|  |
| --- |
| WORKFLOW DE IMPLEMENTACIÓN |

En la implementación se comienza con el resultado del diseño e implementamos el sistema en términos de componentes, es decir, archivos de código fuente, scripts, archivos de código binario, ejecutables y similares.

El propósito fundamental de la implementación es desarrollar la arquitectura el sistema como un todo.

El resultado principal de la implementación es el modelo de implementación.

Propósitos de la implementación

• Planificar las integraciones de sistema necesarias en cada iteración. Se sigue un enfoque incremental, dando lugar a un sistema que se implementa en una sucesión de pasos pequeños y manejables.

• Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue.

• Implementar las clases y subsistemas encontrados durante el diseño.

• Probar los componentes individualmente, y a continuación integrarlos compilándolos y enlazándolos a uno o más ejecutables.

El papel de la implementación en el ciclo de vida del software

La implementación es el centro durante las iteraciones de construcción, aunque también se lleva a cabo trabajo de implementación durante la fase de elaboración, para crear la línea base ejecutable de la arquitectura, y durante la fase de transición, para tratar defectos tardíos.

Artefactos de la implementación

1. Modelo de implementación.

2. Componente.

3. Subsistema de implementación.

4. Interfaz.

5. Descripción de la arquitectura (vista del modelo de implementación).

6. Plan de integración de construcciones.

Modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables, etc. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros.

Componente

Un componente es el empaquetamiento físico de los elementos de un modelo, como son las clases en el modelo de diseño. Algunos estereotipos estándar son:

• <<executable>> es un programa que puede ser ejecutado en un nodo.

• <<file>> es un fichero que contiene código fuente o datos.

• <<library>> es una librería estática o dinámica.

• <<table>> es una tabla de base de datos.

• <<document>> es un documento.

Subsistema de implementación

Los subsistemas de implementación proporcionan una forma de organizar los artefactos del modelo de implementación en trozos más manejables. Puede estar formado por componentes, interfaces, y otros subsistemas (recursivamente).

Un subsistema de implementación se manifiesta a través de un “mecanismo de empaquetamiento” concreto en un entorno de implementación determinado, como

• Un paquete en Java.

• Un proyecto es Visual Basic.

• Un directorio de ficheros en un proyecto de C++.

• Un paquete en una herramienta de modelado como Rational Rose.

La semántica de la noción de subsistema de implementación se refinará ligeramente cuando se manifieste en un entorno de implementación determinado.

Los subsistemas de implementación están muy relacionados con los subsistemas de diseño en el modelo de diseño. **Los subsistemas de implementación deberían seguir la traza uno a uno de sus subsistemas de diseño correspondientes.**

• El subsistema de implementación debería definir dependencias análogas hacia otros subsistemas de implementación o interfaces.

• El subsistema de implementación debería proporcionar las mismas interfaces.

• El subsistema de implementación debería definir qué componentes o, recursivamente, qué otros subsistemas de implementación dentro del subsistema deberían proporcionar las interfaces proporcionales por el subsistema.

Interfaz

Utilizadas en el modelo de implementación para especificar las operaciones implementadas por componentes y subsistemas de implementación.

Un componente que implementa (y por tanto proporciona) una interfaz ha de implementar correctamente todas las operaciones definidas por la interfaz. Un subsistema de implementación que proporciona una interfaz tiene que también que contener componentes que proporcionen la interfaz u otros subsistemas (recursivamente) que proporcionan la interfaz.

Descripción de la arquitectura (vista del modelo de implementación)

La descripción de la arquitectura contiene una visión del modelo de implementación, el cual representa sus artefactos significativos arquitectónicamente.

**Artefactos** considerados significativos para la arquitectura en el modelo de implementación:

• La descomposición del modelo en subsistemas, sus interfaces y las dependencias entre ellos.

• Componentes clave, como los componentes que siguen la traza de las clases de diseño significativas arquitectónicamente, los componentes ejecutables y los componentes que son generales, centrales, que implementan mecanismos de diseño genéricos de los que dependen muchos otros componentes.

Plan de integración de construcciones

Es importante construir el software incrementalmente en pasos manejables, de forma que cada paso dé lugar a pequeños problemas de integración o prueba. El resultado de cada paso es llamado “construcción”, que es una versión ejecutable del sistema. Cada construcción es entonces sometida a pruebas de integración antes de que se cree ninguna otra construcción. Para prepararse ante un fallo de una construcción se lleva un control de versiones de forma que se pueda volver atrás a una construcción anterior.

