Unidad 4: Prueba, Implantación y Evolución.

El Proceso de Pruebas.

Una estrategia de prueba de software proporciona una guía que describe los pasos que deben realizarse como parte de la prueba, cuándo se planean y se llevan a cabo dichos pasos, y cuánto esfuerzo, tiempo y recursos se requerirán. Por tanto, cualquier estrategia de prueba debe incorporar la planificación de la prueba, el diseño de casos de prueba, la ejecución de la prueba, y la recolección y evaluación de los resultados.

Una **prueba** es un conjunto de actividades que se planean con anticipación y se realizan de manera sistemática. Tiene como objetivos:

- Ejecutar un programa con la intención de descubrir un error.
- Definir casos de pruebas con alta probabilidad de encontrar errores.

Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Un Enfoque Estratégico para la Prueba de Software.

La prueba es un conjunto de actividades que pueden planearse por adelantado y realizarse de manera sistemática. Por esta razón, durante el proceso de software, debe definirse una plantilla para la prueba del software: un conjunto de pasos que incluyen métodos de prueba y técnicas de diseño de casos de prueba específicos.

- Para realizar una prueba efectiva, debe realizar revisiones técnicas efectivas. Al hacerlo, eliminará muchos errores antes de comenzar la prueba.
- La prueba comienza en los componentes y opera “hacia afuera”, hacia la integración de todo el sistema de cómputo.
- Diferentes técnicas de prueba son adecuadas para distintos enfoques de ingeniería de software y en diferentes momentos en el tiempo.
- Las pruebas las realiza el desarrollador del software y (para proyectos grandes) un grupo de prueba independiente.
- Prueba y depuración son actividades diferentes, pero la depuración debe incluirse en cualquier estrategia de prueba.

Una estrategia para la prueba de software debe incluir pruebas de bajo nivel, que son necesarias para verificar que un pequeño segmento de código fuente se implementó correctamente, así como pruebas de alto nivel, que validan las principales funciones del sistema a partir de los requerimientos del cliente.

Validación y Verificación del Software.

La prueba de software es un elemento de un tema más amplio que usualmente se conoce como verificación y validación (V&V). La **verificación** se refiere al conjunto de tareas que garantizan que el software implementa correctamente una función específica. La **validación** es un conjunto de tareas que aseguran que el software que se construye sigue los requerimientos del cliente. Boehm afirma esto de esta forma:

Verificación: “¿Construimos el producto correctamente?”

Validación: “¿Construimos el producto correcto?”

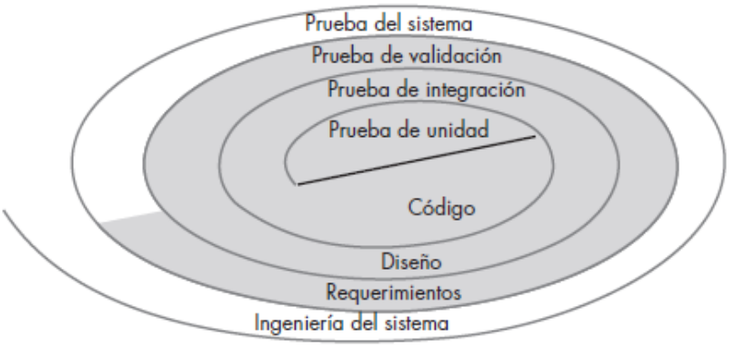
La verificación y la validación incluyen un amplio arreglo de actividades SQA: revisiones técnicas, auditorías de calidad y configuración, monitoreo de rendimiento, simulación, estudio de factibilidad, revisión de documentación, revisión de base de datos, análisis de algoritmos, pruebas de desarrollo, pruebas de usabilidad, pruebas de calificación, pruebas de aceptación y pruebas de instalación.

La verificación y la Validación es el proceso de desarrollo que emplea métodos rigurosos para evaluar la corrección y calidad del producto a lo largo de todo su ciclo de vida.

Estrategias de Prueba de Software. Visión General.

Una estrategia para probar el software también puede verse en el contexto de la espiral. La prueba de unidad comienza en el vértice de la espiral y se concentra en cada unidad (por ejemplo, componente, clase o un objeto) del software como se implementó en el código fuente. La prueba

avanza al moverse hacia afuera a lo largo de la espiral, hacia la prueba de integración, donde el enfoque se centra en el diseño y la construcción de la arquitectura del software. Al dar otra vuelta hacia afuera de la espiral, se encuentra la prueba de validación, donde los requerimientos establecidos como parte de su modelado se validan confrontándose con el software que se construyó. Finalmente, se llega a la prueba del sistema, donde el software y otros elementos del sistema se prueban como un todo. Para probar el software de cómputo, se avanza en espiral hacia afuera en dirección de las agujas del reloj a lo largo de líneas que ensanchan el alcance de las pruebas con cada vuelta.

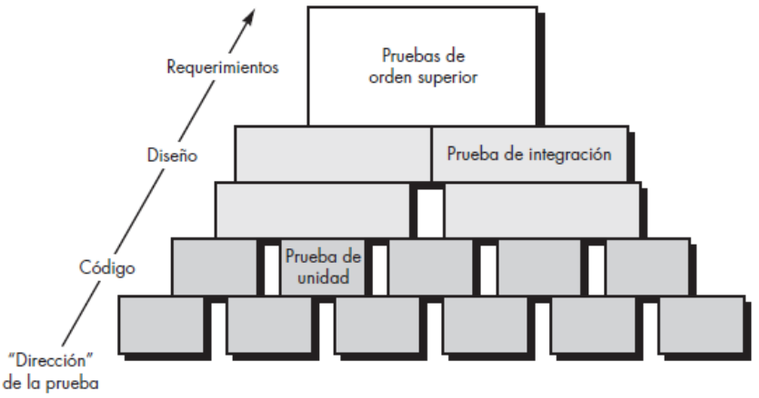


Estrategias de Prueba

Al considerar el proceso desde un punto de vista procedural, las pruebas dentro del contexto de la ingeniería del software en realidad son una serie de cuatro pasos que se implementan de manera secuencial.

Éstos pasos se muestran en la figura que sigue. Inicialmente, las pruebas se enfocan en cada componente de manera individual, lo que garantiza que funcionan adecuadamente como unidad. De ahí el nombre de prueba de unidad. Esta prueba utiliza mucho de las técnicas de prueba que ejercitan rutas específicas en una estructura de control de componentes para asegurar una cobertura completa y la máxima detección de errores.

