Tema 15 – Implementación y prueba

Ingeniería del Software

Rubén Fuentes Fernández
Dep. Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense Madrid

Trabajando con Antonio Navarro, Juan Pavón y Pablo Gervás

Contenidos

- Introducción
  - Desarrollo
  - Implementación
  - Prueba
- Artefactos
- Tareas y planificación
  - Niveles de prueba para la implementación
  - Tipos de prueba para la implementación
  - Automatización de pruebas

Elementos en el desarrollo de sistemas

- El desarrollo de un sistema requiere una variedad de elementos (o artefactos del desarrollo).
  - Ej. documentos, modelos, pruebas y componentes.
- Un sistema se implementa en términos de *artefactos* (UML).
  - Ej. ficheros de código fuente, binarios y ejecutables, y *scripts*.
- Los elementos específicos a considerar en el desarrollo varían según la etapa del proyecto.

Desarrollo incremental

- El *desarrollo incremental* aboga por construir los sistemas incrementalmente en pasos manejables.
  - Cada paso da lugar a pequeños problemas de desarrollo, integración y prueba.
- El resultado de cada paso se llama *construcción*.
  - Amplía una parte específica del sistema sobre otra *construcción*, salvo en la *construcción* inicial.
  - Incluye todos los elementos relacionados con la ampliación / modificación.
    - En la implementación, una *construcción* se centra en una versión ejecutable del sistema.
  - Cada *construcción* es sometida a pruebas de integración antes de crear ninguna otra.
- El *desarrollo incremental* es la aproximación de la mayor parte de los procesos de desarrollo modernos.

Ventajas del desarrollo incremental

- Versión ejecutable del sistema muy pronto.
  - Pruebas de integración pronto.
  - Posibilidad de demostrar el sistema pronto a otras personas.
- Más fácil localizar defectos durante las pruebas.
  - Sólo se ha añadido una parte pequeña desde la construcción anterior.
  - Presumiblemente, cualquier error estará en esa parte.
- Las pruebas de integración son más minuciosas.
  - Son sobre una parte pequeña.

Desarrollo y pruebas

- Todos los artefactos que aparecen en el desarrollo son susceptibles de prueba.
  - El desarrollo comienza con la especificación de requisitos.
  - Continúa con los modelos de análisis y diseño.
  - A partir de los modelos se desarrolla código.
- Las pruebas dependen del tipo de artefacto.
  - Ej. revisiones técnicas formales para especificaciones y modelos, y pruebas automatizadas de accesibilidad en el código.
- *Aquí nos centramos en la fase de implementación, y por tanto en el código.*
  - *El caso de los otros artefactos del desarrollo ya se ha comentado en temas anteriores, pero se revisará aquí brevemente.*

Objetivos de la implementación

- Planificar las integraciones de sistema necesarias en cada iteración.
  - Común con otras fases del desarrollo.
- Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue.
- Implementar las clases y subsistemas encontrados durante el diseño.
- Integrar los componentes individuales (compilándolos y enlazándolos en uno o más ejecutables).

Objetivos del proceso de prueba

- El objetivo de las pruebas es encontrar el mayor número posible de errores, con una cantidad razonable de esfuerzo aplicado sobre un lapso de tiempo realista.
  - Cuando se aplica a un sistema, se trata de ejecutarlo buscando errores.
- Un buen *caso de prueba* es aquel que tiene una probabilidad muy alta de encontrar un error que no se hubiera descubierto todavía.
- Una *prueba* satisfactoria es la que encuentra un error.

ARTEFACTOS

Implementación: artefactos

- Modelo de implementación
- *Artefacto*
- Subsistema de implementación
- Interfaz
- Descripción de la arquitectura
  - Vista del modelo de implementación
- Plan de integración de construcciones

Ver el tema de UML para una descripción detallada de estos conceptos.

