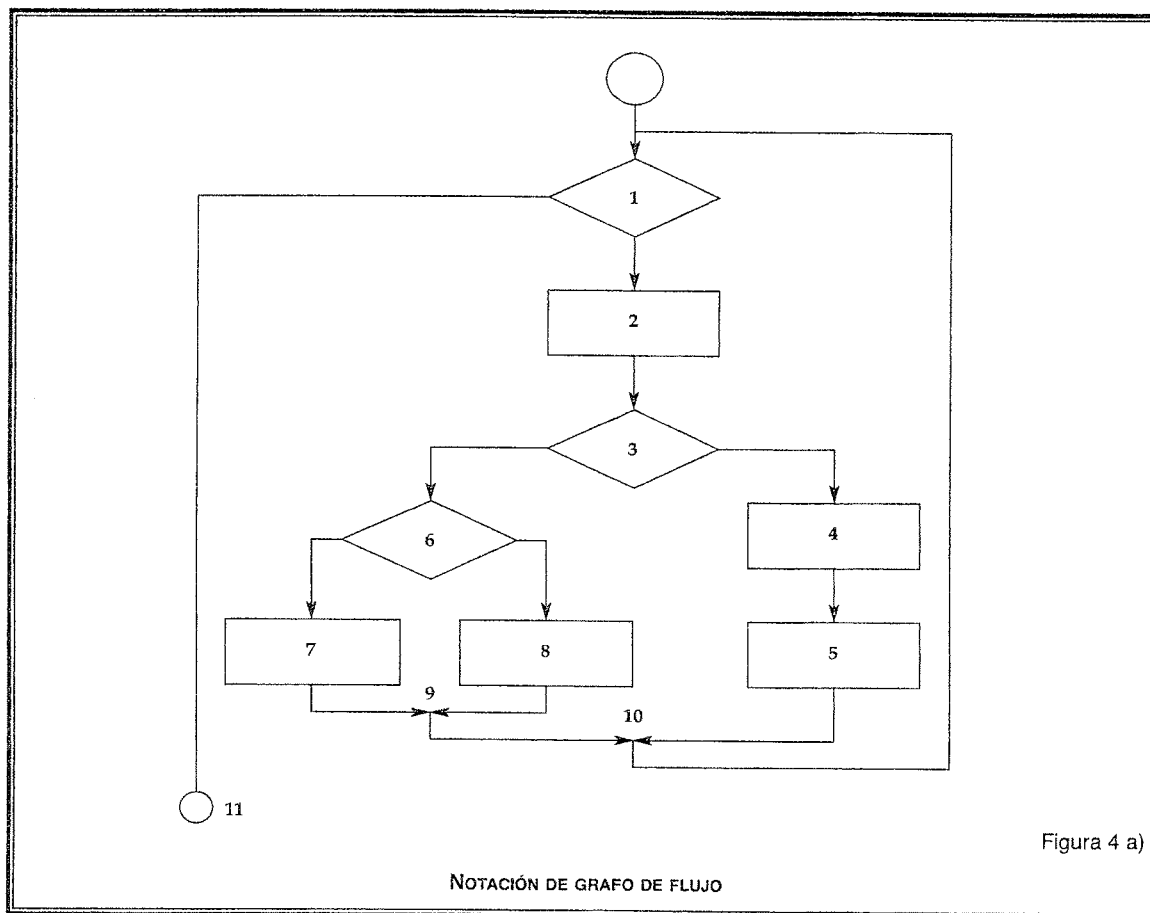
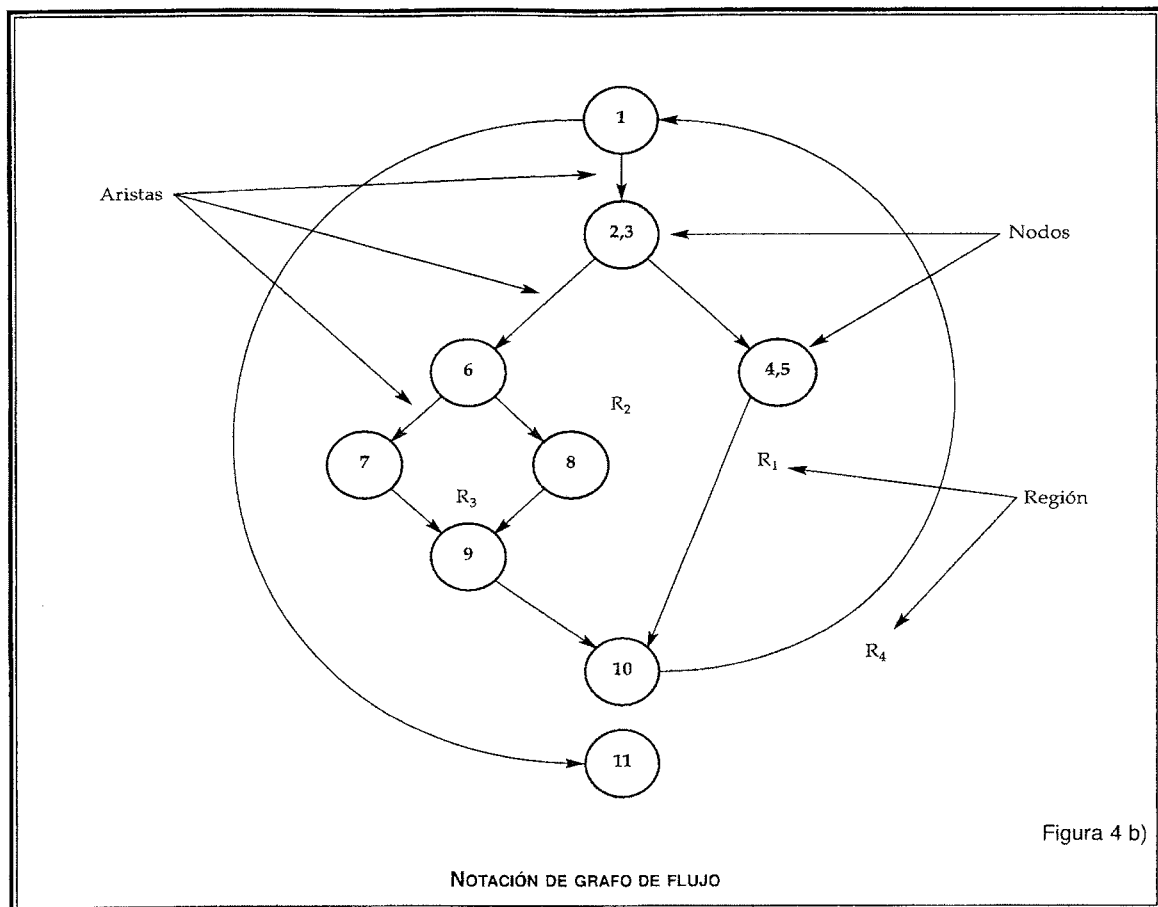