A continuación, los componentes deben ensamblarse o integrarse para formar el paquete de software completo. La prueba de integración aborda los conflictos asociados con los problemas duales de verificación y construcción de programas. Durante la integración, se usan más las técnicas de diseño de casos de prueba que se enfocan en entradas y salidas, aunque también pueden usarse técnicas que ejercitan rutas de programa específicas para asegurar la cobertura de las principales rutas de control. Después de integrar (construir) el software, se realiza una serie de pruebas de orden superior. Deben evaluarse criterios de validación (establecidos durante el análisis de requerimientos). La prueba de validación proporciona la garantía final de que el software cumple con todos los requerimientos informativos, funcionales, de comportamiento y de rendimiento.



Pasos de la Prueba del Software

El último paso de la prueba de orden superior cae fuera de las fronteras de la ingeniería de software y en el contexto más amplio de la ingeniería de sistemas de cómputo. El software, una vez validado, debe combinarse con otros elementos del sistema (por ejemplo, hardware, personal, bases de datos). La prueba del sistema verifica que todos los elementos se mezclan de manera adecuada y que se logra el funcionamiento/rendimiento global del sistema.

Estrategias para la Prueba de Software.

Prueba de unidad.

La prueba de unidad enfoca los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de software. Al usar la descripción del diseño de componente como guía, las rutas de control importantes se prueban para descubrir errores dentro de la frontera del módulo. La relativa complejidad de las pruebas y los errores que descubren están limitados por el ámbito restringido que se establece para la prueba de unidad. Las pruebas de unidad

se enfocan en la lógica de procesamiento interno y de las estructuras de datos dentro de las fronteras de un componente. Este tipo de pruebas puede realizarse en paralelo para múltiples componentes.

Consideraciones de las pruebas de unidad. La interfaz del módulo se prueba para garantizar que la información fluya de manera adecuada hacia y desde la unidad de software que se está probando. Las estructuras de datos locales se examinan para asegurar que los datos almacenados temporalmente mantienen su integridad durante todos los pasos en la ejecución de un algoritmo. Todas las rutas independientes a través de la estructura de control se ejercitan para asegurar que todos los estatutos en un módulo se ejecuten al menos una vez. Las condiciones de frontera se prueban para asegurar que el módulo opera adecuadamente en las fronteras establecidas para limitar o restringir el procesamiento. Y, finalmente, se ponen a prueba todas las rutas para el manejo de errores.

El flujo de datos a través de la interfaz de un componente se prueba antes de iniciar cualquiera otra prueba. Si los datos no entran y salen de manera adecuada, todas las demás pruebas son irrelevantes.

La prueba selectiva de las rutas de ejecución es una tarea esencial durante la prueba de unidad. Los casos de prueba deben diseñarse para descubrir errores debidos a cálculos erróneos, comparaciones incorrectas o flujo de control inadecuado.

Procedimientos de prueba de unidad. El diseño de las pruebas de unidad puede ocurrir antes de comenzar la codificación o después de generar el código fuente. La revisión de la información del diseño proporciona una guía para establecer casos de prueba que es probable que descubran errores en cada una de las categorías analizadas anteriormente. Cada caso de prueba debe acoplarse con un conjunto de resultados esperados.

Puesto que un componente no es un programa independiente, con frecuencia debe desarrollarse software controlador y/o de resguardo para cada prueba de unidad.

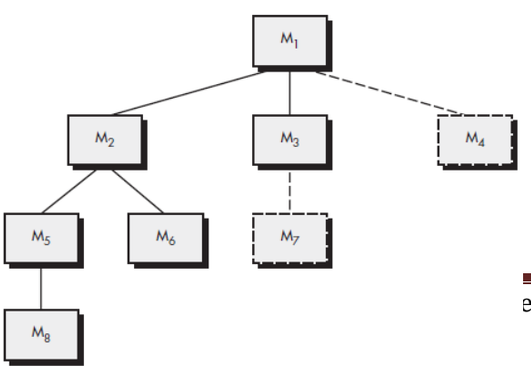
Prueba de Integración.

Las pruebas de integración son una técnica sistemática para construir la arquitectura del software mientras se llevan a cabo pruebas para descubrir errores asociados con la interfaz. El objetivo es tomar los componentes probados de manera individual y construir una estructura de programa que se haya dictado por diseño.

Integración descendente. Los módulos se integran al moverse hacia abajo a través de la jerarquía de control, comenzando con el módulo de control principal (programa principal). Los módulos subordinados al módulo de control principal se incorporan en la estructura en una forma de primero en profundidad o primero en anchura.

El proceso de integración se realiza en una serie de cinco pasos:

1. El módulo de control principal se usa como un controlador de prueba y los representantes (stubs) se sustituyen con todos los componentes directamente subordinados al módulo de control principal.
2. Dependiendo del enfoque de integración seleccionado (es decir, primero en profundidad o anchura), los representantes subordinados se sustituyen uno a la vez con componentes reales.
3. Las pruebas se llevan a cabo conforme se integra cada componente.
4. Al completar cada conjunto de pruebas, otro representante se sustituye con el componente real.
5. Las pruebas de regresión pueden realizarse para asegurar que no se introdujeron nuevos errores.



El proceso continúa desde el paso 2 hasta que se construye todo la estructura del programa.

La estrategia de integración descendente verifica los principales puntos de control o de decisión al principio en el proceso de prueba. En una estructura de programa "bien factorizada", la toma de decisiones ocurre en

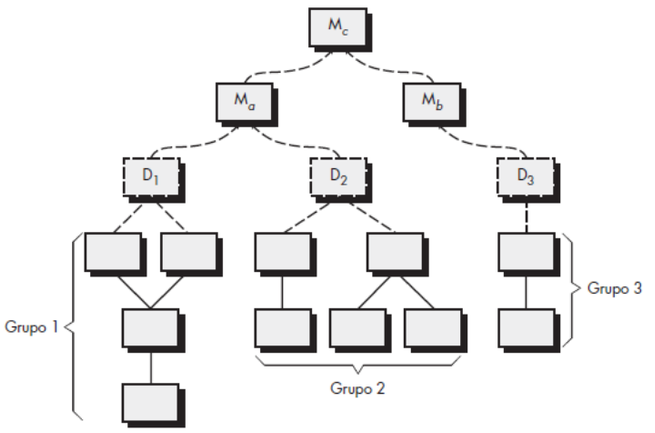
niveles superiores en la jerarquía y, por tanto, se encuentra primero. Si se selecciona la integración primero en profundidad, es posible implementar y demostrar un funcionamiento completo del software.