Modelo de implementación

- El *modelo de implementación* describe los siguientes aspectos del sistema:
  - Cómo los elementos del modelo de diseño se implementan en términos de *artefactos*.
    - En el modelo de diseño tenemos fundamentalmente clases.
    - Ejemplos de *artefactos* son los ficheros de código fuente, los ejecutables y los de configuración, y los *scripts*.
  - Cómo se organizan los *artefactos*.
    - De acuerdo con los mecanismos de estructuración y modularización del entorno de implementación y los lenguajes de programación utilizados.
  - Cómo dependen los *artefactos* unos de otros.

Artefacto

- Empaquetamiento físico de los elementos de un modelo.
- Cada uno puede implementar varios elementos de diseño dependiendo del lenguaje que se utilice.
- Proporcionan las mismas interfaces que los elementos de diseño que implementan.
- Tienen:
  - Relaciones de traza con los elementos del diseño que implementan.
  - Dependencias de compilación entre ellos.
    - Unos deben haberse compilado antes para poder compilar otros.

Ejemplos de artefacto

- «executable»
  - Programa que puede ser ejecutado en un nodo
- «file»
  - Fichero que contiene código fuente o datos
- «library»
  - Librería estática o dinámica
- «table»
  - Tabla de base de datos
- «document»
  - Documento

Subsistema de implementación

- Forma de organizar los elementos del modelo de implementación en partes más manejables.
- Un subsistema puede estar formado por:
  - *artefactos*
  - componentes
  - interfaces
  - otros subsistemas (recursivamente)
- Corresponde a distintos conceptos en cada lenguaje de programación.

Interfaz

- Una *interfaz* es una descripción de las responsabilidades de un componente.
  - Generalmente en términos de operaciones
    - Descritas por sus signaturas, pre y post-condiciones, y otras restricciones.
  - Aunque también se pueden incluir miembros de datos e interacciones.
- Un *componente* que implementa una *interfaz* debe implementar correctamente todas las responsabilidades de la *interfaz*.
  - Generalmente proporcionar una implementación de sus operaciones.
- Un *subsistema* que implementa una *interfaz* debe contener *componentes* u otros *subsistemas* que proporcionen dicha interfaz.

Descripción de la arquitectura

- La *descripción de la arquitectura* de un sistema es la descomposición de su modelo de implementación en subsistemas, sus interfaces y las dependencias entre ellos.
  - Cómo vienen dados por los equivalentes del modelo de diseño, suele ser innecesario representarlos.
- Identifica los componentes clave del sistema.
  - Son los que tienen traza a clases de diseño significativas arquitectónicamente y los ejecutables.

Plan de integración de construcciones

- El *plan de integración de construcciones* describe la secuencia de *construcciones* necesarias en una iteración.
- Para cada *construcción* indica:
  - La funcionalidad que se espera que se implemente en esa *construcción*.
    - Lista de casos de uso o escenarios, o partes de ellos
    - También puede incluir requisitos adicionales
  - Las partes del modelo de implementación afectadas por la *construcción*.
    - Lista de los subsistemas, componentes y artefactos necesarios para implementar esa funcionalidad

Uso de *stubs*

- Un *stub* es un componente con una implementación esquelética o de propósito especial que puede ser utilizada para desarrollar o probar otro componente que depende de él.
- Su uso es común en desarrollos incrementales a la hora de implementar.
  - Se usan para minimizar el número de componentes nuevos necesarios en cada nueva versión (intermedia) del sistema.

Prueba: artefactos

- Caso de prueba
- Procedimiento de prueba
- Componente de prueba
- Modelo de pruebas
- Plan de prueba
- Resultados de las pruebas
- Modelo de carga de trabajo

Caso de prueba

- Un *caso de prueba* es la especificación del conjunto de los datos de prueba, condiciones de ejecución, y resultados esperados para probar un objetivo concreto.
- Se pueden derivar de los casos de uso, de los documentos de diseño o del código.
- Cada caso de prueba puede ser implementado por uno o más procedimientos de prueba.

