Introducción a la ingeniería del software

Autores: Simon Pickin
Marisol García Valls

Address: Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid

Version: 1.0

© Los autores

Índice

© Los autores

1. Visión general / definición de términos

© Los autores

¿Qué es la ingeniería del software? (1/4)

- El establecimiento y uso de principios de ingeniería robustos, orientados a obtener económicamente software que sea fiable y funcione eficientemente sobre máquinas reales.

  F.L. Bauer. *Software Engineering.* Information Processing 71., 1972

© Los autores

¿Qué es la ingeniería del software? (2/4)

- La disciplina tecnológica y de gestión que concierne a la producción y el mantenimiento sistemático de productos software desarrollados y modificados dentro de unos plazos estipulados y costes estimados.

  R. Fairley. *Software Engineering Concepts*. New York: McGraw-Hill, 1985.

© Los autores

¿Qué es la ingeniería del software? (3/4)

- Ingeniería del software. (1) La aplicación de un enfoque sistemático, disciplinado y cuantificable del desarrollo, la operación y el mantenimiento del software; esto es, la aplicación de la ingeniería al software. (2) El estudio de enfoques tales como (1).

  IEEE Std 610-1990

© Los autores

¿Qué es la ingeniería del software? (4/4)

- Ingeniería es la aplicación sistemática de conocimiento científico en la creación y construcción de soluciones, que satisfacen una buena relación efectividad/precio, de problemas prácticos al servicio de la humanidad. La ingeniería del software es la forma de ingeniería que aplica los principios de las ciencias de la computación y las matemáticas en la obtención de soluciones de los problemas del software que satisfacen una buena relación efectividad/precio.

SEI Report on Undergraduate Software Engineering Education, 1990.

© Los autores

¿Qué no es la ingeniería del software? (1/2)

- **Las ciencias de la computación**
  - atañe a la teoría y a los aspectos fundamentales
  - la ingeniería del software atañe a los aspectos prácticos del desarrollo y entrega de software útil.
  - aún son insuficientes para actuar como apuntalamiento de la ingeniería del software (al contrario, p.e., de la física y la ingeniería eléctrica)

Software Engineering, Sommerville

© Los autores

¿Qué no es la ingeniería del software? (2/2)

- **La ingeniería de sistemas**
  - atañe a todos los aspectos del desarrollo de sistemas basados en ordenadores, incluyendo el hardware, el software y la ingeniería de procesos
  - la ingeniería del software es la parte de este proceso que atañe al desarrollo de la infrastructura software, el control, las aplicaciones y las bases de datos del sistema.
  - los ingenieros de sistema están implicados en la especificación del sistema, el diseño arquitectónico, la integración y el despliegue.

Software Engineering, Sommerville

© Los autores

¿Para qué la ingeniería del software? (1/4)?

- El término “ingeniería del software” se popularizó al final de los años 60
- Respuesta a la llamada “crisis del software”
  - las prestaciones del hardware aumentaban mucho más rápido que las del software.
  - el desarrollo de sistemas de software grandes resultaba muy insatisfactorio:
    - entrega habitualmente retrasada (a veces por mucho)
    - presupuesto habitualmente excedido (a veces masivamente)
    - calidad del producto final habitualmente baja: productos poco fiables y difíciles de mantener

© Los autores

¿Para qué la ingeniería del software? (2/4)?

- Las técnicas utilizadas para software “uni-desarrollador” no escalan
  - proyectos grandes necesitan un equipo
  - la calidad de comunicación entre los miembros es un problema serio → documentación
- Deseo de beneficiarse de experiencia previa
- Necesidad de planificar para el mantenimiento y la evolución
  - Las decisiones de diseño importantes deben documentarse

© Los autores

¿Para qué la ingeniería del software? (3/4)?

- La comunicación con el cliente/usuario es primordial
  - entender los requisitos del cliente
  - p.e. *Xtreme programming* (programación extrema): el cliente tiene representante en el equipo de desarrollo

© Los autores

¿Para qué la ingeniería del software? (4/4)?