Integración ascendente. En la prueba de integración ascendente, comienza la construcción y la prueba con módulos atómicos (es decir, componentes en los niveles inferiores dentro de la estructura del programa). Puesto que los componentes se integran de abajo hacia arriba, la funcionalidad que proporcionan los componentes subordinados en determinado nivel siempre está disponible y se elimina la necesidad de representantes (stubs).

Una estrategia de integración ascendente puede implementarse con los siguientes pasos:

1. Los componentes en el nivel inferior se combinan en grupos (en ocasiones llamados construcciones o builds) que realizan una subfunción de software específica.
2. Se escribe un controlador (un programa de control para pruebas) a fin de coordinar la entrada y salida de casos de prueba.
3. Se prueba el grupo.
4. Los controladores se remueven y los grupos se combinan moviéndolos hacia arriba en la estructura del programa.

Los componentes se combinan para formar los grupos 1, 2 y 3 (como se ilustra en la figura). Cada uno de ellos se prueba usando un controlador (que se muestra como un bloque rayado). Los componentes en los grupos 1 y 2 se subordinan a M_a. Los controladores D_1 y D_2 se remueven y los grupos se ponen en interfaz directamente con M_a. De igual modo, el controlador D_3 para el grupo 3 se remueve antes de la integración con el módulo M_b. Tanto M_a como M_b al final se integrarán con el componente M_c, y así sucesivamente.



Conforme la integración avanza hacia arriba, se reduce la necesidad de controladores de prueba separados. De hecho, si los dos niveles superiores del programa se integran de manera descendente, el número de controladores puede reducirse de manera sustancial y la integración de grupos se simplifica enormemente.

Prueba de Validación / Alto Nivel.

Las pruebas de validación comienzan en la culminación de las pruebas de integración, cuando se ejercitaron componentes individuales, el software está completamente ensamblado como un paquete y los errores de interfaz se descubrieron y corrigieron.

Se deben comprobar los criterios de validación (establecidos durante el análisis de requisitos). Proporciona una seguridad final de que el software satisface todos los requisitos funcionales, de comportamiento y de rendimiento. Se usan exclusivamente técnicas de prueba de caja negra. Las pruebas se enfocan en las acciones visibles para el usuario y las salidas del sistema reconocibles por el usuario. La validación es exitosa cuando el software funciona en una forma que cumpla con las expectativas razonables del cliente.

Criterios de pruebas de validación: La validación del software se logra a través de una serie de pruebas que demuestran conformidad con los requerimientos. Un plan de prueba subraya las clases de pruebas que se van a realizar y un procedimiento de prueba define casos de prueba específicos que se diseñan para garantizar que: se satisfacen todos los requerimientos de funcionamiento, se logran todas las características de comportamiento, todo el contenido es preciso y se presenta de manera adecuada, se logran todos los requerimientos de rendimiento, la documentación es correcta y se satisfacen la facilidad de uso y otros requerimientos (por ejemplo, transportabilidad, compatibilidad, recuperación de error, mantenimiento).

Después de realizar cada caso de prueba de validación, existen dos posibles condiciones: **1)** La característica de función o rendimiento se conforma de acuerdo con las especificaciones y se acepta, o **2)** se descubre una desviación de la especificación y se crea una lista de deficiencias.

Un elemento importante del proceso de validación es una revisión de la configuración. La intención de la revisión es garantizar que todos los elementos de la configuración del software se desarrollaron de manera adecuada, y que se cataloga y se tiene el detalle necesario para reforzar las actividades de apoyo.

Pruebas alfa y beta: La mayoría de los constructores de productos de software usan un proceso llamado prueba alfa y prueba beta para descubrir errores que al parecer sólo el usuario final es capaz de encontrar.

La **prueba alfa** se lleva a cabo en el sitio del desarrollador por un grupo representativo de usuarios finales. El software se usa en un escenario natural con el desarrollador "mirando sobre el hombro" de los usuarios, y registrando los errores y problemas de uso. Las pruebas alfa se realizan en un ambiente controlado.

La **prueba beta** se realiza en uno o más sitios del usuario final y por lo general el desarrollador no está presente. Por tanto, la prueba beta es una aplicación "en vivo" del software en un ambiente que no puede controlar el desarrollador. El cliente registra todos los problemas que se encuentran durante la prueba beta y los reporta al desarrollador periódicamente.

En ocasiones se realiza una variación de la prueba beta, llamada prueba de aceptación del cliente, cuando el software se entrega a un cliente bajo contrato. El cliente realiza una serie de pruebas específicas con la intención de descubrir errores antes de aceptar el software del desarrollador.

Pruebas del Sistema.

El software, una vez validado, se debe combinar con otros elementos del sistema (por ejemplo, hardware, personas, bases de datos, información) y se lleva a cabo una serie de pruebas de integración y validación del sistema. Se verifica que cada elemento encaja de forma adecuada, y que se alcanza la funcionalidad y el rendimiento del sistema total.

Un problema clásico en la prueba del sistema es el "dedo acusador". Esto ocurre cuando se descubre un error y los desarrolladores de diferentes elementos del sistema se culpan unos a otros por el problema. Estos deben anticiparse a los problemas de interfaz y: **1)** diseñar rutas de manejo de error que prueben toda la información proveniente de otros elementos del sistema, **2)** realizar una serie de pruebas que simulen los datos malos u otros errores potenciales en la interfaz del software, **3)** registrar los resultados de las pruebas para usar como "evidencia" si ocurre el dedo acusador, y **4)** participar en planificación y diseño de pruebas del sistema para garantizar que el software se prueba de manera adecuada.

En realidad, la prueba del sistema es una serie de diferentes pruebas cuyo propósito principal es ejercitar por completo el sistema basado en computadora, es decir, son pruebas de integración del sistema de información completo. Permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen.

Las pruebas de sistemas incluyen:

Pruebas de recuperación: fuerza al software a fallar en varias formas y que verifica que la recuperación se realice de manera adecuada. Si la recuperación es automática (realizada por el sistema en sí), se evalúa el reinicio, los mecanismos de puntos de verificación, la recuperación de datos y la reanudación para correcciones. Si la recuperación requiere intervención humana, se evalúa el tiempo medio de reparación para determinar si está dentro de límites aceptables.