Cada iteración resultara en al menos una construcción. Sin embargo, la funcionalidad a ser implementada en una iteración determinada es a menudo demasiado compleja para ser integrada en una sola construcción. En lugar de esto, puede que se cree una secuencia de construcciones dentro de una iteración, cada una de las cuales representará un paso manejable en el que se hace un pequeño incremento al sistema.

Un plan de integración de construcciones describe la secuencia de construcciones necesarias en una iteración. Describe lo siguiente para cada construcción:

• La funcionalidad que se espera que sea implementada en dicha construcción.

• Las partes del modelo de implementación que están afectadas por la construcción.

Trabajadores de la implementación

1. Arquitecto.

2. Ingeniero de componentes.

3. Integrador de sistemas.

Arquitecto

El arquitecto es responsable de la integridad del modelo de implementación y asegura que el modelo como un todo es correcto, completo y legible.

El modelo es correcto cuando implementa la funcionalidad descrita en el modelo de diseño y en los requerimientos adicionales, y solo esa funcionalidad.

El arquitecto es responsable también de la arquitectura del modelo de implementación, es decir, de la existencia de sus partes significativas arquitectónicamente.

Un resultado importante de la implementación es la asignación de componentes ejecutables a nodos. El arquitecto es responsable de esta asignación, la cual se representa en la vista de la arquitectura del modelo de despliegue.

Ingeniero de componentes

El ingeniero de componentes define y mantiene el código fuente de uno o varios componentes, garantizando que cada componente implementa la funcionalidad correcta.

A menudo, el ingeniero de componentes también mantiene la integridad de uno o varios subsistemas de implementación. Ya que los subsistemas de implementación siguen la traza uno a uno a los subsistemas de diseño., necesita garantizar que los contenidos de los subsistemas de implementación son correctos, que sus dependencias con otros subsistemas o interfaces son correctas y que implementan correctamente las interfaces que proporciona.

Un ingeniero de componentes debería, diseñar e implementar las clases bajo su responsabilidad.

Integrador de sistemas

Entre sus responsabilidades se incluye planificar la secuencia de construcciones necesarias en cada iteración y la integración de cada construcción cuando sus partes han sido implementadas. La planificación da lugar a un plan de integración de construcciones.

Flujo de trabajo (actividades)

1. Este proceso es iniciado por el arquitecto esbozando los componentes clave en el modelo de implementación.

2. El integrador de sistemas planea las integraciones de sistema necesarias en la presente iteración como una secuencia de construcciones. Para cada construcción, el integrador de sistemas describe la funcionalidad que debería ser implementada y que partes del modelo de implementación se verán afectadas.

3. Los ingenieros de componentes implementan los requerimientos sobre los subsistemas y componentes en la construcción.

4. Los componentes son probados y pasados al integrador de sistemas para su integración. El integrador de sistemas integra los nuevos componentes en una construcción y pasa a los ingenieros de pruebas de integración para llevar a cabo pruebas de integración.

5. Los desarrolladores inician la implementación de la siguiente construcción, tomando en consideración los defectos de la construcción anterior.

Implementación de la arquitectura (por arquitecto)

Es esbozar el modelo de implementación y su arquitectura mediante:

• La identificación de componentes significativos arquitectónicamente, tales como componentes ejecutables.

• La asignación de componentes a los nodos en las configuraciones de redes relevantes.

La identificación de los subsistemas de implementación y sus interfaces es más o menos trivial y por lo tanto no se trata aquí (traza uno a uno con los subsistemas de diseño y proporcionan las mismas interfaces).

Durante esta actividad, arquitecto mantiene, refina y actualiza la descripción de la arquitectura y la vista de la arquitectura de los modelos de implementación y de despliegue.

Integrar el sistema (por integrador de sistemas)

Los objetivos de la integración del sistema son:

• Crear un plan de integración de construcciones que describa las construcciones necesarias en una iteración y los requerimientos de cada construcción.

• Integrar cada construcción antes de que sea sometida a pruebas de integración.

|  |  |
| --- | --- |
| 2 | Planificación de una construcción |

Se discute como planificar los contenidos de una construcción, independientemente de que partamos de una construcción previa o de no partamos de ninguna. Suponemos que tenemos un número de casos de uso o escenarios y otros requerimientos que han de ser implementados en la iteración actual.