- Necesidad de estimar el dinero, tiempo y esfuerzo requeridos
  - las estimaciones se basan principalmente en el modelado del proyecto actual y su comparación con proyectos anteriores
  - ¿Queremos el trabajo? ¿Cuánto cobramos?
  - Constructive Cost Model COCOMO
    Constructive Systems Engineering Cost Model COSYSMO
  - *The mythical man month*, Brooks, 1975:
    añadir recursos humanos a un proyecto de software atrasado lo atrasa más

© Los autores

¿Qué es un proceso de desarrollo? (1/2)

- Proceso:

  Una serie de acciones u operaciones que conducen a un fin (*Websters*)

  conjunto de las fases sucesivas de un fenómeno natural o de una operación artificial (*RAE*)

- Proceso de desarrollo de software

  El conjunto de actividades, métodos y prácticas utilizados en la producción y evolución de software.

© Los autores

¿Qué es un proceso de desarrollo? (2/2)

- Un proceso de desarrollo de software puede incluir
  - un modelo de ciclo de vida
    - divide el desarrollo en fases y prescribe las actividades que deben realizarse en cada fase
    - proporciona criterios para determinar cuándo cada fase de desarrollo ha terminado
    - define los deliverables / artefactos / productos de cada fase
  - consideración de herramientas y equipamiento
  - consideración de personal y de su organización
  - restricciones sobre las actividades, los artefactos, las herramientas, el personal etc.
- Para muchos autores:

  proceso de desarrollo de software = ciclo de vida de software

© Los autores

¿Qué es un ciclo de vida de software?

- El periodo de tiempo que comienza cuando se concibe un software y concluye cuando el producto ya no está disponible para su uso.
- El ciclo de vida del software típicamente incluye una fase de requisitos, una fase de diseño, una fase de pruebas, una fase de instalación y aceptación, una fase de operación y mantenimiento, y, en ocasiones, una fase de retirada.
- Un *modelo de ciclo de vida* es una abstracción particular que representa un ciclo de vida de software. Un modelo de ciclo de vida se denomina con frecuencia un ciclo de vida de desarrollo software (SDLC, siglas inglesas).

  IEEE Standard Glossary of Soft. Eng. Terminology

© Los autores

El modelado de software (& hardware)

- Perspectiva escéptica sobre modelos de software:
  "burbujas y flechas, al contrario que los programas, nunca cascan" Bertrand Meyer 1997
- El uso de modelos es tan antiguo como la ingeniería
  - antes de construir el ente real, los ingenieros construyen modelos y aprenden de ellos
- Algunas características deseables de un modelo
  - abstracto
  - comprensible
  - preciso
  - predictivo
  - no muy caro de construir

© Los autores

Modelado: el propósito de modelos

- Ayudarnos a entender un problema complejo mediante análisis y simulación
- Permitir la investigación y comparación de soluciones alternativas
- Facilitar la comunicación de ideas sobre un problema o sobre su solución
- Permitir la detección de errores y omisiones durante el diseño
- Para dirigir la implementación
  - particularidad del software:
    el modelo se convierte en la implementación