Figura 4 a)



Del grafo de flujo se obtiene que el número de nodos es $n = 9$ y el número de aristas es $e = 11$. El número ciclomático de McCabe es $NC = e - n + 2 = 11 - 9 + 2 = 4$, que coincide con el número de regiones.

5.1.4. Prueba de bucles.

La prueba de bucles, también denominada prueba de condiciones límite, es otra técnica de «caja blanca» que se centra en comprobar la validez de las construcciones de bucles. La mayoría de los algoritmos implementados en software tienen a los bucles como parte principal integrante en su estructura. Determinar la validez de las construcciones de los bucles es fundamental para garantizar que es correcto el módulo que se está probando.

Existen cuatro tipos de bucles en relación a las pruebas a realizar en ellos. Son:

1. Bucles simples. Siendo N el número máximo de iteraciones permitidas en el bucle, el conjunto de pruebas a aplicar será:

- Saltar el bucle, esto es, no ejecutarlo.
- Ejecutarlo una vez.
- Ejecutarlo M veces, siendo $M < N$.
- Ejecutarlo $N - 1$ veces, luego N veces y luego $N + 1$ veces.

2. Bucles anidados. Un bucle está anidado cuando está en el interior de otro bucle. Esto produce que el número de posibles pruebas aumente geométricamente con el nivel de anidamiento, por lo que para reducir el número de pruebas a realizar se puede utilizar la siguiente estrategia:
 - Comenzar a probar el bucle más interior como si fuera un bucle simple, estableciendo los demás bucles en sus valores mínimos.
 - Progresar hacia fuera llevando a cabo pruebas para el siguiente bucle, manteniendo los bucles externos en sus valores mínimos y los internos en sus valores típicos.
 - Continuar hasta que se prueben todos los bucles.
3. Bucles concatenados. Al tratarse de bucles encadenados existen dos posibilidades de prueba: si los bucles son independientes, se aplicará el enfoque de bucle simple a cada uno de ellos; por el contrario, si se trata de bucles dependientes, se aplicará el enfoque de bucles anidados.
4. Bucles no estructurados. En este caso, ante la complejidad que puede representar la comprensión del flujo de control, es más práctico rediseñar el módulo a probar de forma que se codifique mediante bucles estructurados.

En definitiva, la técnica de pruebas de «caja blanca», que tiene la ventaja de que posibilita verificar que los criterios seleccionados han sido satisfechos, plantea, sin embargo una serie de inconvenientes, siendo de destacar los siguientes:

- No proporciona muchos medios para cuantificar los posibles errores que pueden existir en los caminos no probados.
- Nunca se puede demostrar la corrección absoluta de los programas, aunque se pudieran probar todos los caminos.
- La dependencia del diseño de los casos de prueba de la estructura interna del programa hace que este diseño varíe mucho dependiendo del diseñador del programa.

5.2. EL MÉTODO DE «CAJA NEGRA».

El enfoque de la estrategia de pruebas conocido por el nombre de método de «caja negra» o, también, «conducido por los datos» («data driven») o «conducido por la entrada/salida» («input-output driven»), no considera el detalle procedimental de los programas y se centra en buscar situaciones donde el programa no se ajusta a su especificación, utilizando ésta como entrada para derivar los casos de prueba.

Si por las especificaciones funcionales se sabe lo que tiene que hacer un módulo, es más sencillo comprobarlo que desmenuzarlo y examinarlo internamente en todas las circunstancias posibles. Por ello, las pruebas de «caja negra» son el enfoque más simple de prueba del software, y en ella diseñaremos los casos de prueba a partir de las especificaciones funcionales.

Con las pruebas del tipo «caja negra» se intenta probar:

- Las funciones realizadas por el sistema.
- El cumplimiento de los objetivos del sistema.
- Las reacciones del sistema ante estímulos exteriores.
- Las transacciones manejadas por el sistema.

En este tipo de pruebas los casos de prueba consisten en conjuntos de datos de entrada que deberán generar una salida acorde con la especificación. La atención se centra, pues, en los datos de entrada y salida ignorando intencionadamente el conocimiento del código del programa.