Criterios para crear una construcción:

• Una construcción debería añadir funcionalidad a la construcción previa implementando casos de uso completos o escenarios de éstos.

• Una construcción no debería incluir demasiados componentes nuevos o refinados. Si no es así, puede ser muy difícil integrar la construcción y llevar a cabo las pruebas de integración.

• Una construcción debería estar basada en la construcción anterior, y debería expandirse hacia arriba y hacia los lados en la jerarquía de subsistemas. Esto significa que la construcción inicial debería empezar en capas inferiores; las construcciones subsiguientes se expanden entonces hacia arriba a las capas generales de aplicación. Esto es porque es difícil implementar componentes en las capas superiores antes de que estén colocados y funcionando los componentes necesarios en las capas inferiores.

Los resultados deberían estar recogidos en el plan de integración de la construcción y ser comunicados a los ingenieros de componentes responsables de los subsistemas y componentes de implementación afectados. Los ingenieros de componentes pueden entonces empezar a implementar los requerimientos de los subsistemas y componentes de implementación en la construcción actual y llevar a cabo las pruebas individuales de cada unidad.

Integración de una construcción

Esto se hace recopilando las versiones correctas de los subsistemas de implementación y de los componentes compilándolos y enlazándolos para generar una construcción.

Implementar un subsistema (por ing. de componentes)

El propósito de implementar un subsistema es el de asegurar que un subsistema cumple su papel en cada construcción, tal y como se especifica en el plan de integración de la construcción. Esto quiere decir que se asegura que los requerimientos implementados en la construcción y aquellos que afectan al subsistema son implementados correctamente por componentes o por otros subsistemas (recursivamente) dentro del subsistema.

Implementar una clase (por ing. de componentes)

El propósito de la implementación de una clase es implementar una clase de diseño en un componente fichero.

Esbozo de los componentes fichero

Es normal implementar varias clases de diseño en un mismo componente fichero. Sin embargo, el tipo de modularización y convenciones de los lenguajes de programación restringen la forma en que los componentes fichero son esbozados.

Generación de código a partir de una clase de diseño

Solo se genera la signatura (la firma) de las operaciones (métodos) las cuales todavía deben ser implementadas.

Implementación de operaciones

Cada operación definida por la clase de diseño ha de ser implementada, a no ser que sea virtual (o abstracta) y ésta sea implementada por descendientes de la clase.

La implementación de una operación incluye la elección de un algoritmo y unas estructuras de datos apropiadas, y la codificación de las acciones requeridas por el algoritmo.

Los componentes han de proporcionar las interfaces apropiadas

El componente resultante debería proporcionar las mismas interfaces que las clases de diseño que éste implementa.

Realizar prueba de unidad (por ing. de componentes)

Probar los componentes implementados como unidades individuales. Dos tipos de prueba de unidad:

• Prueba de especificación (“prueba de caja negra”): se verifica el comportamiento de la unidad observable externamente.

Se realiza para verificar el comportamiento del componente sin tener en cuenta cómo se implementa dicho comportamiento en el componente. Tienen en cuenta la salida que el componente devolverá cuando se le da una determinada entrada en un determinado estado.

• Prueba de estructura (“prueba de caja blanca”): verifica la implementación interna de la unidad. Se fija en el código.

Se realizan para verificar que un componente funciona internamente como se quería. El ingeniero de componentes debería también de asegurarse de probar todo el código durante las pruebas de estructura, lo que quiere decir que cada sentencia ha de ser ejecutada al menos una vez. Debería también asegurarse de probar los caminos más importantes en el código.

|  |  |
| --- | --- |
| 3 | WORKFLOW DE PRUEBA |

Introducción

En el flujo de trabajo de la prueba verificamos el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema a ser entregadas a los clientes.

Propósitos de la prueba

• Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de integración y las pruebas de sistema. Las pruebas de integración son necesarias para cada construcción dentro de la iteración, mientras que las pruebas de sistema son necesarias sólo al final de la iteración.

• Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar.

• Realizar las diferentes pruebas y manejar los resultados de cada prueba simultáneamente.