Pruebas de seguridad: intenta verificar que los mecanismos de protección que se construyen en un sistema lo protegerán de cualquier penetración impropia. Durante la prueba de seguridad, quien realiza la prueba juega el papel del individuo que desea penetrar al sistema. Este puede intentar adquirir contraseñas por medios administrativos externos; puede atacar el sistema con software a la medida diseñado para romper cualquier defensa que se haya construido; puede abrumar al sistema, y por tanto negar el servicio a los demás; puede causar a propósito errores del sistema con la

esperanza de penetrar durante la recuperación; puede navegar a través de datos inseguros para encontrar la llave de la entrada al sistema.

Pruebas de esfuerzo: Se diseñan para enfrentar los programas con situaciones anormales, es decir, ejecuta un sistema en forma que demanda recursos en cantidad, frecuencia o volumen anormales. Por ejemplo, pueden ejecutarse casos de prueba que requieran memoria máxima y otros recursos, crearse casos de prueba que puedan causar búsqueda excesiva por datos residentes en disco. En esencia, la persona que realiza la prueba intenta romper el programa.

Pruebas de rendimiento: Se diseña para poner a prueba el rendimiento del software en tiempo de corrida, dentro del contexto de un sistema integrado. Las pruebas de rendimiento con frecuencia se aparean con las pruebas de esfuerzo y por lo general requieren instrumentación de hardware y de software, es decir, con frecuencia es necesario medir la utilización de los recursos (por ejemplo, ciclos del procesador) en forma meticulosa.

Pruebas de despliegue o configuración: ejercita el software en cada entorno en el que debe operar. Además, examina todos los procedimientos de instalación y el software de instalación especializado (por ejemplo, "instaladores") que usarán los clientes, así como toda la documentación que se usará para introducir el software a los usuarios finales.

Métodos y Técnicas.

Prueba de Caja Blanca.

La prueba de caja blanca o caja de vidrio, es una filosofía de diseño de casos de prueba que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba. Al usar los métodos de prueba de caja blanca, puede derivar casos de prueba que: 1) garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez, 2) revisen todas las decisiones lógicas en sus lados verdadero y falso, 3) ejecuten todos los bucles en sus fronteras y dentro de sus fronteras operativas y 4) revisen estructuras de datos internas para garantizar su validez. Las técnicas pueden ser pruebas de ruta básica o prueba de la estructura de control.

Prueba de Ruta Básica:

El método de ruta básica permite al diseñador de casos de prueba derivar una medida de complejidad lógica de un diseño de procedimiento y usar esta medida como guía para definir un conjunto básico de rutas de ejecución. Los casos de prueba derivados para revisar el conjunto básico tienen garantía para ejecutar todo enunciado en el programa, al menos una vez durante la prueba.

Prueba de la Estructura de Control:

La técnica de prueba de ruta básica que es una de varias técnicas para probar la estructura de control es simple y enormemente efectiva, pero no es suficiente en sí misma. Por lo que se presentan otras variaciones acerca de la prueba de la estructura de control.

Prueba de condición: Es un método de diseño de casos de prueba que revisa las condiciones lógicas contenidas en un módulo de programa para asegurar que no contiene errores. Si una condición es incorrecta, entonces al menos un componente de la condición es incorrecto. Por tanto, los tipos de errores en una condición incluyen errores de operador booleano (operadores booleanos incorrectos / perdidos / adicionales), de variable booleana, de paréntesis booleanos, de operador relacional y de expresión aritmética.

Prueba de flujo de datos: Este método selecciona rutas de prueba de un programa de acuerdo con las ubicaciones de las definiciones y con el uso de variables en el programa.

Prueba de bucle: Es una técnica que se enfoca exclusivamente en la validez de los constructos bucle.

Prueba de Caja Negra.

Las pruebas de caja negra o pruebas de comportamiento, se enfocan en los requerimientos funcionales del software; es decir, permiten derivar conjuntos de condiciones de entrada que revisarán por completo todos los requerimientos funcionales para un programa. No son una

alternativa para las técnicas de caja blanca, es un enfoque complementario que es probable que descubra una clase de errores diferente que los métodos de caja blanca.

Las pruebas de caja negra intentan encontrar errores en las categorías siguientes: **1)** funciones incorrectas o faltantes, **2)** errores de interfaz, **3)** errores en las estructuras de datos o en el acceso a bases de datos externas, **4)** errores de comportamiento o rendimiento y **5)** errores de inicialización y terminación. La prueba de caja negra no considera la estructura de control, la atención se enfoca en el dominio de la información.

Partición de Equivalencia:

Es un método de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos de los que pueden derivarse casos de prueba. El diseño de casos de prueba para la partición de equivalencia se basa en una evaluación de las clases de equivalencia para una condición de entrada. Si un conjunto de objetos puede vincularse mediante relaciones que son simétricas, transitivas y reflexivas, se presenta una clase de equivalencia. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada. Por lo general, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición booleana. Las clases de equivalencia pueden definirse de acuerdo con los siguientes lineamientos:

1. Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos inválidas.
2. Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos inválidas.
3. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una inválida.
4. Si una condición de entrada es booleana, se define una clase válida y una inválida.

Análisis de Valor Límite o Frontera:

Un mayor número de errores ocurre en las fronteras del dominio de entrada y no en el "centro". Por esta razón es que el análisis de valor de límite se desarrolló como una técnica de prueba. El análisis de valor de frontera conduce a una selección de casos de prueba que revisan los valores de frontera. Es una técnica de diseño de casos de prueba que complementan la partición de equivalencia. En lugar de seleccionar algún elemento de una clase de equivalencia, el AVL conduce a la selección de casos de prueba en los "bordes" de la clase. En lugar de enfocarse exclusivamente en las condiciones de entrada, el AVL también deriva casos de prueba a partir del dominio de salida.

Los lineamientos para el AVL son:

1. Si una condición de entrada especifica un rango acotado por valores a y b, los casos de prueba deben designarse con valores a y b, justo arriba y justo abajo de a y b.
2. Si una condición de entrada especifica un número de valores, deben desarrollarse casos de prueba que revisen los números mínimo y máximo. También se prueban los valores justo arriba y abajo, mínimo y máximo.
3. Aplicar lineamientos 1 y 2 a condiciones de salida. Por ejemplo, suponga que como salida de un programa de análisis de ingeniería se requiere una tabla de temperatura contra presión. Deben diseñarse casos de prueba para crear un reporte de salida que produzca el número máximo (y mínimo) permisible de entradas de tabla.
4. Si las estructuras de datos de programa internos tienen fronteras prescritas (por ejemplo, una tabla que tenga un límite definido de 100 entradas), asegúrese de diseñar un caso de prueba para revisar la estructura de datos en su frontera.