Si con esta técnica se quisieran encontrar todos los errores del programa, habría que recurrir a probar todas las posibles combinaciones de casos de entrada, lo que supondría generar todas las posibles combinaciones de valores para todas las posibles variables de entrada, y ello, en la realidad, es imposible. De esta imposibilidad se pueden extraer dos conclusiones: que mediante el método de la «caja negra» no es posible asegurar que un programa esté libre de errores; y que el problema de la prueba de los programas tiene un componente esencialmente económico.

Dado que no se pueden probar todos los casos posibles, el método de «caja negra» contempla una serie de técnicas encaminadas a simplificar los casos de prueba. Éstas son:

- Las particiones de equivalencia.
- El análisis de valores límite.
- Los valores típicos de error y los valores imposibles.
- Los grafos causa-efecto.
- Las pruebas de comparación.

5.2.1. Particiones de equivalencia.

La finalidad de esta técnica es encontrar juegos de prueba que representen de la mejor forma posible el conjunto potencialmente infinito, y en todos los casos inmanejable, de entradas posibles al sistema.

La técnica de la partición de equivalencia consiste en dividir la entrada de un programa en clases de datos de los que se pueden derivar los casos de prueba, y la estrategia consiste en definir casos de prueba que descubran clases de errores, reduciéndose así el número de casos de prueba que hay que desarrollar.

La partición de equivalencia se basa en el concepto de «clase de equivalencia», entendiendo por tal un conjunto de estados válidos o inválidos para unas determinadas condiciones de entrada. Las clases de equivalencia se identifican tomando una condición de entrada y dividiéndola en dos o más grupos; y se pueden definir de acuerdo con las siguientes directrices:

- a) Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos inválidas.

- b) Si una condición de entrada requiere un valor específico un rango, se define una clase de equivalencia válida y dos inválidas.
- c) Si una condición de entrada especifica un elemento de un conjunto, se define una clase de equivalencia válida y una inválida.
- d) Si una condición de entrada es lógica, se define una clase de equivalencia válida y una inválida.

En definitiva, la aplicación de la técnica de las particiones de equivalencia como técnica de prueba de «caja negra» supone:

1. Identificar las clases de equivalencia, siguiendo las directrices apuntadas anteriormente.
2. Definir los casos de prueba.

Más adelante se muestra un ejemplo de esta técnica.

5.2.2. Análisis de los valores límite.

El análisis de valores límite es una técnica de diseño de casos de prueba que complementa a la partición de equivalencia y se justifica en la constatación de que para una condición de entrada que admite un rango de valores, es más fácil que existan errores en los límites que en el centro. Por tanto, la diferencia entre esta técnica y la partición de equivalencia estriba en que en el análisis de valores límite no se selecciona un elemento representativo de la clase de equivalencia, sino que se seleccionan uno o más elementos de manera que los límites de cada clase de equivalencia son objeto de prueba. Otra diferencia es que con esta técnica también se derivan casos de prueba para las condiciones de salida.

Al igual que en el caso anterior, la técnica de análisis de valores límite no asegura la prueba completa, ya que es imposible probar exhaustivamente todos los conjuntos de datos de entrada tanto en su vertiente válida como inválida. Sin embargo, la ventaja que presenta esta técnica es que maximiza el número de errores encontrados con el menor número de casos de prueba posibles, lo que rentabiliza la inversión efectuada en la prueba.

5.2.3. Valores típicos de error y valores imposibles.

Un buen complemento de las dos técnicas fundamentales de pruebas tipo «caja negra» (particiones de equivalencia y análisis de valores límite) consiste en incluir en los casos de prueba ciertos valores de los datos de entrada susceptibles de causar problemas, esto es, valores típicos de error, y valores especificados como no posibles, es decir, valores imposibles.

La determinación de los valores típicos de error se realiza en función de la naturaleza y funcionalidad del programa a probar, por lo que depende en buena medida de la experiencia del diseñador de la prueba. Un criterio corriente que se suele seguir es considerar que en un intervalo de valores de una variable de entrada es más probable que haya error en los extremos del intervalo que en los valores centrales.

Asimismo, dentro de las especificaciones del sistema o del programa a probar puede haber valores de datos especificados como no posibles. El hecho de probar estos valores imposibles se debe a que dichos valores podrían haber sido generados internamente por el sistema o el programa provocando un mal funcionamiento del mismo. La prueba de valores imposibles debe realizarse siempre que dichos valores puedan ser detectados y el programa pueda manejarlos adecuadamente sin provocar errores irreparables.

5.2.4. Ejemplo: aplicación de las técnicas de prueba tipo «caja negra».

Sea una aplicación bancaria en la que el usuario puede llamar al banco a través de un ordenador y realizar una serie de operaciones. Los datos de entrada que tiene que enviar el usuario son los siguientes:

- Código de banco. En blanco o un número de tres dígitos, siendo el primero mayor que 1.
- Código de sucursal. Un número de cuatro dígitos, el primero mayor que 0.
- Número de cuenta. Un número de cinco dígitos.
- Clave personal. Un valor alfanumérico de cinco posiciones.
- Orden. Puede ser una de las dos cadenas siguientes:
 - Talonario, en cuyo caso el usuario recibe un talonario de cheques.
 - Movimientos, en cuyo caso el usuario recibe los movimientos del mes en curso.

O puede estar en blanco, en cuyo caso el usuario recibe los dos documentos anteriores.