Procedimiento de prueba

- Un *procedimiento de prueba* especifica cómo realizar uno o varios casos de prueba o partes de éstos.
  - Describe el conjunto de instrucciones detalladas para la puesta en marcha, la ejecución y la evaluación de los resultados de una prueba.
- El procedimiento de prueba puede recoger:
  - Instrucciones sobre cómo ha de realizarse un caso de prueba manualmente.
  - Una especificación de cómo ejecutar componentes de prueba.

Componente de prueba

- Un *componente de prueba* es aquel que contiene código que automatiza la ejecución de algún procedimiento de prueba.
  - *stubs* y *drivers* (o *test drivers*)
- Se pueden generar:
  - Paquetes y clases de prueba
  - Subsistemas y *artefactos* de prueba

Modelo de pruebas

- El *modelo de pruebas* es una vista de los modelos de diseño e implementación que incluye las pruebas.

Plan de prueba

- El *plan de prueba* es una descripción del propósito, objetivos, estrategias, recursos y planificación de las pruebas.
- Incluye:
  - La definición del tipo de pruebas a realizar para cada iteración y sus objetivos
  - El nivel de cobertura de prueba
  - El nivel de código necesario
  - El porcentaje de pruebas que deberían ejecutarse con un resultado específico

Resultados de las pruebas

- Los *resultados de las pruebas* son los datos capturados durante la ejecución de las pruebas.
- La evaluación de los resultados de los esfuerzos de prueba incluye:
  - cobertura del caso de prueba
  - cobertura de código
  - estado de los defectos

Modelo de carga de trabajo

- El *modelo de carga de trabajo* se utiliza para probar rendimientos.
- Identifica y define:
  - Los valores de las variables utilizadas para simular en pruebas de rendimiento
  - Las características de los actores y las funciones del sistema
  - La carga y volumen de los anteriores
  - ....

TAREAS Y PLANIFICACIÓN

Implementación y fases: largo plazo

- Elaboración
  - Crear la línea base ejecutable de la arquitectura.
- Construcción
  - Realizar el grueso de la implementación.
  - Se desarrolla la arquitectura hasta el sistema final.
- Transición
  - Tratar defectos tardíos encontrados en distribuciones beta.

Nótese que hablamos de fases del Proceso Unificado, pero son asimilables a las de otros procesos de desarrollo.

Pruebas y fases: largo plazo

- Inicio
  - Establecer la planificación inicial de las pruebas.
- Elaboración
  - Se prueba la línea base de la arquitectura.
    - Es el núcleo estable del sistema.
  - Cada construcción se somete a pruebas de integración.
- Construcción
  - Se prueba todo el sistema.
- Transición
  - Corrección de defectos y pruebas de regresión.

Implementación: corto plazo

- Se planifican incrementos de la implementación.
  - Cada iteración resultará en al menos una *construcción*.
  - Puede haber una secuencia de *construcciones* dentro de una iteración.
- Tareas de una iteración:
  - Esbozar los componentes clave a abordar del *modelo de implementación*.
  - Planear la secuencia de *construcciones*.
  - Para cada *construcción*:
    - Describir la funcionalidad a implementar y las partes de modelo afectadas.
    - Implementar los requisitos como subsistemas y artefactos.
    - Probar componentes individualmente → Prueba
    - Integrar nuevos artefactos en una construcción.
    - Probar la integración → Prueba

Prueba: corto plazo

- Tareas de una iteración:
  - Planificar el esfuerzo de prueba en cada iteración.
  - Describir los casos de prueba y los procedimientos necesarios.
  - Crear los componentes de prueba.
  - Probar cada construcción.
  - Detectar defectos.

Criterios de planificación de construcción

- Añadir funcionalidad a la construcción previa.
  - Considerar casos de uso o escenarios completos.
- No debe incluir demasiados componentes nuevos o refinados.
  - Algunos componentes pueden implementarse como *stubs*.
- Cada construcción está basada en la construcción anterior y debe expandirse hacia arriba y hacia los lados en la jerarquía de subsistemas.
  - Empezar en las capas inferiores porque es difícil implementar las demás sin esas.