Prueba de Arreglo Ortogonal:

Existen muchas aplicaciones en las cuales el dominio de entrada es relativamente limitado, es decir, el número de parámetros de entrada es pequeño y los valores que cada uno de los parámetros puede tomar están claramente acotados. Cuando dichos números son muy pequeños (por ejemplo, tres parámetros de entrada que toman tres valores discretos cada uno), es posible considerar cada permutación de entrada y probar de manera exhaustiva el dominio de entrada. Sin embargo,

conforme crece el número de valores de entrada y el número de valores discretos para cada ítem de datos, la prueba exhaustiva se vuelve impráctica o imposible.

La prueba de arreglo ortogonal puede aplicarse a problemas en los que el dominio de entrada es relativamente pequeño pero demasiado grande para alojar la prueba exhaustiva. El método de prueba de arreglo ortogonal es particularmente útil para encontrar los fallos de región, una categoría de error asociada con lógica defectuosa dentro de un componente de software.

Prueba Basada en Modelo:

La prueba basada en modelo es una técnica de prueba de caja negra que usa la información contenida en el modelo de requerimientos como la base para la generación de casos de prueba. En muchos casos, usa diagramas de estado UML, un elemento del modelo de comportamiento, como la base para el diseño de los casos de prueba. Esta técnica ayuda a descubrir errores en el comportamiento del software y, como consecuencia, es extremadamente útil cuando se prueban aplicaciones impulsadas por un evento.

Prueba para Entornos, Arquitecturas y Aplicaciones Especializados.

Prueba de Interfaces Graficas de Usuario:

Debido a que muchas GUI modernas tienen la misma apariencia y ambiente, puede derivarse una serie de pruebas estándar. Es posible usar las gráficas de modelado de estado finito para derivar una serie de pruebas que aborden objetos de datos y programa específicos que sean relevantes para la GUI. Esta es la técnica de prueba basada en modelo.

Prueba de Arquitecturas Cliente-Servidor:

La naturaleza distribuida de los entornos cliente-servidor, los conflictos de rendimiento asociados con el procesamiento de transacciones, la presencia de algunas plataformas de hardware diferentes, las complejidades de la comunicación en red, la necesidad de atender a múltiples clientes desde una base de datos y los requerimientos de coordinación impuestos al servidor se combinan para realizar las pruebas de las arquitecturas cliente-servidor y el software que reside dentro de ellas es más difícil que las aplicaciones independientes.

La prueba del software cliente-servidor ocurre en tres niveles diferentes:

1. Las aplicaciones clientes individuales se prueban en un modo “desconectado”; no se considera la operación del servidor ni la red subyacente.
2. El software cliente y las aplicaciones servidor asociadas se prueban en concierto, pero las operaciones de red no se revisan de manera explícita.
3. Se prueba la arquitectura cliente-servidor completa, incluidos la operación de red y el rendimiento.

Aunque en cada uno de estos niveles de detalle se realizan muchos tipos de pruebas diferentes, para las aplicaciones cliente-servidor se encuentran los siguientes abordajes de prueba:

Pruebas de función de aplicación. La funcionalidad de las aplicaciones cliente se prueba usando los métodos analizados anteriormente. La aplicación se prueba en forma independiente con la intención de descubrir errores en su operación.

Pruebas de servidor. Se prueban las funciones de coordinación y gestión de datos del servidor. También se considera el rendimiento del servidor (tiempo de respuesta y cantidad de datos transmitidos).

Pruebas de base de datos. Se prueban la precisión y la integridad de los datos almacenados por el servidor. Se examinan las transacciones colocadas por las aplicaciones cliente para asegurar que los datos se almacenen, actualicen y recuperen de manera adecuada.

Pruebas de transacción. Se crea una serie de pruebas para garantizar que cada clase de transacciones se procese de acuerdo con los requerimientos. Las pruebas se enfocan en comprobar el procesamiento y en los conflictos de rendimiento (por ejemplo, tiempos de procesamiento de transacción y volumen de transacción).

Pruebas de comunicación de red. Verifican que la comunicación entre los nodos de la red ocurre de manera correcta. También pueden realizarse pruebas de seguridad de red.

Otras Pruebas.

Las pruebas de implantación incluyen las verificaciones necesarias para asegurar que el sistema funcionará correctamente en el entorno de operación al responder satisfactoriamente a los requisitos de rendimiento, seguridad y operación, y coexistencia con el resto de los sistemas de la instalación, y conseguir la aceptación del sistema por parte del usuario de operación.

Las pruebas de aceptación van dirigidas a validar que el sistema cumple los requisitos de funcionamiento esperado, recogidos en el catálogo de requisitos y en los criterios de aceptación del sistema de información, y conseguir la aceptación final del sistema por parte del usuario.

Plan de Implantación.

La implantación es un conjunto de actividades que se planean para poner en funcionamiento el sistema desarrollado. Comprende las actividades que se deben realizar una vez concluido el diseño.

- Depende de la complejidad y alcance del proyecto total.
- Se debe identificar el tiempo que lleva cada actividad y los recursos humanos.
- Una implantación exitosa, no garantiza que la organización mejore, pero, la instalación inadecuada, seguramente, lo impedirá.

Actividades del Plan de Implantación:

- Capacitación del personal.
- Conversión.
- Acondicionamiento de las instalaciones (Preparación del sitio).
- Revisión posterior de la implantación.

Otras Actividades del Plan de Implantación.

- Solicitar la tecnología.
- Preparar el lugar.
- Revisar especificaciones.
- Instalar tecnología.
- Escribir programas.
- Capacitar y educar a los usuarios.
- Probar la tecnología.
- Realizar las Pruebas.
- Conversión del sistema.
- Implantación y seguimiento.

Capacitación de los Usuarios.

Los analistas de sistemas participan en un proceso educativo con los usuarios el cual se denomina capacitación. En la implementación de proyectos grandes, el analista normalmente estará manejando la capacitación en lugar de estar involucrado personalmente en ella. Uno de los recursos más valiosos que el analista puede aportar en cualquier situación de capacitación es la habilidad de ver el sistema desde el punto de vista del usuario.

Estrategias para la Capacitación.

