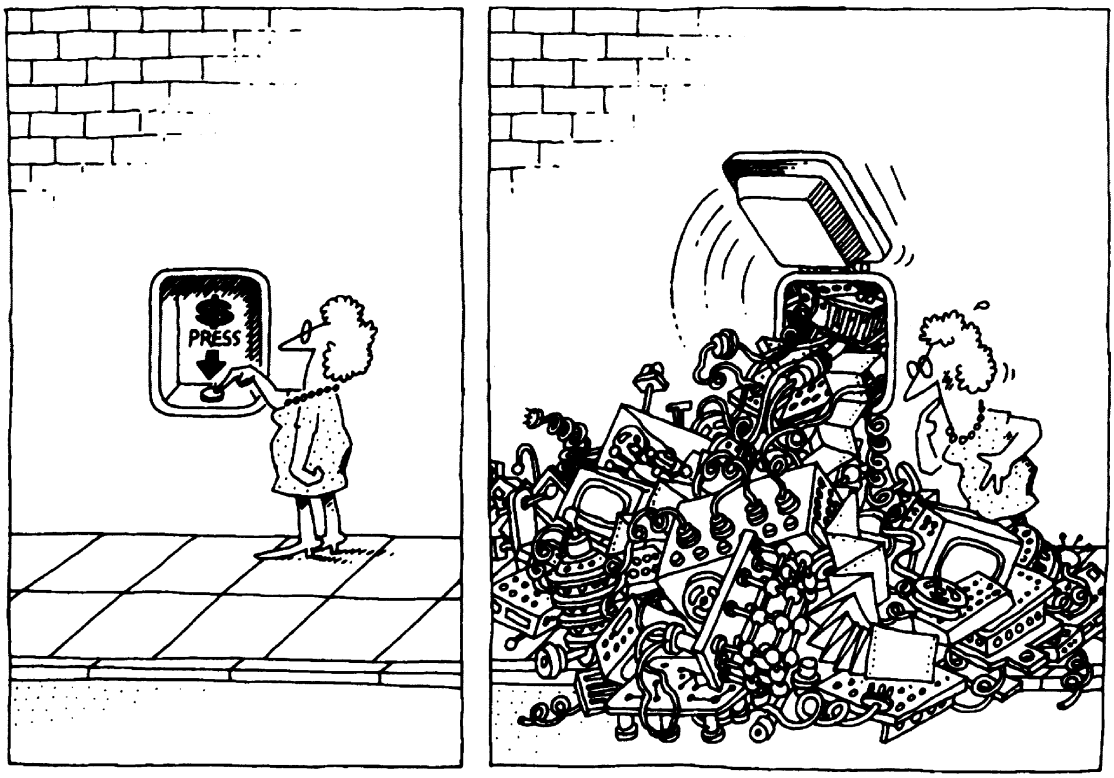
Tema 1º
La complejidad del desarrollo de software

Contenidos

1.1 La complejidad inherente al software
1.2 La estructura de los sistemas complejos
1.3 Imponiendo orden al caos
1.4 Diseño de sistemas complejos
1.5 Resumen
1.6 Test de autoevaluación
1.7 Lecturas recomendadas
1.8 Referencias



La tarea del equipo de desarrollo de software es ofrecer ilusión de simplicidad.
[Booch 94]

1.1 La complejidad inherente al software

La construcción de software puede involucrar elementos de gran complejidad, que en muchos casos no son tan evidentes como los que se pueden ver en otras ingenierías. Un puente, un edificio, una mina, una red de ferrocarriles son ejemplos de sistemas complejos de otras ingenierías, pero el ingeniero del software construye sistemas cuya complejidad puede parecer que permanece oculta. El usuario siempre supone que Informática todo es muy fácil (“apretar un botón y ya está”)

La complejidad se estudiará en los siguientes apartados: [Booch 94]

- Las propiedades de los sistemas de software simples y complejos
- ¿Por qué el software es complejo de forma innata?
- Las consecuencias de la complejidad ilimitada

Propiedades de los sistemas de software simples o artesanales

- No son complejos
- Suelen estar construidos y mantenidos por una sola persona (software artesanal)
  - No suelen pasar de 1.000.000 de líneas de código, aunque el número de líneas de código no sea la mejor medida de la complejidad del software.
- Ciclo de vida corto
  - El ciclo de vida del software es todo el tiempo que un proyecto software está activo, comienza con las primeras tareas de análisis y no concluye hasta que ese software deja de utilizarse por los usuarios
- Pueden construirse aplicaciones alternativas en un periodo razonable de tiempo
- No necesitan grandes esfuerzos en análisis y diseño
- No son nuestro objetivo de estudio

Las propiedades de los sistemas software complejos

- También se denominará software de dimensión industrial
- Es muy difícil o imposible que un desarrollador individual pueda comprender todas las sutilidades de su diseño
- La complejidad es una propiedad esencial de estos sistemas, que puede dominarse, pero no eliminarse
- Son el objetivo de nuestro estudio
- Ejemplos: Un sistema de reservas, anulaciones y ventas de billetes aéreos para un conjunto de compañías aéreas que se pueda utilizar en cualquier lugar del mundo.