Integrar una construcción

- Recopilar las versiones correctas de los subsistemas que la compongan.
- Compilarlos y enlazarlos para generar la *construcción*.
- Compilación de abajo arriba en la jerarquía de capas.
  - Es decir, siguiendo las dependencias de compilación.

Efecto sobre gestión de configuración

- El control de versiones de los artefactos permite prepararse ante el fallo de una *construcción*.
  - La *construcción* falla si no pasa las pruebas.
  - Si hay fallo, se puede volver a una *construcción* anterior.

Planificar las pruebas

- Las pruebas deben planificarse mucho antes de empezar a probar.
  - Incluso antes de empezar a generar código.
- Probar de menor a mayor.
  - Primero componentes individuales, luego irlos integrando.

Evolución de un modelo de pruebas

- Se crean nuevos casos de prueba para cada nueva construcción.
- Los casos de prueba de las primeras construcciones se pueden usar luego como casos de prueba de regresión para las siguientes construcciones.
- Se eliminan casos de prueba obsoletos.

Prueba y depuración

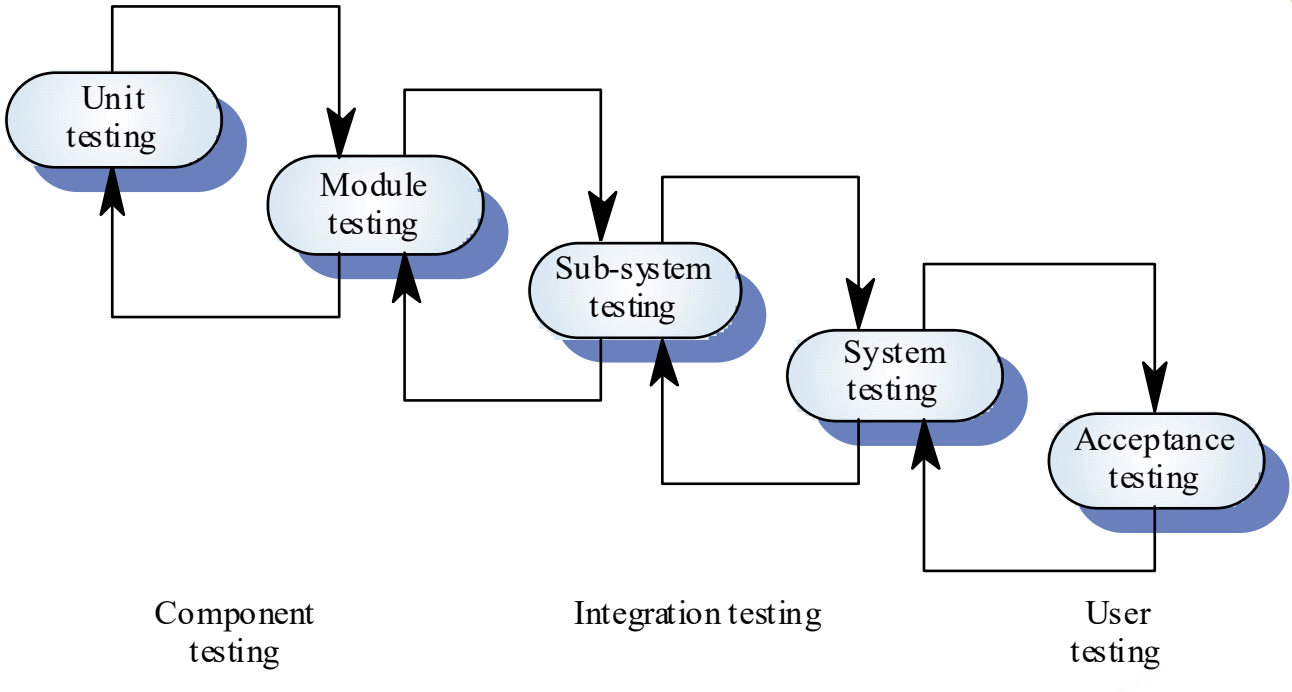
- La *prueba* y la *depuración* son procesos distintos.
  - La prueba se refiere a la confirmación de la presencia de errores.
  - La depuración se refiere a la localización y reparación de estos errores.
- La depuración involucra la formulación de una hipótesis acerca del comportamiento del programa y comprobar la hipótesis para encontrar los errores en el sistema.