Los factores que determinan la estrategia de capacitación son las personas que serán capacitadas y quiénes las capacitarán. El analista tendrá que garantizar que cualquiera cuyo trabajo sea afectado por el nuevo sistema de información sea propiamente capacitado por el instructor correcto.

A quien Capacitar.

Todas las personas que tendrán uso principal o secundario del sistema deben recibir capacitación. Esto incluye a todos, desde el personal de entrada de datos hasta aquellos que usarán la salida para tomar decisiones sin usar personalmente una computadora. La cantidad de

capacitación que requiere un sistema depende de cuánto cambiará el trabajo de alguien debido al nuevo sistema.

Debe asegurarse de que los usuarios con diferentes niveles de habilidad e intereses de trabajo estén separados. Habrá problemas si incluye a principiantes en las mismas sesiones de capacitación que los expertos, debido a que los principiantes se pierden con rapidez y los expertos se aburren con los elementos básicos. En consecuencia, ambos grupos se pierden.

Capacitación de Operadores: es decir, las personas responsables de mantener los equipos funcionando, que además, proporcionan el servicio de apoyo necesario. Su capacitación debe asegurar que puedan manejar todas las operaciones posibles. Se les debe dar información de los desperfectos más comunes, de cómo reconocerlos y de los pasos a seguir para solucionarlos. Los operadores deberán tener conocimiento además, del tiempo que insumirá la ejecución de cada una de las aplicaciones, bajo condiciones normales.

Capacitación de Usuarios: implica el uso del nuevo sistema y del equipo (dependiendo de los conocimientos y habilidades de los usuarios). En algunas organizaciones, los usuarios deberán preparar discos, formatearlos, cambiar cartuchos, poner papel en las impresoras, etc., por lo que un programa de capacitación, deberá incluir estas actividades.

La falta de una capacitación correcta trae la posibilidad de llegar a situaciones que, no sólo producirán errores, sino la propia frustración de los usuarios. Si bien puede existir una buena documentación del sistema, la misma no reemplazará a la capacitación, dado que no hay sustituto para la experiencia directa.

Personas que Capacitan a los Usuarios.

Para un proyecto grande, se podrían usar muchos instructores diferentes dependiendo de cuántos usuarios se deben capacitar y quiénes son. Las posibles fuentes de capacitación incluyen lo siguiente:

1. Vendedores.
2. Analistas de sistemas.
3. Instructores externos.
4. Instructores internos.
5. Otros usuarios del sistema.

Lineamientos para la Capacitación.

El analista tiene cuatro lineamientos principales para establecer la capacitación. Éstos son: (1) establecer objetivos medibles, (2) usar métodos de capacitación apropiados, (3) seleccionar sitios de capacitación convenientes y (4) emplear materiales de capacitación entendibles.

Objetivos de la capacitación.

Quien está siendo capacitado dicta, en gran medida, los objetivos de la capacitación. Los objetivos de capacitación para cada grupo se deben explicar claramente. Los objetivos bien definidos son de gran ayuda permitiendo a los aprendices saber lo que se espera de ellos. Además, los objetivos permiten evaluación de capacitación cuando están completos. Por ejemplo, los operadores deben saber dichos elementos esenciales como encender la máquina, qué hacer cuando ocurren errores comunes, solucionar problemas de elementos esenciales y cómo acabar un proceso de entrada.

Métodos de Capacitación.

Cada usuario y operador necesitarán capacitación ligeramente diferente. Hasta cierto punto, sus trabajos determinan lo que necesitan saber y sus personalidades, experiencia y antecedentes determinan cómo aprenden mejor. Algunos usuarios aprenden mejor viendo, otros oyendo e incluso otros haciendo. Debido a que normalmente no es posible personalizar la capacitación para un individuo, una combinación de métodos es a menudo la mejor forma de proceder. Así, la mayoría de los usuarios se atrae mediante un método u otro.

Los métodos para aquellos que aprenden mejor viendo incluyen demostraciones de equipo y exposición para manuales de capacitación. Aquellos que aprenden mejor oyendo se beneficiarán de las conferencias sobre los procedimientos, discusiones y sesiones de preguntas y respuestas entre instructores y aprendices. Aquellos que aprenden mejor haciendo necesitan experiencia práctica con el nuevo equipo.

Sitios de Capacitación.

La capacitación se hace en muchas ubicaciones diferentes, algunas de las cuales son más favorables para aprender que otras. Los grandes vendedores de cómputo proporcionan **ubicaciones remotas especiales** en donde se mantiene equipo operable en forma gratuita. Sus instructores ofrecen experiencia práctica así como también seminarios en situaciones que permiten a los usuarios concentrarse en aprender el nuevo sistema. Una de las desventajas de la capacitación remota es que los usuarios están fuera del contexto organizacional en el cual eventualmente se deben desempeñar.

La capacitación en **las instalaciones de la organización** a la cual pertenecen los usuarios también es posible con varios tipos diferentes de instructores. La ventaja es que los usuarios ven el equipo tal como estará cuando sea totalmente operacional. Una desventaja seria es que los aprendices con frecuencia se sienten culpables de no cumplir sus tareas rutinarias de trabajo si permanecen en el lugar de la capacitación. En estos casos, no se puede lograr una total concentración en la capacitación.

Los sitios de capacitación se pueden establecer en lugares que alquilan espacios para reuniones, tal como un hotel, o incluso podrían ser instalaciones permanentes mantenidas por los instructores.

Materiales de Capacitación.

Los materiales incluyen manuales de capacitación; casos de capacitación, en los cuales los usuarios se asignan para trabajar a través de un caso que incluye la mayoría de las interacciones normalmente encontradas con el sistema, y prototipos y muestras de salidas. Los usuarios de sistemas grandes a veces se podrán capacitar en simulaciones detalladas basadas en Web o software que es idéntico a lo que se escribe o se compra. La mayoría de los vendedores de software de COTS proporciona tutoriales en línea que ilustran las funciones básicas, y los vendedores podrían mantener sitios Web que ofrecen páginas dedicadas a responder preguntas frecuentes, las cuales se pueden descargar e imprimir.

Debido a que el entendimiento del sistema por parte de los usuarios depende de ellos, los materiales de capacitación se deben escribir claramente para el público correcto con un mínimo de jerga.

Conversión.

La conversión es proceso de convertir o cambiar físicamente el sistema de información viejo a uno nuevo o modificado. Hay muchas estrategias de conversión disponibles y cada una debe ser considerada en función de las ventajas que ofrece y de las desventajas que puede ocasionar.