¿Por qué el software es complejo de forma innata?

La complejidad inherente al software se deriva de los siguientes cuatro elementos [Booch 94]

- La complejidad del dominio del problema
- La dificultad de gestionar el proceso de desarrollo
- El detalle que se puede alcanzar a través del software
- El problema de caracterizar el comportamiento de sistemas discretos

La complejidad del dominio del problema

- **Gran cantidad de requisitos que compiten entre sí, incluso contradiciéndose**
  - La forma habitual de especificar los requisitos son grandes cantidades de texto con unos pocos dibujos
- **Desacoplamientos de impedancias entre usuarios del sistema y desarrolladores**
  - Los usuarios suelen tener ideas vagas de lo que desean
  - Dificultades de comunicación
  - Distintas perspectivas de la naturaleza del problema
- **Modificación de los requisitos con el paso del tiempo, pues los usuarios y desarrolladores comienzan a compenetrarse mejor**

  "No sobrevivirán las especies más fuertes, tampoco las más inteligentes, lo harán aquellas que más se adapten al cambio" ***Charles Darwin****,* El origen de las especies.
  - Mantenimiento de software
    - Cuando se corrigen errores
  - Evolución del software
    - Cuando se responde a requisitos que cambian
  - Conservación del software
    - Se emplean medios extraordinarios para mantener en operación un elemento software anticuado y decadente

La dificultad de gestionar el proceso de desarrollo

- Dirigir un equipo de desarrolladores
  - mantener la unidad de acción
  - conservar la integridad del diseño
- Manejar gran cantidad de código
  - Uso de herramientas
    - gestión de proyectos
    - Análisis, diseño e implementación
    - Desarrollo rápido de prototipos (RAD)
    - Control de versiones
  - Uso de lenguajes de muy alto nivel
  - Reutilización de código
  - Uso de frameworks (marcos estructurales)
  - Uso de componentes software
  - Uso de patrones de diseño

El detalle que se puede alcanzar a través del software

"No vuelvas a inventar la rueda"

- Evitar el desarrollo de todo desde el nivel más inferior
- Utilizar los estándares, al igual que en otras ingenierías
- Verificar la fiabilidad de los estándares existentes en el mercado

El problema de caracterizar el comportamiento de sistemas discretos

- La ejecución de un programa es una transición entre estados de un sistema discreto
- Cada estado contiene las variables, su valor, direcciones en tiempo de ejecución, etc.
- Diseñar el sistema de forma que el comportamiento de una parte del sistema tenga mínimo impacto en otra parte del mismo
- La transición entre estados debe ser determinista
  - Sin embargo a veces no lo es, pues un evento puede corromper el sistema pues sus diseñadores no lo tuvieron en cuenta
- Es imposible hacer una prueba exhaustiva
  - Es decir probar todos los casos posibles, pues aunque el número de estados es finito el determinar todas las combinaciones posibles produce un número tan elevado de combinaciones que es imposible de probar cada una de ellas

Las consecuencias de la complejidad ilimitada

"Cuanto más complejo es un sistema más probable es que se caiga"

- La crisis del software
  - *Son los sucesivos fracasos de las distintas metodologías para dominar la complejidad del software, lo que implica el retraso de los proyectos de software, las desviaciones por exceso de los presupuestos fijados y la existencia de deficiencias respecto a los requisitos del cliente* [Booch 94]
  - Este término alude al conjunto de problemas que aparecen en el desarrollo de software [Pressman 97, apartado 1.3]
  - Muchas de las causas de la crisis del software son los mitos del sofware [Pressman 97, apartado 1.4]
    - Mitos de gestión
      - Si fallamos en la planificación, podemos añadir más programadores y adelantar el tiempo perdido
    - Mitos del cliente
      - Una declaración general de los objetivos es suficiente para comenzar a escribir programas; podemos dar los detalles más adelante
    - Mitos de los desarrolladores
      - Una vez que escribimos el programa y hacemos que funcione nuestro trabajo ha terminado
      - Hasta que no se ejecuta el programa no hay forma de comprobar su calidad
      - Lo único que se entrega al terminar el proyecto es el programa funcionando

Fallo del primer vuelo del Ariane 5

“*El fallo del vuelo 501 se debió a la pérdida completa de la información de guiado y altitud, 37 segundos después de la ignición del motor principal (30 segundos después del despegue). Esta pérdida de información tiene su origen en* ***errores de diseño y especificación del software*** *del Sistema de Referencia Inercial”*

Comisión de Investigación del fallo del primer vuelo del Ariane 5

TRIBUNA DE ASTRONOMÍA, nº 130, pp. 81, 1996.