Pruebas para modelos

- Existen dos aspectos clave que se han de garantizar acerca de los modelos:
  - La consistencia entre los distintos modelos que describen un sistema.
  - Su modificación y comunicación apropiada por sus responsables.
- Estos aspectos se suelen tratar a través de la gestión de configuración.
  - Los modelos son Elementos de Configuración del SW (ECSs).
  - Están sujetos a Revisiones Técnicas Formales (RTFs).
  - Cuanto antes se descubran errores en el desarrollo, menos costosa será su reparación.
- Al margen de estos aspectos, no existen mecanismos generales para revisar modelos.
  - Pueden variar en función de la notación.
  - Los objetivos pueden ser muy diversos.

FASES DE PRUEBAS

Fases de pruebas



Fases de pruebas

- Pruebas de unidades
  - Prueba de componentes individuales
- Prueba de módulos
  - Prueba de conjuntos de componentes dependientes
- Prueba de subsistemas
  - Prueba de colecciones de módulos integrados en subsistemas
- Prueba del sistema
  - Prueba del sistema completo antes de su entrega
- Prueba de aceptación
  - Prueba de los usuarios para verificar que el sistema cumple con los requisitos
  - Llamada en ocasiones prueba alfa

Pruebas de unidad

- En el contexto de la OO la unidad más pequeña es la clase.
- De cada clase hay que probar cada una de sus operaciones.
  - Debido al polimorfismo, las operaciones pueden variar en el contexto de una jerarquía de clases.
- Para cada operación hay que comprobar:
  - Interfaz de la operación
  - Estructuras de datos locales
  - Condiciones límite
  - Caminos independientes
  - Caminos de manejo de errores

Pruebas de integración

- Existen dos estrategias diferentes para las pruebas de integración OO:
  - Pruebas *basadas en hilos*
  - Pruebas *basadas en usos*
- Las pruebas *basadas en hilos* integran el conjunto de clases requeridas para responder a una entrada o suceso del sistema.
  - Cada *hilo* se integra y prueba individualmente.
    - En este contexto *hilo* es encadenamiento de mensajes, al estilo diagrama de secuencia.
  - Además, se aplican *pruebas de regresión*.
- Las pruebas *basadas en el uso* siguen la jerarquía de dependencias de clases y subsistemas.
  - Comienzan la construcción del sistema probando las *clases independientes*.
    - Las clases independientes utilizan muy pocas o ninguna clases servidora.
  - Después se continúa una secuencia de pruebas por capas de clases dependientes.
    - Se progresa hasta construir el sistema completo.

Pruebas de regresión

- La *prueba de regresión* consiste en volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente.
  - Busca asegurar que los cambios que se derivan al integrar módulos probados independientemente no propaguen efectos colaterales no deseados.

Pruebas de sistema

- Las *pruebas de sistema* se centran en las acciones visibles al usuario y las salidas reconocibles desde el sistema.
  - Pruebas de validación
  - Pruebas alfa
  - Pruebas beta
- Las *pruebas de validación* se basan en los casos de uso y son las principales.
  - Se comprueba que el sistema cumple con los requisitos del cliente y usuario.
- Las *pruebas alfa* y *beta* surgen en respuesta a la dificultad de prever como utilizará el usuario el programa.
  - Las *pruebas alfa* las llevan a cabo usuarios finales en el lugar de trabajo del desarrollador, con el desarrollador.
  - Las *pruebas beta* se llevan a cabo en el lugar de trabajo de los usuarios finales, sin el desarrollador.