El papel de la prueba en el ciclo de vida del software

La realización de pruebas se centra en las fases de elaboración, cuando se prueba la línea base ejecutable de la arquitectura, y de construcción cuando el grueso del sistema está implementado. Durante la fase de transición el centro se desplaza hacia la corrección de defectos durante los primeros usos y las pruebas de regresión.

Artefactos de la prueba

1. Modelo de pruebas.

2. Caso de prueba.

3. Procedimiento de prueba.

4. Plan de prueba.

5. Defecto.

6. Evaluación de prueba.

Modelo de pruebas

El modelo de pruebas describe principalmente cómo se prueban los componentes ejecutables en el modelo de implementación con pruebas de integración y de sistema. Puede describir también como han de ser probados aspectos específicos del sistema; por ejemplo; si la interfaz de usuario es utilizable y consistente o si el manual de usuario del sistema cumple su cometido. El modelo de pruebas es una colección de casos de prueba, procedimientos de prueba y componentes de prueba.

Caso de prueba

Un caso de prueba especifica una forma de probar el sistema, incluyendo la entrada o resultado con la que se ha de probar y las condiciones bajo las que ha de probarse.

Casos de prueba comunes:

• Un caso de prueba que especifica cómo probar un caso de uso o un escenario específico de un caso de uso (verificación del resultado de la interacción ente actores y el sistema, si se cumplen las precondiciones y postcondiciones y que se siguen la secuencia de acciones especificadas en el caso de uso). Prueba de sistema como “caja negra”, es decir, una prueba del comportamiento observable del sistema.

• Un caso de prueba que especifica cómo probar una realización de caso de uso de diseño o un escenario específico de la realización (verificación de la interacción entre componentes que implementan el caso de uso). Prueba de sistema como “caja blanca”, es decir, una prueba de la interacción interna entre los componentes del sistema.

Se pueden especificar otros casos de prueba para probar el sistema como un todo. Por ejemplo:

• Pruebas de instalación: verifican que el sistema puede ser instalado en la plataforma del cliente y que funcionará correctamente cuando sea instalado.

• Pruebas de configuración: verifican que el sistema funciona correctamente en diferentes configuraciones.

• Pruebas negativas: intentan provocar que el sistema falle para poder así revelar sus debilidades. Consiste en utilizar el sistema en formas para los que no sido diseñado (“usar mal el sistema a propósito”).

• Pruebas de tensión o de estrés: identifican problemas con el sistema cuando hay recursos insuficientes o cuando hay competencia por recursos.

Procedimiento de prueba

Un procedimiento de prueba especifica cómo realizar uno a varios casos de pruebas o partes de éstos.

Componente de prueba

Un componente de prueba automatiza uno o varios procedimientos de prueba o partes de ellos.

Se utilizan para probar los componentes en el modelo de implementación, proporcionando entradas de prueba, controlando y monitorizando la ejecución de los componentes a probar y, posiblemente, informando de los resultados de las pruebas.

Plan de prueba

El plan de prueba describe las estrategias, recursos y planificación de la prueba. La estrategia de prueba incluye la definición del tipo de pruebas a realizar para cada iteración y sus objetivos.

Defecto

Un defecto es una anomalía en el sistema.

Evaluación de prueba

Una evaluación de prueba es una evaluación de los resultados de los esfuerzos de prueba.

Trabajadores de la prueba

1. Diseñador de pruebas (ingeniero de prueba).

2. Ingeniero de componentes.

3. Ingeniero de pruebas de integración.

4. Ingeniero de pruebas de sistema.

Diseñador de pruebas

Un diseñador de pruebas es responsable de la integridad del modelo de pruebas, asegurando que el modelo cumple con su propósito. También planean las pruebas, lo que significa que deciden los objetivos apropiados y la planificación de las pruebas. Además, seleccionan y describen los casos de prueba y los procedimientos de prueba correspondientes que se necesitan, y son responsables de la evaluación de pruebas de integración y de sistema cuando éstas se ejecutan.

Ingeniero de componentes

Los ingenieros de componentes son responsables de los componentes de prueba que automatizan algunos procedimientos de prueba.

Ingeniero de pruebas de integración

Los ingenieros de pruebas de integración son los responsables de realizar las pruebas de integración que se necesitan para cada construcción producida en el flujo de trabajo de la implementación. Las pruebas de integración se realizan para verificar que los componentes integrados en una construcción funcionan correctamente juntos.