Véase la página web [ARIAN501]

Este fallo ha dado lugar a varios artículos [Meyer 97].

1.2 La estructura de los sistemas complejos

"¿Cómo puede cambiarse esta imagen desoladora? Ya que el problema subyacente surge de la complejidad inherente al software, la sugerencia que se plantea aquí es estudiar cómo se organizan los sistemas complejos en otras disciplinas. Realmente, si se hecha una mirada al mundo que nos rodea, se observarán sistemas con éxito dentro de una complejidad que habrá que tener en cuenta." [Booch 94, capítulo 1]

- Ejemplos de sistemas complejos
- Los cinco atributos de un sistema complejo
- Complejidad organizada y desorganizada

Ejemplos de sistemas complejos

- La estructura de un ordenador personal
  - Su complejidad se domina por medio de una estructura jerárquica basada en componentes
  - Puede razonarse como funciona el ordenador debido a que puede descomponerse en partes susceptibles de ser estudiadas por separado.
  - Los niveles de la jerarquía son niveles de abstracción
  - Para resolver cada problema se elige un determinado nivel de abstracción
- La estructura de plantas y animales
  - Estructura jerárquica (célula, tejido,…)
  - Agentes (savia, sangre,…) tienen comportamiento complejo y contribuye a funciones de nivel superior.
- La estructura de la materia
  - Estructura jerárquica (átomos, electrones, protones, neutrones, quarks)
- La estructura de las instituciones sociales
  - Estructuras jerárquicas que evolucionan con el tiempo

Los cinco atributos de un sistema complejo

- **Jerarquía**
  - Un sistema complejo se compone de subsistemas relacionados que tienen a su vez sus propios subsistemas, y así sucesivamente hasta que se alcanza algún nivel ´´ínfimo de componentes elementales.
- **Componentes**
  - La elección de qué componentes de un sistema son primitivos depende de la decisión del analista.
- **Enlaces entre componentes**
  - Los enlaces internos de los componentes son más fuertes que los enlaces externos entre los componentes.
- **Patrones**
  - Los sistemas complejos tienen patrones comunes, que permiten la reutilización de componentes. Así los sistemas jerárquicos están compuestos usualmente de sólo unas pocas clases diferentes de subsistemas en varias combinaciones y disposiciones que se pueden encontrar en sistemas aparentemente muy diferentes.
- **Evolución**
  - Los sistemas complejos que funcionan son la evolución de sistemas simples que funcionaron previamente como formas intermedias estables.

Complejidad organizada y desorganizada

- **La forma canónica de un sistema complejo**
  - El descubrimiento de abstraciones y mecanismos comunes facilita en gran medida la comprensión de sistemas complejos
  - Dos tipos de jerarquías
    - Jerarquía estructural (“parte-de”, ”part of “, o de clases)
    - Jerarquía de tipos (“es un”, “is a”, o de objetos)
  - Combinando los dos tipos de jerarquías con los cinco atributos de un sistema complejo se pueden definir todos los sistemas complejos de forma canónica (la forma menos compleja)
  - Se denominará arquitectura de un sistema a la estructura de clases y objetos
- **Las limitaciones de la capacidad humana para enfrentarse a la complejidad**
  - La complejidad desorganizada es la que se nos presenta antes de determinar las abstracciones y jerarquías.
  - En los sistemas discretos hay que enfrentarse con un espacio de estados claramente grande, intrincado y a veces no determinista
  - Una persona no puede dominar todos los detalles de un sistema complejo
  - La complejidad se organizará según el modelo de objetos

1.3 Imponiendo orden al caos

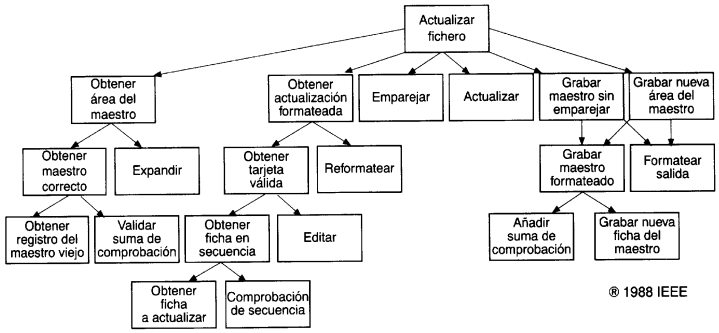
Complejidad organizada

- El papel de la descomposición
- El papel de la abstracción
- El papel de la jerarquía