© Los autores

2. Modelos de ciclo de vida

© Los autores

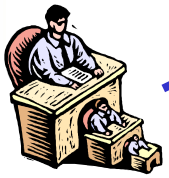
El proceso de desarrollo de software

Análisis de sistema

Desarrollo

Operación y Mantenimiento

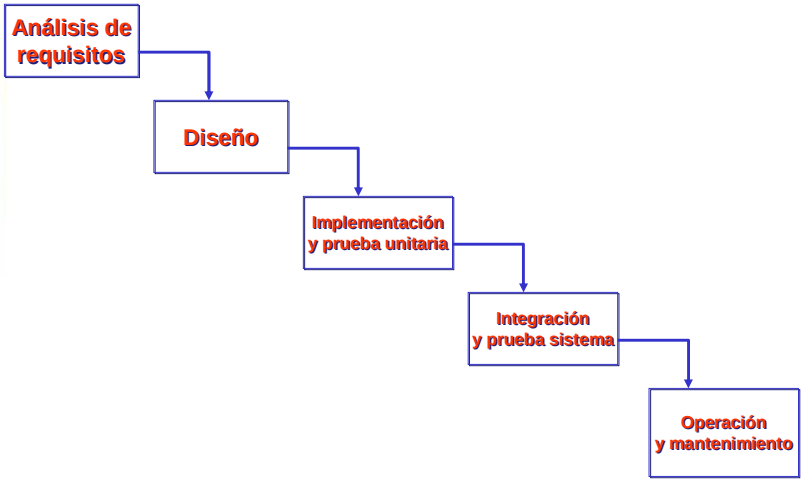
Sistema software



Requisitos de usuario

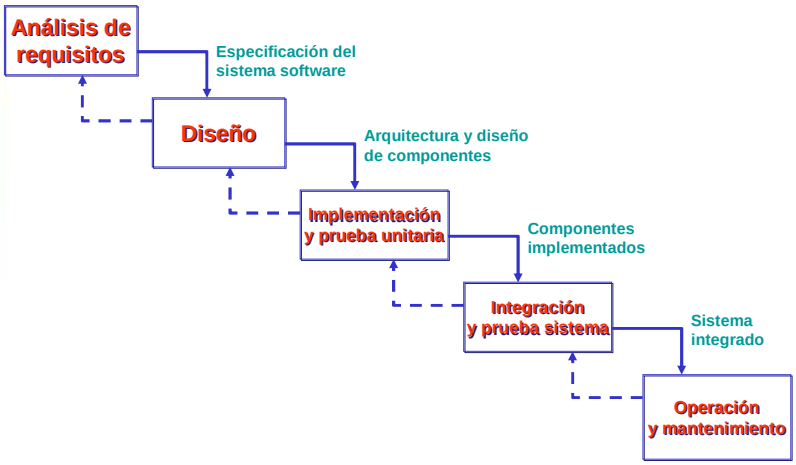
© Los autores

Modelo de ciclo de vida en cascada (1/2)



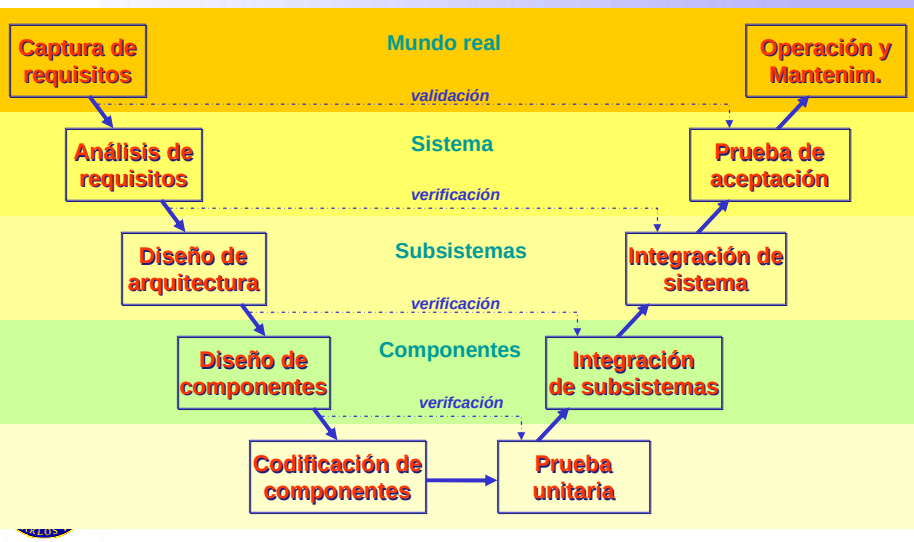
© Los autores

Modelo de ciclo de vida en cascada (2/2)



© Los autores

Modelo de ciclo de vida en V



© Los autores

Modelo de ciclo de vida incremental

- Se planifican el sistema entero y sus distintas iteraciones al principio del desarrollo
- El desarrollo y la entrega se dividen en incrementos
  - con cada incremento se entrega parte de la funcionalidad
- Se priorizan los requisitos del usuario
  - los que tienen la prioridad más alta se incluyen en los primeros incrementos
- Se congelan los requisitos de un incremento cuyo desarrollo ha empezado
  - los requisitos para incrementos posteriores pueden seguir evolucionando
- **Prototipado**:
  - cada iteración puede tratarse como un prototipo

© Los autores

Modelo de ciclo de vida evolutivo

- No se planifican el sistema entero y sus distintas iteraciones
  - el sistema final evoluciona a partir de una especificación esbozo inicial
- Se empieza con los requisitos bien entendidos y se añaden elementos nuevos tal como los propone el cliente a partir de la iteración anterior
- El objetivo de cada iteración es entender los requisitos del sistema
- **Prototipado**:
  - cada iteración puede tratarse como un prototipo