El ingeniero de pruebas de integración se encarga de documentar los defectos en los resultados de las pruebas de integración.

Ingeniero de pruebas de sistema

Un ingeniero de pruebas de sistema es responsable de realizar las pruebas de sistema necesarias sobre una construcción que muestra el resultado (ejecutable) de una iteración completa.

Las pruebas de sistema se llevan a cabo principalmente para verificar las interacciones entre los actores y el sistema, por eso suelen derivarse de los casos de prueba que especifican como probar los casos de uso.

El ingeniero de pruebas de sistema se encarga de documentar los defectos en los resultados de las pruebas de sistema.

Flujo de trabajo (actividades)

1. Los ingenieros de pruebas determinar el objetivo de la prueba a realizar planificando el esfuerzo de prueba a realizar en cada iteración

2. Describen entonces los casos de prueba necesarios y los procedimientos de prueba correspondientes para llevar a cabo las pruebas.

3. Si es posible, los ingenieros de componentes crean a continuación los componentes de prueba para automatizar algunos procedimientos de prueba.

4. Con estos casos, procedimientos y componentes de prueba como entrada, los ingenieros de pruebas de integración y de sistema prueban cada construcción y detectan cualquier defecto que encuentren en ellos.

5. Los defectos sirven como retroalimentación tanto para otros flujos de trabajo, como el diseño y el de implementación, como para los ingenieros de pruebas para que lleven a cabo una evaluación sistemática de los resultados de las pruebas.

Planificar prueba (ing. de pruebas)

El propósito de la planificación de prueba es planificar los esfuerzos de prueba en una iteración llevando a cabo las siguientes tareas:

• Describiendo una estrategia de prueba.

• Estimando los requerimientos para el esfuerzo de la prueba (ejemplo: recursos humanos y sistemas necesarios).

• Planificando el esfuerzo de la prueba.

En general, se crea el plan de prueba estableciendo las estrategias y recursos.

Diseñar prueba (ing. de pruebas)

Propósitos de diseñar las pruebas:

• Identificar y describir los casos de prueba para cada construcción.

• Identificar y estructurar los procedimientos de prueba especificando cómo realizar los casos de prueba.

Diseño de los casos de prueba de integración

La mayoría de los casos de prueba de integración pueden ser derivados de las realizaciones de casos de uso de diseño, ya que las realizaciones de casos de uso describen cómo interaccionan las clases y los objetos, y por tanto cómo interaccionan los componentes. Se consideran como entrada los diagramas de interacción de las realizaciones de casos de uso.

Diseño de los casos de prueba de sistema

Las pruebas de sistema se usan para probar que el sistema funciona correctamente como un todo. Cada prueba de sistema prueba principalmente combinaciones de casos de uso instanciados bajo condiciones diferentes (diferentes configuraciones de hardware, diferentes cargas del sistema, diferentes números de actores y diferentes tamaños de la base de datos).

Diseño de los casos de prueba de regresión

Algunos casos de prueba de construcciones anteriores pueden ser usados para pruebas de regresión en construcciones subsiguientes, aunque no todos los casos de prueba son adecuados para pruebas de regresión.

Identificación y estructuración de los procedimientos de prueba

Se intenta reutilizar procedimientos de prueba existentes tanto como sea posible. Los diseñadores de pruebas también intentan crear procedimientos de prueba que puedan ser reutilizados en varios casos de prueba.

Implementar prueba (ing. de componentes)

El propósito de la implementación de las pruebas es automatizar los procedimientos de prueba creando componentes de prueba si esto es posible, ya que no todos los procedimientos de prueba pueden ser automatizados.

Realizar pruebas de integración (ing. de pruebas de integración)

En esta actividad se realizan las pruebas de integración necesarias para cada una de las construcciones creadas en una iteración y se recopilan los resultados de las pruebas.

Realizar prueba de sistema (ing. de pruebas de sistema)

El propósito de la prueba de sistema es el realizar las pruebas de sistemaϑ.

La prueba de sistema puede empezar cuando las pruebas de integración indican que el sistema satisface los objetivos de calidad de integración fijados en el plan de pruebas de la iteración actual.

Evaluar la prueba (ing. de pruebas)

El propósito de la evaluación de la prueba es el evaluar los esfuerzos de prueba de una iteración.