Con estos datos vamos a determinar los casos de prueba siguiendo la técnica de las particiones de equivalencia y del análisis de valores límite.

Solución.

Lo primero es analizar cada dato de entrada para determinar las clases de equivalencia y los valores límite.

- Código de banco. La condición de entrada es lógica, ya que puede, o no, existir dicho código. Por tanto, siguiendo las directrices de definición de clases de equivalencia, hay que definir una válida y otra inválida.

Como valores límite del rango de valores se define el inferior (199) y el superior (1000).

- Código de sucursal. La condición de entrada es un rango, por tanto se definen una clase de equivalencia válida y dos inválidas.

Como valores límite se define el inferior (999) y el superior (10000).

- Número de cuenta. La condición de entrada es un valor, por tanto se definen una clase de equivalencia válida y una inválida.

Como valores límite se define el inferior (cuatro dígitos) y el superior (seis dígitos).

- Clave personal. La condición de entrada es un valor, por tanto se definen una clase de equivalencia válida y una inválida.

Como valores límite se define el inferior (cuatro caracteres) y el superior (seis caracteres).

DATOS DE ENTRADA	CONDICIONES DE ENTRADA (CE)	CLASES DE EQUIVALENCIA (CLE)	VALORES LÍMITE
Código de banco	Lógica (puede existir o no) Si existe, rango, desde 200 hasta 999	1 válida y 1 inválida 1 válida y 2 inválidas	Inferior Superior
Código de sucursal	Rango, desde 1000 hasta 9999	1 válida y 2 inválidas	Inferior Superior
Número de cuenta	Valor, número de 5 dígitos	1 válida y 1 inválida	Inferior Superior
Clave personal	Valor, cadena de 5 caracteres	1 válida y 1 inválida	Inferior Superior
Orden	Lógica (puede existir o no) Si existe, conjunto «talonario» «movimientos»	1 válida y 1 inválida	

Una vez determinadas las condiciones de entrada, clases de equivalencia y valores límite, el paso siguiente es derivar los casos de prueba a partir de lo anterior.

Anotando «b» como blancos, «C» como correcto e «I» como incorrecto, en la siguiente tabla se muestran algunos casos de prueba y el resultado esperado.

Derivación de los casos de prueba

CASO PRUEBA	Cód. BANCO	Cód. SUCURSAL	NÚMERO DE CUENTA	CLAVE PERSONAL	ORDEN	RESULTADO
1	«C» bbb	«C» 1000	«C» 00000	«C»	«C» bb...b	«C» CORRECTO
2	«I» 20A	«C»	«C»	«C»	«C»	«I» CB no numér.
3	«I» 199	«C»	«C»	«C»	«C»	«I» CB fuera rango
4	«I» 1000	«C»	«C»	«C»	«C»	«I» CB fuera rango
5	«C» 200	«I» 100A	«C»	«C»	«C»	«I» CS no numér.
6	«C» 201	«I» 999	«C»	«C»	«C»	«I» CS fuera límite
7	«C»	«I» 10000	«C»	«C»	«C»	«I» CS fuera límite
8	«C» 999	«C» 9999	«I» 0000A	«C»	«C»	«I» NC no numér.
9	«C» 998	«C» 9998	«I» 1234	«C»	«C»	«I» NC fuera límite
10	«C»	«C» 1001	«I» 123456	«C»	«C»	«I» NC fuera límite
11	«C»	«C»	«C»	«I» ABCD	«C»	«I» CP long. inadec.
12	«C»	«C»	«C»	«I» ABCDEF	«C»	«I» CP long. inadec.
13	«C»	«C»	«C»	«C»	«C» Talonario	«C» CORRECTO
14	«C»	«C»	«C»	«C»	«I» Talonarios	«I» Orden incorrec.
15	«C»	«C»	«C»	«C»	«C» Movimientos	«C» CORRECTO
16	«C»	«C»	«C»	«C»	«I» Movimiento	«I» Orden incorrec.

5.2.5. Otras técnicas de «caja negra».

Además de las técnicas de «caja negra» mencionadas, existen otras que, aunque menos usuales, añaden valor y complementan a aquéllas. Entre ellas destacaremos los grafos causa-efecto y las pruebas de comparación.

Los grafos causa-efecto es una técnica de diseño de casos de prueba que proporciona una representación de las condiciones lógicas y sus correspondientes acciones. Un grafo causa-efecto es un lenguaje formal que construye un circuito digital con una notación lógica simplificada, al que se traslada una especificación realizada en lenguaje natural. Permite analizar una serie de condiciones de entrada proporcionando una herramienta de selección sistemática de casos de prueba.

Los pasos que utiliza esta técnica son los siguientes:

1. Descomponer el programa en módulos de pequeño tamaño.
2. Identificar para cada módulo: las causas, es decir, las condiciones de entrada, y los efectos, esto es, las condiciones de salida.
3. Desarrollar el grafo causa-efecto.
4. Convertir el grafo en tablas de decisión.
5. Convertir las tablas de decisión en casos de prueba.

La prueba de comparación o «prueba mano a mano» se realiza en aquellos casos en que por ser crítica una aplicación, se han desarrollado dos versiones distintas del software. Estas versiones han de ser probadas independientemente contra las especificaciones para asegurar que los resultados no sólo son correctos sino que son idénticos en cualquier circunstancia.

6. PRUEBAS DE INTEGRACIÓN.

Una vez realizadas las pruebas unitarias, la siguiente fase en el proceso de prueba son las pruebas de integración. Estas pruebas son necesarias porque la prueba de unidad no asegura que cuando se tomen todos los módulos como un conjunto, el sistema completo funcione. En otras palabras, aunque cada módulo funcione correctamente por separado, una vez integrados pueden contener fallos.