© Los autores

Modelo de ciclo de vida en espiral (versión generalizada)

Determinación de:

objetivos

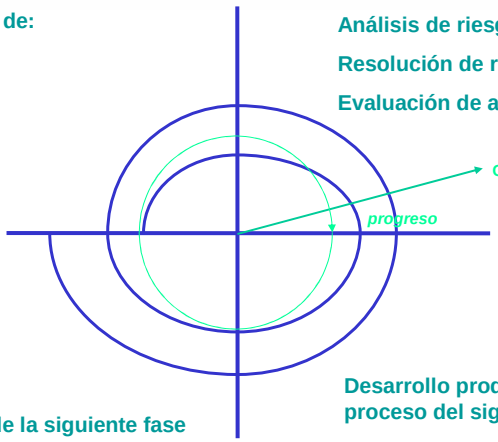
restricciones

alternativas

Análisis de riesgos

Resolución de riesgos

Evaluación de alternativas



Evaluación

Planificación de la siguiente fase

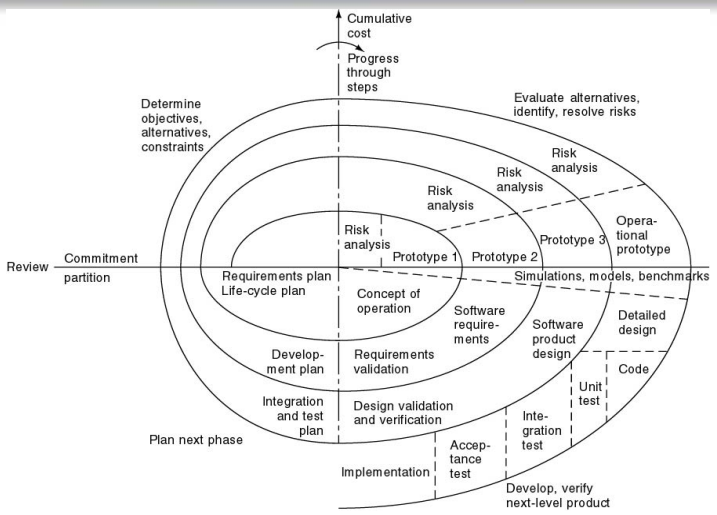
Revisión proceso

Desarrollo producto y proceso del siguiente nivel

V&V de producto y proceso

© Los autores

Modelo de ciclo de vida en espiral (versión original)



Source: A Spiral Model of Software Development and Enhancement
Barry Boehm, IEEE Computer, May 1988

© Los autores

3. Captura, Análisis y Especificación de Requisitos

© Los autores

Ingeniería de requisitos

Uso sistemático de principios contrastados, técnicas, lenguajes y herramientas para obtener el análisis efectivo en coste y documentación de las necesidades en continua evolución del usuario, y la especificación del comportamiento externo del sistema requerido para satisfacer dichas necesidades.

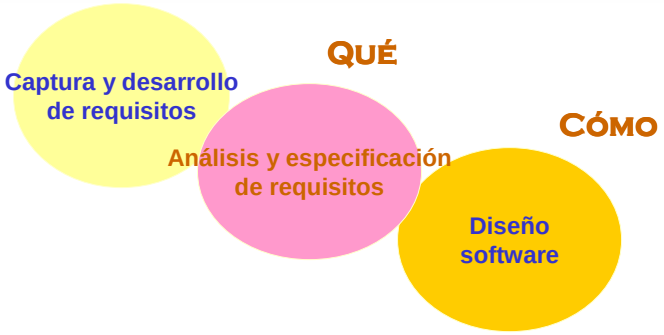
Donald Reifer (*ver Pressman*)

La tarea de capturar, estructurar y representar con precisión los requisitos del usuario de modo que puedan ser correctemente encarnados en sistemas que verifican tales requisitos.