Otras pruebas de sistema

- Muchos sistemas de computadora deben ser capaces de recuperarse ante fallos u obviarlos.
  - La *prueba de recuperación* fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo de forma satisfactoria.
- Muchos sistemas deben proteger sus datos de accesos y manipulaciones inadecuadas.
  - La *prueba de seguridad* intenta verificar que los mecanismos de protección incorporados en el sistema lo protegen de accesos indebidos.
- Los sistemas se pueden utilizar más allá de su capacidad normal.
  - La *prueba de resistencia* ejecuta un sistema de forma que demande recursos en cantidad, frecuencia o volúmenes anormales.
- Los sistemas de tiempo real y los sistemas empotrados tienen fuertes requisitos de rendimiento.
  - La *prueba de rendimiento* está diseñada para probar el rendimiento del software en tiempo de ejecución dentro de un sistema.
  - Se da incluso a nivel unidad, pero en el contexto del sistema es definitiva.

TIPOS DE PRUEBAS

Pruebas de caja negra y de caja blanca

- La *prueba de caja negra* se centra en el comportamiento observable del sistema, no en la organización interna del mismo.
  - Las pruebas de sistema anteriores pertenecen a esta categoría.
- La *prueba de caja blanca* usa la estructura de control descrita como parte del diseño para derivar los casos de prueba.
  - Busca garantizar que se recorren todos los caminos independientes de cada módulo.
  - Busca ejercitar todas las decisiones lógicas en sus vertientes *verdadera* y *falsa.*
  - Busca ejercitar todos los bucles en sus límites.
  - Busca ejercitar las estructuras internas de datos.

AUTOMATIZACIÓN DE PRUEBAS

Automatización de pruebas

- Existen marcos que sirven para automatizar el proceso de prueba.
  - Generan entradas para componentes del sistema y recogen sus salidas.
  - Comparan objetos obtenidos con objetos esperados.
- Algunos ejemplos son:
  - jUnit (http://www.junit.org/)
  - C++Unit (http://cppunit.sourceforge.net/)
  - jUnitScenario
    - Es un marco automatizado construido sobre jUnit.
  - HTTPUnit (http://httpunit.sourceforge.net/)
    - Para emular comportamientos de navegación.
  - JMeter (http://jmeter.apache.org/)
    - Para aplicaciones web y factores de carga.
  - SUT (*Schema Unit Test*) (http://sut.sourceforge.net/)
    - Para comprobar esquemas XML.

Desarrollo basado en pruebas

- El *desarrollo basado en pruebas* trabaja en ciclos muy cortos de desarrollo en los que se agrega una prueba y se modifica el sistema para que funcione.
  - Típico del *eXtreme Programming*.
- Estas *pruebas de programador* son recogidas y ejecutadas periódicamente.
  - Cada vez que cualquier programador libera cualquier código en el repositorio.
    - Varias veces al día

CONCLUSIONES

Conclusiones

- Las pruebas son fundamentales para obtener sistemas correctos en tiempo.
- Se aplican a todos los artefactos del desarrollo a lo largo de todo el proyecto.
  - No son exclusivas del código.
  - Se ha de empezar con ellas desde la misma planificación.
  - Aunque no tienen el mismo peso en todas las fases del proyecto.
- Al igual que el resto de los componentes de un proyecto, las pruebas se han de planificar, especificar, ejecutar y valorar.
- Existen herramientas que permiten la automatización, al menos parcial, de algunas pruebas.

Glosario

- ECS = Elemento de Configuración del Software
- OO = Orientación a Objetos, Orientado a Objetos
- RTF = Revisión Técnica Formal
- SUT = *Schema Unit Test*
- XML = *eXtensible Markup Language*

Referencias

- R. Pressman: Ingeniería del Software. Un enfoque práctico, 7ª edición. McGraw-Hill, 2010.
  - Capítulos 17-20
- I. Sommerville: Ingeniería del Software, 7ª edición. Addison Wesley, 2007.
  - Capítulos 17-20, 22-24
- I. Jacobson, G. Booch, J. Rumbaugh: El proceso unificado de desarrollo de software. Addison-Wesley, 2000
  - Capítulos 10 y 11