Los diseñadores de pruebas evalúan los resultados de la prueba comparando los resultados obtenidos con los objetivos esbozados en el plan de prueba. Éstos preparan métricas que les permiten determinar el nivel de calidad del software y qué cantidad de pruebas es necesario hacer.

|  |  |
| --- | --- |
| 4 | PRUEBA |

Introducción

Las pruebas intentan demostrar que un programa hace lo que se intenta que haga, así como descubrir defectos en el programa antes de usarlo.

El proceso de prueba tiene dos metas distintas:

1. Demostrar al desarrollador y al cliente que el software cumple con los requerimientos.

2. Encontrar situaciones donde el comportamiento del software sea incorrecto, indeseable o no esté de acuerdo con su especificación.

La primera meta conduce a la prueba de validación; en ella, se espera que el sistema se desempeñe de manera correcta mediante un conjunto dado de casos de prueba, que refleje el uso previsto del sistema. La segunda meta se orienta a pruebas de defectos, donde los casos de prueba se diseñan para presentar los defectos.

Las pruebas no pueden demostrar que el software está exento de defectos o que se comportará como se especifica en cualquier circunstancia.

Las pruebas pueden mostrar sólo la presencia de errores, más no su ausencia.

El propósito de la prueba es ENCONTRAR FALLAS. El testing exitoso es el que encuentro defectos.

Las pruebas pueden dividirse en:

• Validación: ¿construimos el producto correcto?

• Verificación: ¿construimos bien el producto?

La finalidad de la verificación es comprobar que el software cumpla con su funcionalidad y con los requerimientos no funcionales establecidos. Sin embargo, la validación es un proceso más general. La meta de la validación es garantizar que el software cumpla las expectativas del cliente. La validación es esencial pues, las especificaciones de requerimientos no siempre reflejan los deseos o las necesidades reales de los clientes y usuarios del sistema.

El objetivo final de los procesos de verificación y validación es establecer confianza de que el sistema de software es “adecuado”.

Prueba e inspección

Al igual que las pruebas de software, el proceso de verificación y validación implicaría inspecciones y revisiones de software. Estas últimas analizan y comprueban los requerimientos del sistema, los modelos de diseño, el código fuente, y las pruebas propuestas para el sistema. Éstas son las llamadas técnicas “estáticas”.

Técnicas “estáticas”: donde no es necesario ejecutar el software para verificarlo.

Las inspecciones se enfocan principalmente en el código del sistema.

Ventajas de la inspección sobre las pruebas

1. Durante las pruebas, los errores pueden enmascarar (ocultar) otras fallas. Cuando un error lleva a salidas inesperadas, nunca se podrá asegurar si las anomalías de salida posteriores se deben a un nuevo error o son defectos colaterales del error original.

2. Las versiones incompletas de un sistema se pueden inspeccionar sin costos adicionales.

3. Una inspección puede considerar también atributos más amplios de calidad de un programa, como el cumplimiento con estándares, la portabilidad y la mantenibilidad. Pueden buscarse inferencias, algoritmos inadecuados y estilos de programación imitados que hagan al sistema difícil de mantener y actualizar.

Las inspecciones no sustituyen las pruebas del software, ya que no son eficaces para descubrir defectos que surjan por interacciones inesperadas entre diferentes partes de un programa, problemas de temporización o dificultades con el rendimiento del sistema.

Casos de prueba: son especificaciones de las entradas a las pruebas y la salida esperada del sistema (los resultados de la prueba).

Es posible automatizar la ejecución de pruebas. Los resultados provistos se comparan automáticamente con los resultados establecidos, de manera que no haya necesidad de que un individuo busque errores y anomalías al correr las pruebas.

Etapas de prueba

1. Pruebas de desarrollo: donde el sistema se pone a prueba durante el proceso para descubrir errores (bugs) y defectos. Intervienen diseñadores y programadores del sistema.

2. Prueba de versión: donde un equipo de prueba por separado experimenta una versión completa del sistema, antes de presentarlo a los usuarios.

3. Pruebas de usuario: donde los usuarios reales o potenciales del sistema prueban el sistema en su propio entorno.

Pruebas de desarrollo

Las pruebas de desarrollo incluyen todas las actividades de prueba que realiza el equipo que elabora el sistema.

Niveles de prueba

• Pruebas de unidad: donde se ponen a prueba unidades de programa, clases de objetos, bloques y paquetes. Las pruebas de unidad deben enfocarse en comprobar la funcionalidad de objetos y métodos.