FOLDOC (*http://foldoc.org/*)

© Los autores

Contexto de la fase de análisis



Definición habitual:
ingeniería de requisitos = captura, análisis y especificación de requisitos

Algunos autores:
ingeniería de requisitos = análisis de requisitos ⊂ captura y especificación de requisitos

© Los autores

Análisis estructurado

© Los autores

Análisis orientado a objetos

- Basado en objetos y sus operaciones y atributos, en vez de en flujos de datos
- Notación más comúnmente usada: UML
  - modelos estructurales (de clases, de componentes,…)
  - modelos comportamentales (de casos de uso, de estados, de colaboraciones, de interacciones,…)
- Modelos estructurales
  - a través de análisis de dominio
- Modelos comportamentales
  - modelado de casos de uso es la técnica más popular
  - uso de otros modelos comportamentales puede ser problemático
    - ¿cómo refinar en modelo de diseño?

© Los autores

4. Diseño del software

© Los autores

Notas históricas breves (1/2)

- Finales de los 60: diseño orientado a control (*Structured Programming*)
  - modularidad
  - construcciones de programa de entrada y salida única (secuencia, selección, iteración)
  - prohibición de la construcción "*go to*" (¿y la gestión de excepciones?) :

    "Goto Statement Considered Harmful", Dijkstra, *Comm. ACM*, 1969

© Los autores

Notas históricas breves (2/2)

- 1970s: diseño orientado a estructuras de datos (extensión de *Structured Programming*)
  - la estructura del código del programa refleja la estructura de los datos

    p.e. Jackson Structured Programming
- 1980s: diseño orientado a objetos
  - unifica el diseño orientado a estrcturas de datos y el diseño orientado a control (¿es cierto?)
  - notación más comunamente usada: UML

© Los autores

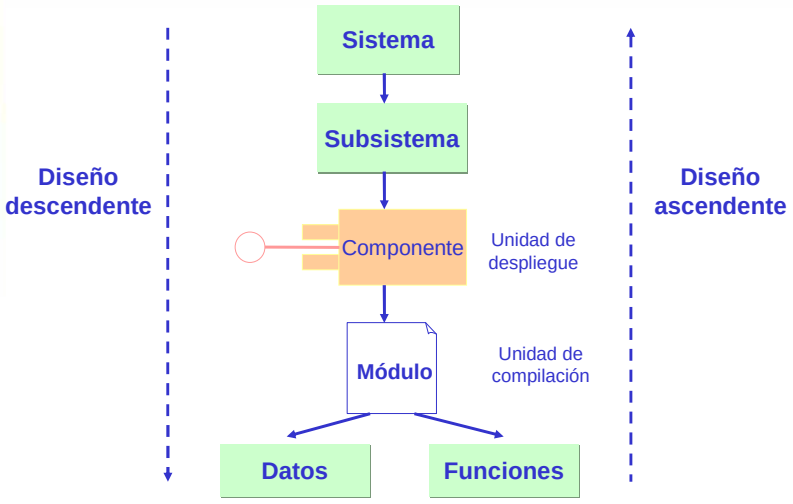
¿Qué es una arquitectura software?

- La organización fundamental de un sistema, encarnada en sus componentes, las relaciones entre ellos y con su entorno, y los principios que gobiernan su diseño y evolución.

  ANSI/IEEE Std 1471-2000, *Prácticas recomendadas para la descripción arquitectural de sistemas mayoritariamente software*

© Los autores

Arquitecturas de software



© Los autores

Fundamentos de la arquitectura de software

- Componentes y subsistemas
  - los elementos individuales
- Conexiones
  - cómo los componentes se comunican
- Topología
  - cómo los components y conexiones se organizan
- Restricciones
  - sobre componentes, conexiones, topología, evolución,…