Estrategias de Conversión.

Conversión directa.

La conversión directa significa que en una fecha especificada, el sistema viejo se abandona y el nuevo sistema se pone en uso. La conversión directa sólo puede tener éxito si la comprobación extensa se hace de antemano, y funciona mejor cuando se pueden tolerar algunos retrasos en el procesamiento. Una ventaja de la conversión directa es que los usuarios no tienen ninguna posibilidad de usar el sistema viejo en lugar del nuevo.

Conversión Paralela.

La conversión paralela se refiere a ejecutar al mismo tiempo el sistema viejo y el nuevo, en paralelo. Ambos sistemas se ejecutan simultáneamente por un periodo específico y la confiabilidad de los resultados se examina. Cuando se obtienen los mismos resultados todo el tiempo, el nuevo sistema se pone en uso y el viejo se detiene.

Una ventaja de ejecutar ambos sistemas en paralelo es la posibilidad de verificar los nuevos datos contra los viejos para percibir cualesquier errores en el procesamiento del nuevo sistema.

Entre las desventajas, se incluyen el costo de ejecutar dos sistemas al mismo tiempo, el agobio en los empleados de virtualmente doblar su carga de trabajo durante la conversión, además, los empleados continuarán usando el viejo debido a su familiaridad con él.

Conversión Gradual.

La conversión gradual, o por fases, intenta combinar las mejores características de los dos planes previamente mencionados, sin incurrir en todos los riesgos. El volumen de las transacciones manejado por el nuevo sistema aumenta gradualmente conforme el sistema se introduce por fases. Las ventajas de este enfoque incluyen permitir a usuarios que se involucren gradualmente con el sistema y la posibilidad de descubrir y recuperar errores sin desperdiciar mucho tiempo. Las desventajas de la conversión gradual incluyen tomar demasiado tiempo para colocar el nuevo sistema en el lugar y su impropiedad para la conversión de sistemas pequeños y sencillos.

Conversión de Prototipo Modular.

La conversión de prototipo modular usa la construcción de prototipos modulares y operacionales para cambiar de los sistemas viejos a los nuevos de forma gradual. Las ventajas son que cada módulo se prueba completamente antes de ser usado y que los usuarios se familiarizan con cada módulo conforme se vuelve operacional. La desventaja es que no siempre es posible la elaboración de prototipos y se debe poner especial atención a las interfaces para que los módulos que se construyen realmente trabajen como un sistema.

Conversión Distribuida.

La conversión distribuida se refiere a una situación en que se contemplan muchas instalaciones del mismo sistema, como es el caso en actividades bancarias o franquicias tal como restaurantes o tiendas de ropa. Cuando esta conversión se completa exitosamente, se hacen otras conversiones para sitios determinados.

Una ventaja de la conversión distribuida es que se pueden detectar y contener los problemas en lugar de infligir simultáneamente en todos los sitios. Una desventaja es que incluso cuando una conversión es exitosa, cada sitio tendrá sus propias peculiaridades para trabajar y se deben manejar como corresponde.

Plan de Conversión.

Un plan de conversión debe anticipar los posibles problemas y la forma de enfrentarlos. Los problemas más frecuentes tienen que ver con documentos perdidos, variación de los formatos para los datos, errores en la conversión de datos, extravío de datos o pérdida de archivos. Para cada caso hay que especificar los planes de emergencia adecuados.

La duración de la conversión es un reto y siempre es conveniente que haya un responsable de la conversión, encargado de verificar los acuerdos, revisar los planes de conversión para la entrega del software y preparar las instalaciones. El responsable de la conversión será la persona a contactar, por los futuros usuarios, en caso de la existencia de algún problema durante la conversión.

Preparación de datos y archivos.

La etapa de la conversión que más tiempo insume es la preparación de datos y archivos maestros del sistema. Es necesario contar con estos datos para poner en marcha el sistema y asegurarse que no se pase por alto ningún registro.

Algunas de las **actividades** necesarias para llevar a cabo la conversión son:

1. Listar todos los archivos a convertir.
2. Identificar los datos necesarios para construir los archivos durante la conversión.
3. Listar los documentos nuevos y los procedimientos que se usarán durante la conversión.
4. Identificar los controles a usar en la conversión.
5. Asignar responsabilidades para cada actividad.
6. Verificar los tiempos para la conversión.

Aspectos de Seguridad.

La seguridad de las instalaciones de cómputo, almacén de datos y la información generada es parte de una conversión exitosa. El reconocimiento de la necesidad de seguridad es una consecuencia natural de la creencia de que la información es un recurso organizacional importante.

Aunque no hay tal cosa como un sistema totalmente seguro, las acciones de los analistas y usuarios pretenden mover los sistemas hacia el lado más seguro del espectro, disminuyendo la vulnerabilidad del sistema. Se debe observar que conforme más personas en la organización obtienen mayor poder de la computadora, obtienen acceso a la Web, o se acoplan a las intranets y extranets, la seguridad se vuelve incrementalmente difícil y compleja. A veces, las organizaciones contratarán a un consultor de seguridad para trabajar con el analista de sistemas cuando la seguridad es crucial para el funcionamiento exitoso.

La seguridad es responsabilidad de todos aquellos que están en contacto con el sistema, y sólo es tan buena como la conducta más indefinida o la política en la organización. La seguridad tiene tres aspectos interrelacionados: físico, lógico y conductual. Los tres deben trabajar juntos si la calidad de seguridad permanece alta.

Física.

La seguridad física se refiere a proteger el sitio donde se encuentra la computadora, su equipo y software a través de medios físicos. Puede incluir acceso controlado a las salas de cómputo por medio de signos legibles por la máquina o un registro de entrada y salida del sistema por un humano, usando cámaras de televisión de circuito cerrado para supervisar las áreas de la computadora y frecuentemente apoyando los datos y almacenando los respaldos en un área a prueba de fuego o a prueba de agua.

Además, el equipo de cómputo pequeño se debe asegurar para que un usuario típico no pueda moverlo y se debe garantizar el suministro ininterrumpido de energía eléctrica. Las alarmas que notifican a las personas apropiadas en caso de fuego, inundación o intrusión no autorizada de una persona deben estar en todo momento en funcionamiento activo.

El analista debe tomar las decisiones acerca de la seguridad física cuando esté planeando las instalaciones de cómputo y la compra de equipo.