El objetivo de la prueba de integración es verificar la existencia de errores una vez ensamblado el software, comprobando para ello los siguientes aspectos:

- La integridad de interfaz, esto es, se comprueban las interfaces internas y externas.
- El contenido de la información, para descubrir errores asociados a las estructuras de datos globales o locales.
- La validez funcional, con la que se pretenden descubrir errores de función en los módulos ya integrados.
- El rendimiento, que se utiliza para verificar los límites de rendimiento establecidos durante la especificación y diseño del software.

Alguno de estos aspectos es paralelo a los indicados en las pruebas unitarias, con lo que un conjunto de módulos integrados, ya probados individualmente, se podría considerar como una nueva unidad y aplicarles un subconjunto de las pruebas unitarias, en concreto, la prueba de interfaz y la de estructura de los datos. Ahora bien, por razones de tipo económico, como la de obtener el máximo rendimiento de la prueba con el mínimo esfuerzo, la cantidad y exigencia de las pruebas de integración debe restringirse, y una estrategia adecuada es identificar los módulos críticos para el funcionamiento del sistema y centrar los esfuerzos en ellos.

La forma de ensamblar o integrar los módulos introduce una complejidad adicional a estas pruebas, por lo que para paliar, en la medida de lo posible, los problemas de integración, se han definido las siguientes estrategias:

- Integración incremental, donde los módulos se integran poco a poco y a medida que se integran se van probando. Dentro de esta estrategia caben dos posibilidades:
 - La integración incremental descendente o de arriba hacia abajo (top-down).
 - La integración incremental ascendente o de abajo hacia arriba (bottom-up).
- Integración no incremental, donde todos los módulos se juntan de una vez y se prueban todos al mismo tiempo.

A continuación se estudia cada estrategia de integración.

6.1. INTEGRACIÓN INCREMENTAL.

Mediante este enfoque el programa se construye y prueba en pequeños segmentos. De esta forma, los errores a localizar en cada segmento son menores que los que podría haber en el programa tomado como conjunto y, además, están más localizados espacialmente.

Cuando se prueba cada módulo individualmente es necesario crear módulos auxiliares que simulen las acciones de los módulos invocados por el que se está probando, y módulos «conductores» para establecer las precondiciones necesarias, llamar al módulo objeto de la prueba y examinar los resultados de la prueba. Con la integración incremental se agrega cada módulo o componente individual al conjunto de módulos existentes y el conjunto resultante es lo que se prueba. Esto reduce la necesidad de crear módulos conductores y permite, además, examinar más detalladamente las interfaces, ya que al incorporar un nuevo componente a un grupo ya probado, lo más probable es que los problemas que surjan se deban precisamente al componente que se incorpora o a las interfaces entre él y los otros componentes.

Dentro de la estrategia de integración incremental se distinguen dos aproximaciones: la integración descendente y la integración ascendente.

6.1.1. Integración incremental descendente o de arriba hacia abajo.

En esta aproximación, también llamada integración «top-down», la construcción y prueba comienza con los módulos de mayor nivel e incorpora los módulos subordinados progresivamente, es decir, la integración y la prueba discurre de arriba hacia abajo.

Puesto que los módulos de menor nivel están subordinados al módulo que se prueba, se necesitará un software específicamente creado para sustituir estos módulos de más bajo nivel, es decir, se necesitan módulos auxiliares, también llamados módulos de resguardo.

Aunque no hay un orden óptimo de incorporación de los módulos, sin embargo existen dos formas de incorporación principales:

- En profundidad, integrando todos los módulos de un camino principal de la estructura y moviéndose por ésta de forma vertical.
- En anchura, incorporando los módulos directamente subordinados en cada nivel y moviéndose por la estructura de forma horizontal.

A la hora de elegir una u otra opción se debe tener en cuenta los siguientes criterios:

- El módulo que se está integrando debe tener todos los módulos que lo llaman previamente integrados.
- Los módulos críticos, si existen, deben incorporarse cuanto antes.
- El orden de integración debe incorporar cuanto antes los módulos de entrada/salida.

Las etapas en que se debe realizar la prueba de integración descendente son:

- Probar el módulo de mayor nivel utilizando módulos auxiliares para sustituir a los subordinados.
- Sustituir los módulos subordinados por módulos reales, probando cada vez que se incorpora un módulo nuevo.
- Al terminar cada prueba, sustituir otro módulo auxiliar por el módulo real, siguiendo el modo elegido, ya sea en profundidad o en anchura.

6.1.2. Integración incremental ascendente o de abajo hacia arriba.

En esta aproximación, también llamada integración «bottom-up», la construcción y prueba comienza desde los niveles más bajos de la estructura del programa, y se integra desde los módulos de menor nivel hacia el programa en su conjunto, es decir, de abajo hacia arriba.

Debido a que los módulos de menor nivel no son funcionalmente independientes, se necesitará un software específicamente creado para llamar a los módulos que se deben probar, es decir, se necesitan módulos conductores.

Las etapas en que se debe realizar la prueba de integración ascendente son:

- Combinar los módulos de bajo nivel en grupos que realizan una subfunción específica.
- Escribir un módulo conductor para la entrada de casos de prueba al grupo de módulos que se está probando y para la salida de resultados.

- Probar cada grupo.
- Eliminar los módulos conductores y combinar grupos ya probados progresando sucesivamente hacia niveles más altos.