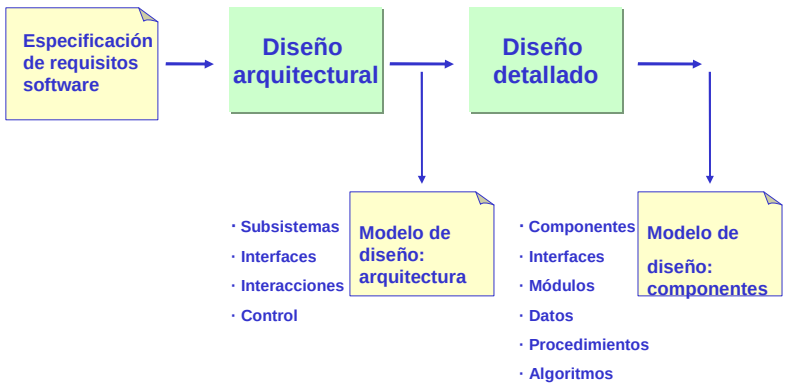
© Los autores

Estilos de arquitectura software

- Restringen la manera en la que pueden conectarse los componentes
- Promocionan principios fundamentales
  - separación de intereses, generalidad, incrementalidad, acoplamiento débil, cohesión fuerte,...
- Basado en experiencias exitosas
  - elegida también en función del tipo de aplicación
- Ej: *pipe-and-filter*, capas, bus de software, cliente-servidor, P2P, jerárquica, control centralizado, cliente-servidor *3-tier*, etc.

© Los autores

Proceso de diseño típico



© Los autores

Algunos conceptos básicos de diseño

- **Abstracción**
  - énfasis en detalles importantes, omitiendo características no relevantes en el contexto
- **Refinamiento**
  - proceso de añadir más detalles paulatinamente, pasando de modelos más abstractos a modelos más concretos.
- **Modularidad**
  - descomposición en componentes que se integrarán para satisfacer los requisitos del problema
- **Ocultación de información / encapsulación**
  - los componentes sólo dejan disponible para su entorno la información que necesitarán otros componentes (las interfaces no ofrecen detalles de implementación / diseño)

© Los autores

5. La Calidad del Software

© Los autores

Factores en la calidad del diseño (1/2)

- **Criterios externos (perspectiva del usuario)**
  - la corrección
  - la fiabilidad
  - la usabilidad (facilidad de manejo)
  - las prestaciones
  - la robustez
- **Criterios internos (perspectiva del desarrollador)**
  - la eficiencia
  - la mantenibilidad
  - la reusabilidad
  - la portabilidad
  - la interoperabilidad

© Los autores

Factores en la calidad del diseño (2/2)

- **Desde la perspectiva del mantenimiento y reuso**
  - **Independencia funcional de componentes**
    - cohesión intra-componente fuerte
    - acomplamiento inter-componente débil
  - **Legibilidad / comprensibilidad**
    - esquema de nombramiento
    - documentación actualizada y completa
    - simplicidad / elegancia
  - **Adaptabilidad**
    - evolutividad y generalidad
    - automatización del acceso a la documentación
    - automatización del control de versiones

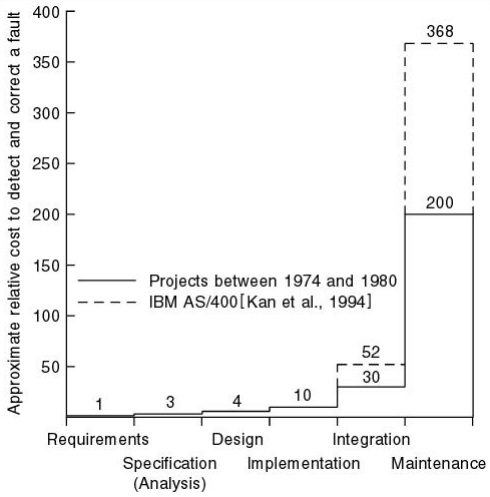
© Los autores

Garantía de calidad / Control de calidad del software (1/2)

- Implica actividades realizadas a lo largo del ciclo de vida
- Definición de **verificación (a partir de IEEE)**:
  - asegurar que un sistema software, o modelo del mismo, cumple una especificación (con frecuencia producida en una fase previa del desarrollo)
  - coherencia interna
  - "estamos construyendo el sistema correctamente?"
- Definición de **validación (a partir de IEEE)**:
  - asegurar que un sistema software, o modelo del mismo, cumple los requisitos (los deseos del cliente)
  - coherencia externa
  - "estamos construyendo el sistema correcto"
- La prueba del software
  - usualmente considerada validación; también puede ser verificación
- Métricas de software

© Los autores

Garantía de calidad / Control de calidad del software (2/2)



Fuente: Object-oriented Software Engineering. Steven Schach. McGraw-Hill.