• Pruebas de componente (de integración): se prueban que las unidades trabajan correctamente juntas. Deben enfocarse en probar interfaces de componentes.

• Pruebas de sistema: se prueba el sistema completo.

Pruebas de unidad

Es el nivel de prueba más bajo y normalmente lo hace el mismo desarrollador. Involucra clases, bloques, paquetes de servicio.

• Prueba de caja negra (de especificación): verifican el comportamiento de la interfaz de la unidad, o sea lo que hace la unidad sin importar como lo hace. Además de verificar si se producen salidas, hay que verificar si la salida es la correcta.

• Prueba de caja blanca (de estructura): se verifica si la estructura interna es la correcta. Todos los caminos posibles planteados en el código deben ser contemplados y ejecutados. Por lo general esta prueba se realiza al último.

• Prueba basada en estados: prueba la interacción entre las operaciones de una clase, monitoreando los cambios que tienen lugar en los atributos de los objetos. Es importante basarse en diagramas de transición estados, así, al menos cada estado es visitado por lo menos una vez y cada transición es atravesada al menos una vez.

Cuando se pone a prueba las clases de los objetos, hay que diseñar pruebas para brindar cobertura a todas las características del objeto:

• Probar todas las operaciones asociadas al objeto.

• Establecer y verificar el valor de todos los atributos relacionados con el objeto.

• Poner el objeto en todos los estados posibles.

La generalización o herencia provoca que sea más difícil la prueba de las clases de objetos. Por consiguiente, tienen que poner a prueba la operación heredada en todos los contextos en que se utilice.

Siempre que sea posible, se deben automatizar las pruebas de unidad (por ejemplo, JUnit).

Pruebas de componentes (de integración)

Una vez que las unidades han sido certificadas en las pruebas de unidad, estas unidades deberían integrarse en unidades más grandes y finalmente al sistema.

El propósito es determinar si las distintas unidades que han sido desarrolladas trabajan apropiadamente juntas.

En general, los componentes de software son componentes compuestos constituidos por varios objetos en interacción. El acceso a la funcionalidad de dichos objetos es a través de la interfaz de componente definida. Por consiguiente, la prueba de componentes compuestos tiene que enfocarse en mostrar que la interfaz de componente se comporta según su especificación.

Pruebas del sistema

Una vez que se han probado todos los casos de uso por separado se probará el sistema completo.

Las pruebas de sistema durante el desarrollo incluyen la integración de componentes para crear una versión del sistema y, luego, poner a prueba el sistema integrado. Las pruebas de sistema demuestran que los componentes son compatibles, que interactúan correctamente y que transfieren los datos correctos en el momento adecuado a través de sus interfaces.

Cuando se integran componentes para crear un sistema, se obtiene un comportamiento emergente. Esto significa que algunos elementos de funcionalidad del sistema sólo se hacen evidentes cuando se reúnen los componentes. Sin embargo, algunas veces, el comportamiento emergente no está planeado ni se desea. Hay que desarrollar pruebas que demuestren que el sistema sólo hace lo que se supone que debe hacer.

Por lo tanto, las pruebas del sistema deben enfocarse en poner a prueba las interacciones entre los componentes y los objetos que constituyen el sistema.

Pruebas de versión

Las pruebas de versión son el proceso de poner a prueba una versión particular de un sistema que se pretende usar fuera del equipo de desarrollo.

Existen dos distinciones importantes entre las pruebas de versión y las pruebas de sistema durante el desarrollo:

1. Un equipo independiente que no intervino en el desarrollo del sistema debe ser el responsable de las pruebas de versión.

2. Las pruebas del sistema por parte del equipo de desarrollo deben enfocarse en el descubrimiento de bugs en el sistema.

La principal meta del proceso de pruebas de versión es convencer al proveedor del sistema de que éste es suficientemente apto para su uso. Por ello, las pruebas de versión deben mostrar que el sistema entrega su funcionalidad, rendimiento y confiabilidad especificados, y que no falla durante el uso normal.

Las pruebas de versión, por lo regular, son un proceso de prueba de caja negra, donde las pruebas se derivan a partir de la especificación del sistema.