6.1.3. Comparación de las estrategias de integración incremental.

Una vez vistas las estrategias de integración descendente y ascendente, la siguiente tabla muestra las ventajas e inconvenientes de cada una de ellas.

INTEGRACIÓN DESCENDENTE (TOP-DOWN)

VENTAJAS	INCONVENIENTES
<ul style="list-style-type: none"> • Ventajosa si aparecen grandes fallos en los niveles superiores del programa. • Una vez incorporadas las funciones de entrada/salida, es fácil representar los casos de prueba. • Permite ver antes una estructura previa del programa, que facilita el poder hacer demostraciones 	<ul style="list-style-type: none"> • Se requieren módulos auxiliares o de resguardo, que no son fáciles de realizar, con el consiguiente trabajo de desarrollo. • Antes de incorporar las funciones de E/S es difícil representar los casos de prueba. • Las entradas para las pruebas pueden ser difíciles de crear, puesto que a menudo se carece de los módulos de nivel inferior. • La observación de la salida de la prueba es más difícil. • Puede dar lugar a pensar que prueba y diseño pueden superponerse. • Induce a retrasar la terminación de la prueba de algunos módulos.

INTEGRACIÓN ASCENDENTE (BOTTOM-UP)

VENTAJAS	INCONVENIENTES
<ul style="list-style-type: none"> • Ventajosa si aparecen grandes fallos en la parte inferior del programa. • Las entradas para las pruebas son más fáciles de crear, puesto que los módulos inferiores tienen funciones más detalladas. • Es más fácil la observación de los resultados de la prueba, ya que los módulos inferiores suelen ser los de operación. 	<ul style="list-style-type: none"> • Se requieren módulos conductores, que deben escribirse especialmente, y que no son sencillos de codificar. • El programa, como entidad, no existe hasta incorporar el último módulo.

Intentando buscar las ventajas de las estrategias de integración presentadas, suele aplicarse a veces un enfoque mixto o estrategia combinada (ascendente-descendente), también llamada prueba «sandwich». La forma de proceder es la siguiente:

- Se aplica la integración descendente en los niveles superiores de la jerarquía de módulos.
- Paralelamente, se aplica la integración ascendente en los niveles inferiores de la jerarquía de módulos.
- La integración termina cuando ambas estrategias se encuentran en un punto intermedio de la jerarquía de módulos.

6.2. INTEGRACIÓN NO INCREMENTAL.

Existe otra posibilidad de prueba de integración denominada «integración no incremental» o «big bang». Esta prueba consiste en la integración de todos los módulos para construir el programa completo, una vez que han pasado la prueba de unidad. La prueba se ejecuta, por tanto, sobre el programa como conjunto.

Debido a que se prueba todo el programa, suelen localizarse un gran número de errores, pero es muy complejo aislar sus causas al tener que buscarlas por todo el programa.

La ventaja de esta estrategia es que consume menos tiempo del recurso máquina y no necesita módulos auxiliares ni conductores. Por el contrario, la eficiencia en la prueba se reduce drásticamente, incidiendo en la fiabilidad del producto software.

7. PRUEBAS DEL SISTEMA Y PRUEBAS DE IMPLANTACIÓN.

Una vez que se han probado los componentes individuales (prueba de unidad) y se han integrado (prueba de integración), la siguiente fase en el proceso de prueba corresponde a la del sistema global, es decir, al sistema integrado de hardware y software al objeto de verificar que cumple los requisitos especificados.

La inmensa mayoría de los textos de Ingeniería del software identifican lo anterior como pruebas del sistema, sin embargo, la metodología métrica versión 3, más sutil, distingue entre pruebas del sistema y pruebas de implantación.

Las pruebas del sistema se llevan a cabo en el entorno de desarrollo y tienen como objetivo comprobar la integración del sistema de información globalmente, verificando el correcto funcionamiento de las interfaces entre los distintos subsistemas que lo componen y entre el sistema y el resto de sistemas de información con los que se comunica. A estos efectos, los aspectos que se deben comprobar, y que constituyen los distintos tipos de pruebas incluidas en las del sistema, son los siguientes:

- Que el sistema realiza correctamente todas las funciones que se han detallado en las especificaciones dadas por el usuario (pruebas funcionales).
- El funcionamiento adecuado, tanto de la interface hombre-máquina (interface de usuario e interface de operador), como de las interfaces entre los componentes del sistema ya sea a través de dispositivos remotos o de dispositivos locales (pruebas de comunicaciones).

- Que los tiempos de respuesta están dentro de los límites establecidos en las especificaciones del sistema (pruebas de rendimiento).
- Que el sistema se adapta a las necesidades de los usuarios, tanto a su modo habitual de trabajo, como a la facilidad de introducción de datos y de obtención de resultados (pruebas de facilidad de uso).
- El funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos (pruebas de volumen).
- El funcionamiento del sistema en el umbral límite de los recursos, sometiénolo a cargas masivas (pruebas de sobrecarga).
- La recuperación del sistema ante fallos de equipo físico o lógico sin pérdidas indebidas de datos (pruebas de disponibilidad de datos).
- Los procedimientos de operación, incluyendo la planificación y control de trabajos y el arranque y re arranque del sistema (pruebas de operación).
- Las interacciones del sistema con otros sistemas del mismo entorno (pruebas de entorno).
- Los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos (pruebas de seguridad).