© Los autores

Métodos formales (1/2)

- **Semántica formal**: semántica basada en teoría de conjuntos, álgebra, lógica, autómatas, teoría de grafos, etc.
- **Especificación formal**: descripción abstracta con una semántica formal.
  - orientada a modelos
  - orientada a propiedades
- **Método formal**: método utilizado en el desarrollo de software/hardware que implica el uso y manipulación de especificaciones formales, p.e.:
  - demostración de propiedades de especificaciones formales
  - derivación de implementaciones, u otros artefactos software (p.e. casos de prueba), a partir de especificaciones formales
  - demostración de propiedades de una implementación a partir de una interpretación abstracta del código
  - …

© Los autores

Métodos formales (2/2)

- Especialmente importantes para sistemas críticos
  - aviones, trenes, metro
  - sistema de transmisión eléctrica, centrales elétricos
  - redes de telecomunicaciones
  - ...
- También para sistemas seguros
- Puede ser interesante para sistemas baratos pero producidos en cantidades muy grandes
- Frecuentemente introducidos después de la ocurrencia de un problema grave, p.e.:
  - *model-checking* en Intel después del descubrimiento del error de división de números flotantes del Pentium

© Los autores

Métodos formales y garantía de calidad (1/2)

- Aumento de la comprensión del sistema modelado
- Automatización de actividades comunes del desarrollo del software
  - generación de código
  - generación de pruebas / síntesis de pruebas
  - …
- Análisis/simulación de modelos desde fases tempranas del desarrollo:
  - temprana comprobación de coherencia
  - temprana detección de errores, omisiones, ambigüedades, propiedades indeseadas,…

© Los autores

Métodos formales y garantía de calidad (2/2)

- Análisis de las implementaciones
  - detección de errores, omisiones, ambigüedades, propiedades indeseables,…
- Transformación de modelos & comprobación de coherencia entre modelos:
  - en diferentes niveles de abstracción
  - de diferentes fases del desarrollo

© Los autores

6. La Prueba de Software

© Los autores

Visión general

- La prueba de software implica la ejecución de la implementación de una manera controlada y utilizando datos de entrada cuidadosamente seleccionados para luego observar el resultado.
- La prueba de software es un aspecto de la garantía de la calidad de software (*Software Quality Assurance* o SQA).

© Los autores

Definición de un caso de prueba

- La especificación de una interacción
  - entre la implementación bajo prueba o IUT (en sus siglas inglesas) y el software de prueba, o usuario humano, que desempeña el papel del entorno de la IUT,
  - en la que este último estimula la IUT a través de sus interfaces, observa su comportamiento y sus respuestas
  - y, si incluye un *oracle*, asigna un *veredicto* al resultado de esta interacción.
- Un caso de prueba se diseña para ejercer una ejecución particular o para verificar la conformidad con un requisito específico.

© Los autores

Prueba de software: cierto o falso (1/3)

- La mayoría de los desarrolladores subestiman el esfuerzo que se tiene que dedicar a la prueba de software.
  - ✓ sin duda; los desarrolladores tienen tendencia a pensar en términos de lineas de código al día
- Las pruebas debería hacerlas siempre un equipo distinto al del desarrollo.
  - ✗ No siempre para todo tipo de prueba (p.e. pruebas de unidad), aunque siempre en el caso de algunos tipos de prueba
- No se puede ejecutar ninguna prueba hasta que esté terminado el código de toda la aplicación.
  - ✗ p.e. la prueba de unidad
- Actualmente, la prueba es más artesanía que ciencia.
  - ✓ frecuentemente, la experiencia del personal es crucial
- No se pueden escribir casos de prueba antes de que esté disponible el código que se quiere probar.
  - ✗ No siempre (p.e. JUnit)

© Los autores

Prueba de software: cierto o falso (2/3)

- El fin último de la prueba de software es demostrar que el software que se está desarrollando está libre de errores.
  - ✘ es imposible de conseguir con las pruebas
- Después de reparar unos errores encontrados en la fase de pruebas, se debería probar de nuevo el software.
  - ✓ es posible que no se hayan reparado los errores o que la reparación de los errores haya introducido nuevos errores
- La fase de pruebas de un ciclo de vida de desarrollo software típico termina cuando el software que se está desarrollando ya no contiene errores.
  - ✘ nunca se puede estar seguro de que no hay errores
- Si un módulo de un producto de software bien probado se reutiliza en otro producto de la misma línea de productos, no hace falta probarlo de nuevo.
  - ✘ p.e. Ariane 5