Pruebas basadas en requerimientos

Los requerimientos tienen que escribirse de forma que pueda diseñarse una prueba para dicho requerimiento. Luego, un examinador comprueba que el requerimiento se cumpla. Las pruebas basadas en requerimientos son un enfoque sistemático al diseño de casos de prueba, donde se considera cada requerimiento y se deriva un conjunto de pruebas para éste.

Pruebas de escenario

Es un enfoque donde se crean escenarios típicos de uso y se les utiliza en el desarrollo de casos de prueba para el sistema. Un escenario es una historia que describe una forma en puede usarse el sistema.

Pruebas de rendimiento

Una vez integrado completamente el sistema, es posible probar propiedades emergentes, como el rendimiento y la confiabilidad. Las pruebas de rendimiento deben diseñarse para garantizar que el sistema procese su carga pretendida. Generalmente, esto implica efectuar una serie de pruebas donde se aumenta la carga, hasta que el rendimiento del sistema se vuelve inaceptable.

La prueba de rendimiento significa estresar el sistema al hacer demandas que están fuera de los límites de diseño del software. Esto se conoce como “prueba de esfuerzo”. Este tipo de prueba tiene dos funciones:

1. Prueba el comportamiento de falla del sistema.

2. Fuerza al sistema y puede hacer que salgan a luz defectos que no se descubrirían normalmente.

Las pruebas de esfuerzo son particularmente relevantes para sistemas distribuidos basados en redes de procesadores.

Pruebas de usuario

Las pruebas de usuario o del cliente son una etapa en el proceso de pruebas donde los usuarios o clientes proporcionan entrada y asesoría sobre las pruebas del sistema. Las pruebas de usuario son esenciales, aun cuando se hayan realizado pruebas abarcadoras de sistema y de versión. La razón de esto es que la influencia del entorno de trabajo del usuario tiene un gran efecto sobre la fiabilidad, el rendimiento, el uso y la robustez de un sistema.

Es casi imposible que un desarrollador de sistema replique el entorno de trabajo del sistema, pues las pruebas en el entorno del desarrollador son forzosamente artificiales.

Hay tres tipos de pruebas de usuario:

• Pruebas alfa: los usuarios del software trabajan con el equipo de diseño para probar el software en el sitio del desarrollador.

• Pruebas beta: una versión del software se pone a disposición de los usuarios, para permitirles experimentar y descubrir problemas que encuentran con los desarrolladores del sistema.

• Pruebas de aceptación: donde los clientes prueban un sistema para decidir si está o no listo para ser aceptado por los desarrolladores del sistema y desplegado en el entorno del cliente.

Tipos de prueba

1. Tests de operación: el sistema es probado en operación normal. Mide la confiabilidad del sistema y se pueden obtener mediciones estadísticas.

2. Tests de escala completa: ejecutamos el sistema al máximo, todos los parámetros se enfocan en valores máximos, todos los equipos conectados, usados por muchos usuarios ejecutando casos de uso simultáneamente.

3. Tests de performance o capacidad: el objetivo es medir la capacidad de procesamiento del sistema. Los valores obtenidos son comparados con los requeridos.

4. Tests de sobrecarga (de estrés): determinan cómo se comporta el sistema cuando es sobrecargado. No se puede esperar que supere esta prueba, pero sí que no se venga abajo, que no ocurra una catástrofe. Cuántas veces se cayó el sistema es una medida interesante. Es importante ver como “queda” el sistema.

5. Tests de requerimientos: estos tests se mapean (rastrean) directamente desde la especificación de requerimientos.

6. Tests de documentación: se prueba la documentación del sistema.

7. Tests de aceptación: ejecutado por la organización que solicita el sistema. Genera la aceptación o no del sistema.

8. Tests de instalación: se verifica que el sistema pueda ser instalado en la plataforma del cliente y que el sistema funcionará correctamente cuando sea instalado.

9. Tests de configuración: verifica que el sistema funciona correctamente en diferentes configuraciones.

10. Tests negativo: el sistema se usa en forma incorrecta intencionalmente para probar casos especiales.

11. Test de procesos de negocio

12. Test de seguridad

Desarrollo dirigido por pruebas (TDD)

El desarrollo dirigido por pruebas (TDD, Test-Driven Development) consiste en desarrollar el código incrementalmente, junto con una prueba para ese incremento. No se avanza hacia el siguiente incremento sino hasta que el código diseñado para la prueba.