Las técnicas utilizadas en la prueba del sistema son técnicas de «caja negra», diseñadas para validar los requisitos sin fijarse en el funcionamiento interno de los programas, así como técnicas de «caja blanca» modificadas.

Normalmente se utilizan herramientas automatizadas para simplificar el diseño de los casos de prueba, tales como generadores de datos de prueba, comparadores, simuladores, etc. La confección de los casos de prueba no es sencilla, pero, en cualquier caso, el objetivo que debe guiar el diseño de las pruebas debe ser intentar mostrar que el software es inconsistente con cada punto de la especificación de requisitos u objetivos del sistema. Para ello, los casos de prueba deben cubrir tres fuentes de procedencia:

- Casos basados en los requisitos del sistema, generados mediante técnicas de «caja negra».
- Casos basados en el diseño; que se generan mediante técnicas de «caja blanca» modificadas.
- Casos necesarios para probar el rendimiento del software y su capacidad funcional.

Las pruebas de implantación se llevan a cabo en el entorno de operación y tienen por objeto comprobar el correcto funcionamiento del sistema integrado de hardware y software en dicho entorno, así como permitir que el usuario, desde el punto de vista de operación, realice la aceptación del sistema una vez instalado en su entorno real y en base al cumplimiento de los requisitos no funcionales especificados.

Las pruebas de implantación son un subconjunto de las pruebas del sistema que incluyen, básicamente, las pruebas de seguridad, las de rendimiento y las de operación. Además, se realizan también las pruebas de gestión de copias de seguridad y recuperación, a fin de verificar que el sistema, al existir un control y seguimiento de los procedimientos de salvaguarda y recuperación, sigue funcionando en caso de caídas en los servicios o en algunos de sus componentes.

Las pruebas de implantación son realizadas por el equipo de operación y son responsabilidad del equipo de desarrollo y del responsable de la instalación, que son los que determinan las pruebas a realizar y los criterios de captación del sistema.

8. PRUEBA DE ACEPTACIÓN.

Las pruebas de aceptación, última fase del proceso de prueba, tienen por objetivo comprobar que el sistema cumple con el funcionamiento esperado, es decir, con los requisitos planteados en la especificación que cubre las necesidades de los usuarios finales desde el punto de vista de su funcionalidad y rendimiento y que por tanto, el sistema está listo para el uso operativo en el entorno en el que se va a explotar el sistema.

Estas pruebas están enfocadas a crear la confianza en el usuario del sistema a fin de su aceptación por parte de éste, por tanto, debe ser el usuario quien las defina (ayudado por el equipo de desarrollo) y quien las realice. Asimismo, es muy recomendable en este tipo de prueba la aprobación por el usuario de los criterios de aceptación y la validación de la documentación y de los procedimientos a usar.

El diseño de las pruebas de aceptación es similar al de las pruebas del sistema y de hecho comparte casos de prueba con ésta. La realización de las pruebas se lleva a cabo mediante técnicas de «caja negra» que demuestren la conformidad con los requisitos funcionales especificados por el usuario y con los requisitos no funcionales relacionados con el rendimiento, y la seguridad de acceso al sistema, a los datos, a los procesos y a los recursos.

9. PRUEBAS DE REGRESIÓN.

Las pruebas de regresión están asociadas a la fase de mantenimiento del sistema y su objetivo es comprobar que los cambios efectuados sobre un componente del sistema de información no introducen un comportamiento no deseado o errores adicionales en los componentes no modificados.

La responsabilidad de estas pruebas recae en el técnico de mantenimiento, que será responsable de especificar el plan de pruebas de regresión y de evaluar los resultados de las mismas, y en el equipo de desarrollo que será responsable de realizar las pruebas.

Las pruebas de regresión se deben realizar cada vez que se efectúa un cambio en el sistema, ya sea para corregir un error o para introducir una mejora, a fin de controlar que las modificaciones no producen efectos negativos sobre el mismo u otros componentes. Normalmente implican la repetición de las pruebas que ya se han realizado previamente (unitarias, de integración, de sistema y de aceptación) y pueden incluir:

- La repetición de casos de pruebas que se han realizado anteriormente y que están relacionados con la parte del sistema modificada.
- La revisión de los procedimientos manuales preparados antes del cambio para asegurar que permanecen correctamente.
- La obtención impresa del diccionario de datos de forma que se compruebe que los elementos de datos que han sufrido algún cambio son correctos.

10. PRUEBAS ESTÁTICAS. REVISIONES.

Como ya se indicó, las pruebas del software tienen dos orientaciones diferentes: las pruebas dinámicas, esto es, aquellas que requieren la ejecución del objeto que se está probando (precisan utilizar el ordenador), que se realizan una vez finalizada la instrumentación y abarcan desde la prueba de unidad, aplicada al módulo, hasta la prueba de aceptación, aplicada al sistema global en el entorno del usuario, y las pruebas o análisis estáticos, no basadas en la utilización del ordenador, y que se aplican en las primeras fases del desarrollo al análisis y al diseño, así como al propio código antes de que se ejecute. Genéricamente, las pruebas estáticas reciben el nombre de «revisiones».

Las revisiones, que nacen como una herramienta de control para el producto software, se han convertido en una clase general de métodos de prueba. De hecho, las revisiones son la única técnica de prueba disponible para las primeras fases del proceso de desarrollo (análisis y diseño), cuando no existe código. Puesto que uno de los objetivos de la prueba es la prevención de errores, los detectados y corregidos en las fases de análisis y diseño no se sufren en el código, de ahí que las revisiones deban ser lo más efectivas posibles.