© Los autores

Prueba de software: cierto o falso (3/3)

- Las actividades de la prueba de software se pueden automatizar facilmente.
  - ✘ es precisamente por esta razón que es más artesanía que ciencia
- Todos los errores encontrados en un producto de software en desarrollo deberían repararse antes de su publicación
  - ✘ se trata siempre de un compromiso entre severidad del error y coste de su reparación
- La generación automática de pruebas tiene el potencial de producir ganancias enormes de productividad
  - ✓ la automatización es difícil pero muy deseable
- Cuando se modifica un software, los casos de prueba que se utilizaron para probar la versión anterior deberían ejecutarse de nuevo sobre la versión modificada.
  - ✓ eso es la prueba de regresión; por supuesto, también se pueden necesitar casos de prueba nuevos

© Los autores

Prueba de software, nociones básicas (1/2)

- El objetivo de la prueba de software es encontrar errores y NO demostrar su ausencia
  - una buena prueba encuentra un error
  - no existe un número de pruebas que pueda garantizar que un programa no tenga errores
- Se debería probar que la aplicación
  - hace lo que tiene que hacer
  - no hace lo que no tiene que hacer (en la medida de lo posible)
- Enfoques (a veces también se utiliza el término “caja gris”)
  - la prueba de caja negra
  - la prueba de caja blanca (la prueba estructural)
- Fases
  - la prueba de unidades
  - la prueba de integración
  - la prueba de sistema

© Los autores

Prueba de software, nociones básicas (2/2)

- Cobertura
  - caja blanca: segmentos, ramas, condiciones, bucles,…
  - caja negra: requisitos
- Selección de datos de prueba (sobre todo para caja negra)
  - partición en equivalentes (hipótesis de uniformidad)
  - análisis de valores límites
- Otros tipos de pruebas
  - pruebas de aceptación
  - pruebas de prestaciones
  - pruebas de robustez
  - pruebas de resistencia
  - pruebas de interoperabilidad
  - pruebas de regresión
  - pruebas de mutación

© Los autores

7. Algunos novedades en el campo

© Los autores

Algunos novedades en el campo (1/3)

- Patrones de diseño
  - solución general, repetible, a un problema recurrente de diseño software
  - inspiración en la arquitectura, especialmente en los trabajos de Christopher Alexander
  - ¿fundamentos teóricos insuficientes?
- Armazones software (*frameworks*):
  - diseño reutilizable para un sistema o subsistema software (¿patrón arquitectural?)
- Líneas de producto software
  - proceso de desarrollo software para un conjunto de productos relacionados
  - generalmente usan armazones software

© Los autores

Algunos novedades en el campo (2/3)

- Desarrollo ligero, en respuesta al hinchazón (*bloat*) del software y de la documentación:
  - *Xtreme programming* (programación extrema), Scrum, etc.
  - modelado ágil
- Desarrollo dirigido por modelos
  - más que los programas, los modelos son los artefactos primarios (⇒ generación de código)
  - iniciativa MDA de la OMG (PIM - modelo independiente de plataforma − y PSM − modelo específico a la plataforma)
  - refactoría de diseño (*design refactoring*)
- Arquitectura orientada a servicios (SOA)
  - estilo de arquitectura cuyo fin es conseguir un acoplamiento débil entre agentes software interactuando y desempeñando papeles de productor o consumidor

© Los autores

Algunos novedades en el campo (3/3)

- Ingeniería del software basada en componentes
  - sistema ensamblado (al menos parcialmente) a partir de componentes ya existentes.
- Ingeniería del software libre
  - colaboración con frecuencia implica un gran número de desarrolladores espacialmente separados
  - gestión y estructura organizacional novedosas
  - heterogeneidad en el uso de herramientas, enfoques etc.
  - relación con la ingeniería del software tradicional actualmente bajo estudio
- Desarrollo de aplicaciones Web / servicios Web
  - tipo de aplicaciones actualmente muy populares
  - ¿se dan características de desarrollo particulares? ¿o es solo *marketing*?

© Los autores