Lógica.

La seguridad lógica se refiere a los controles lógicos en el software. Los controles lógicos son contraseñas o códigos de autorización de alguna clase. Cuando se usan, permiten al usuario entrar al sistema o a una parte particular de una base de datos con una contraseña correcta.

Una forma para que las redes reduzcan el riesgo de exposición al desafío de la seguridad del mundo exterior es construir un firewall o un sistema similar. Un firewall construye una muralla entre la red interna y la externa de una organización. Se asume que la red interna es confiable y segura, mientras que Internet no lo es. Se pretende que los firewalls impidan comunicación dentro o fuera de la red que no haya sido autorizada y que no se requiera. Un sistema firewall no es un remedio perfecto para la seguridad organizacional y de Internet; sin embargo, es una capa adicional de seguridad que ahora se acepta ampliamente.

Los controles lógicos y físicos son importantes, pero no son suficientemente claros para proporcionar la seguridad adecuada. Los cambios conductuales también son necesarios.

Conductual.

Las expectativas conductuales de una organización están implícitas en sus manuales de política e incluso en letreros anunciados en los carteles de anuncios. Sin embargo, la conducta que los miembros de la organización asimilan también es crítica para el éxito de los esfuerzos de seguridad. [Una razón por la que los firewalls no son totalmente a prueba de ataques es que muchos ataques a los sistemas de información vienen de adentro de la organización.]. La seguridad puede empezar con la identificación de empleados que tendrán acceso a las computadoras, datos e información, para asegurar que sus intereses son consistentes con los intereses de la organización y que entienden por completo la importancia de llevar a cabo los procedimientos de seguridad. Algunas organizaciones han escrito reglas o políticas que prohíben a los empleados navegar en Web durante horas de trabajo o incluso prohíben totalmente la navegación de Web, si el equipo de la compañía está involucrado. Otras corporaciones usan software que bloquea el acceso a los sitios Web que se consideran inaceptables en el lugar de trabajo, tal como juegos, apuestas o sitios pornográficos.

Parte del aspecto conductual de seguridad es supervisar la conducta a intervalos irregulares para cerciorarse de que se están siguiendo los procedimientos apropiados y para corregir cualesquier conductas que se podrían deteriorar con el tiempo. Hacer que el sistema registre el número de inicios de sesión fallidos del usuario es una forma de supervisar si usuarios no autorizados están intentando iniciar sesión del sistema. Es conveniente inventariar periódica y frecuentemente el equipo y software. Además, se deben examinar sesiones largas inusuales o el acceso al sistema atípico después de las horas de oficina.

La salida generada por el sistema se debe reconocer por su potencial de poner a la organización en riesgo en algunas circunstancias. Los controles para la salida incluyen pantallas que sólo se pueden acceder mediante la contraseña, la clasificación de información (es decir, a quién se puede distribuir y cuándo) y el almacenamiento seguro de documentos impresos y almacenados magnéticamente.

En algunos casos, se deben tomar medidas para destruir documentos confidenciales que se encuentran en medios magnéticos y papel.

Evolución del Software.

El desarrollo del software no se detiene cuando un sistema se entrega, sino que continúa a lo largo de la vida de éste. Después de desarrollar un sistema, inevitablemente debe modificarse, con la finalidad de mantenerlo útil. Tanto los cambios empresariales como los de las expectativas del usuario generan nuevos requerimientos para el software existente. Es posible que tengan que modificarse partes del software para corregir errores encontrados durante su operación, para adaptarlo a los cambios en su plataforma de software y hardware, y para mejorar su rendimiento u otras características no funcionales.

La evolución del sistema, como la del software es costosa, por varias razones:

1) Los cambios propuestos tienen que analizarse cuidadosamente desde perspectivas técnicas y de negocios. Los cambios tienen que contribuir a los objetivos del sistema y no deben tener simplemente una motivación técnica.
2) Debido a que los subsistemas nunca son completamente independientes, los cambios en uno pueden afectar de forma adversa al funcionamiento o comportamiento de otros. Por lo tanto será necesario cambiar esos subsistemas.
3) A menudo no se registran las razones del diseño original. Los responsables de la evolución del sistema tienen que resolver por qué se tomaron decisiones particulares de diseño.
4) Al paso del tiempo, su estructura se corrompe por el cambio de tal forma que se incrementan los costos de los cambios adicionales.

Mantenimiento.

El mantenimiento del software es el proceso general de cambiar un sistema después de que éste se entregó. El término usualmente se aplica a software personalizado, en el que grupos de desarrollo separados intervienen antes y después de la entrega. Los cambios realizados al software van desde los simples para corregir errores de codificación, los más extensos para corregir errores de diseño, hasta mejorías significativas para corregir errores de especificación o incorporar nuevos requerimientos. Los cambios se implementan modificando los componentes del sistema existentes y agregándole nuevos componentes donde sea necesario.

Existen los siguientes tipos de mantenimiento de software:

Correctivo (Reparaciones de fallas): Se utiliza para referirse al mantenimiento para la reparación de defectos. Los errores de codificación por lo general son baratos de corregir; los errores de diseño son más costosos, ya que quizás impliquen la reescritura de muchos componentes del programa. Los errores de requerimientos son los más costosos de reparar debido a que podría ser necesario un extenso rediseño del sistema.

Adaptativo (Adaptación ambiental): Algunas veces significa adaptarse a un nuevo entorno o puede significar adaptar el software a nuevos requerimientos. Este tipo de mantenimiento se requiere cuando algún aspecto del entorno del sistema, como el hardware, la plataforma operativa

del sistema u otro soporte, cambia el software. El sistema de aplicación tiene que modificarse para lidiar con dichos cambios ambientales.

Perfectivo (Adición de funcionalidad): Puede significar perfeccionar el software implementando nuevos requerimientos y en otros casos significa mantener la funcionalidad del sistema pero mejorando su estructura y su rendimiento. Este tipo de mantenimiento es necesario cuando varían los requerimientos del sistema, en respuesta a un cambio organizacional o empresarial.

Preventivo: El mantenimiento preventivo implica corregir un problema antes de que se presente. El software de computadoras se deteriora debido al cambio y por esto el mantenimiento preventivo también llamado reingeniería del software, se debe conducir para permitir que el software sirva para la necesidad de los usuarios finales. En esencia, el mantenimiento preventivo hace cambios en programas de computadoras a fin de que se puedan corregir, adaptar y mejorar más fácilmente.