Una revisión puede definirse como un conjunto de actividades, realizadas por un grupo de personas que trabajan juntas, que afectan a la evaluación de aspectos técnicos con el objetivo de obtener información fiable de los aspectos evaluados.

De la propia definición se desprende que, en función del aspecto evaluado, las revisiones pueden ser de diferentes tipos: revisión de requisitos, de especificaciones, de diseño, de código, de documentación, etc.

Asimismo, según la manera de llevar a cabo las revisiones, éstas pueden ser formales e informales. Ambas tienen los mismos objetivos, pero se diferencian en que en las revisiones formales los participantes son responsables de la fiabilidad de la evaluación y generan un informe que refleja el acto de la revisión, mientras que las revisiones informales no dejan de ser un intercambio de opiniones entre los participantes, de ahí que sólo se consideren las primeras como técnicas de prueba.

10.1. REVISIÓN O «PRUEBA» DE REQUISITOS.

Históricamente, el producto software menos probado han sido los requisitos. Sin embargo, con la nueva visión de la función de prueba, en paralelo con el desarrollo, esto ha cambiado. La prueba de requisitos tiene dos objetivos básicos, que responden a las siguientes cuestiones:

- ¿Existe algún requisito no identificado? Es decir, ¿están identificadas todas las funciones?, ¿está totalmente definido el software?, ¿está especificado el rendimiento?, etc.
- ¿Puede ser especificado o eliminado algún requisito? Esto es, ¿pueden juntarse?, ¿son redundantes o contradictorios?, etc.

Si la prueba en general es difícil, la de los requisitos en particular lo es más, debido a que lo que se prueba es la definición del problema, es decir, si la solución propuesta en el entorno del usuario resuelve el problema. A esto hay que añadir como mayor dificultad que en esta fase aún no existe especificación de diseño ni estructuras de datos.

Una herramienta útil para identificar cada requisito, los casos de prueba asociados y la situación de los mismos es la matriz de validación de requisitos. Esta técnica proporciona muchas ventajas, entre otras, asegurar que los requisitos están listados y qué pruebas tienen asociadas para su validación, facilitando la revisión y pruebas de éstos.

10.2. REVISIÓN O «PRUEBA» DE DISEÑO.

Teniendo en cuenta que el diseño consiste en transformar la especificación de requisitos en una solución en el dominio de implementación, los objetivos de la prueba de diseño son:

- Determinar si la solución elegida es la mejor de todas las opciones, es decir, si es la más simple y la forma más fácil de realizar el trabajo.
- Determinar si la solución abarca todos los requisitos descritos en la especificación, Esto es, si están todos los requisitos identificados por el diseño y si la solución plasmada en el diseño realizará la función encomendada al software.

Al igual que la prueba de requisitos, la prueba de diseño es crucial, pues los errores detectados en el código o durante el uso implican empezar de nuevo realizando el rediseño del sistema.

10.3. REVISIÓN O «PRUEBA» DE CÓDIGO.

En los años 50 y 60 se tenía la idea de que el código servía para ser leído y ejecutado por la máquina y por tanto, las personas no lo leían. Esta idea cambió a partir de los años 70 cuando Weimberg establece que la gente debe leer los programas puesto que ello constituye un proceso efectivo de detección de errores. Así es cómo surgen las revisiones a nivel de código, que con el paso del tiempo resultan muy efectivas, y que son aplicables en el intervalo definido entre el final de la instrumentación y el inicio de las pruebas dinámicas.

Existen diferentes tipos de revisiones aplicables al código, cada una con sus propios objetivos y beneficios. Las más destacables son: las inspecciones de código y los walkthroughs.

- Las inspecciones de código constituyen uno de los primeros métodos de prueba humana y consisten en la inspección del código de un programa por un grupo de personas, normalmente cuatro:
 - El moderador, que nunca es el autor del programa.
 - El programador que ha realizado el trabajo.
 - El diseñador del programa.
 - Un especialista de pruebas.
- Los walkthroughs consisten en simular la ejecución de casos de prueba para un programa por parte de un equipo de revisión formado normalmente por entre tres y seis personas:
 - El moderador.

- Un secretario, cuya misión es registrar los errores.
- Un especialista de pruebas.
- El programador.
- Un número de personas variables, dos o tres, cuya elección se hace en base a diferentes criterios: formación, experiencia, etc.

BIBLIOGRAFÍA

- Ingeniería del Software. Un enfoque práctico. ROGER S. PRESSMAN. Ed. McGraw Hill.
- Metodología de Planificación y Desarrollo de Sistemas de Información. Métrica versión 2.1. Guía de Técnicas. MINISTERIO PARA LAS ADMINISTRACIONES PÚBLICAS. Ed. Tecnos.
- Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información. Métrica versión 3. MINISTERIO PARA LAS ADMINISTRACIONES PÚBLICAS.
- Plan General de Garantía de Calidad aplicable al desarrollo de equipos lógicos. MINISTERIO PARA LAS ADMINISTRACIONES PÚBLICAS.
- Temario de las pruebas selectivas para ingreso en el Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado. ASTIC.
- Temario de las pruebas selectivas para el acceso, por promoción interna, al Cuerpo de Gestión de Sistemas e Informática de la Administración del Estado. MINISTERIO PARA LAS ADMINISTRACIONES PÚBLICAS.
- Temario del Máster en Ingeniería del Software. FACULTAD DE INFORMÁTICA. UNIVERSIDAD POLITÉCNICA DE MADRID. Ed Centro de Estudios Financieros.