El papel de la descomposición

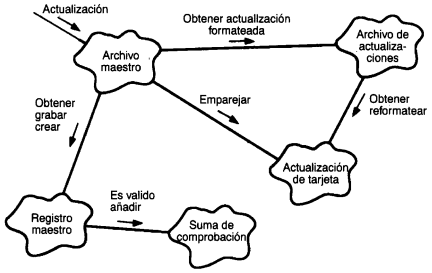
"La técnica de dominar la complejidad se conoce desde tiempos remotos: *divide et impera* (divide y vencerás)"

- **Descomposición algorítmica**
  - Por ejemplo por diseño estructurado descendente [Booch 94]



Descomposición algorítmica.

- **Descomposición orientada a objetos**
  - Se identifican objetos que hacen cosas cuando reciben mensajes [Booch 94]



Descomposición orientada a objetos.

- **Descomposición algoritmica versus descomposición orientada a objetos**
  - Visión algorítmica resalta el orden de los eventos
  - Visión orientada a objetos enfatiza los agentes que causan o padecen acciones
  - Ambas visiones son impotantes y ortogonales

Ventajas de la descomposición orientada a objetos sobre la descomposición algorítmica

- Los sistemas son más pequeños y fáciles de mantener a través de mecanismos de reutilización
- Los sistemas soportan mejor los cambios, pues están mejor preparados para evolucionar en el tiempo, dado que su diseño se basa en formas intermedias estables
- Se refleja mejor el comportamiento de los distintos elementos del sistema complejo al usar objetos
- Se reducen riesgos en los sistemas complejos al acercarse el mundo real compuesto por objetos al mundo del software
- La descomposición algorítmica no contempla los problemas de abstracción de datos y ocultación de información, ni proporciona medios adecuados para tratar la concurrencia

El papel de la abstracción

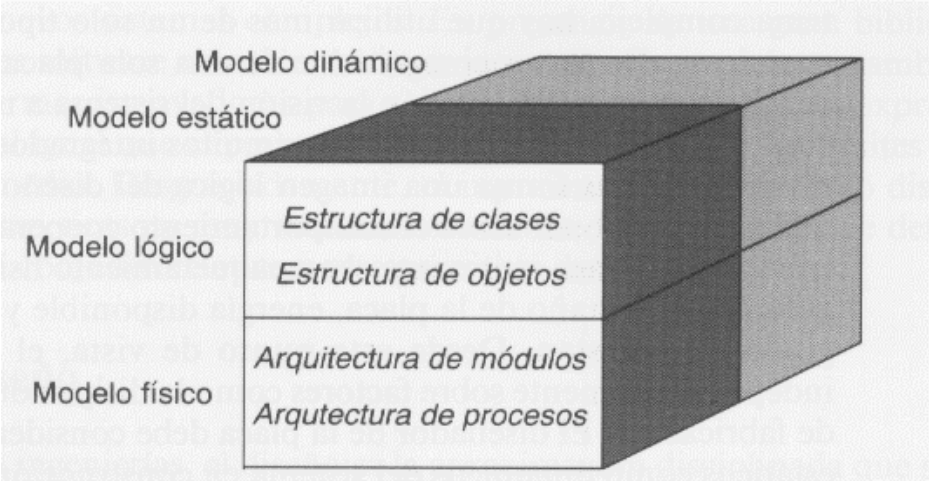
- La abstracción permite ignorar los detalles no esenciales de un objeto complejo
- La abstracción permite modelos generalizados e idealizados de objetos
- La abstracción reduce la complejidad
- Los objetos son abstracciones del mundo real, que agrupan información densa y cohesiva

El papel de la jerarquía

- El reconocimiento explícito de las jerarquías de clases y objetos dentro de un sistema software complejo es una forma de incrementar el contenido semántico de la información del sistema
- La jerarquía de objetos ilustra como diferentes objetos colaboran entre sí a través de patrones de iteracción denominados mecanismos
- La jerarquía de clases resalta la estructura y los comportamientos comunes en el interior del sistema
- La identificación de las jerarquías no es una tarea fácil
- La jerarquía de clases y objetos incluye herencia y composición.

1.4 Diseño de sistemas complejos

- **La ingeniería como ciencia y arte**
  - La práctica de cualquier disciplina de ingeniería (sea civil, mecánica o informática involucra elementos tanto de ciencia como de arte
  - La concepción de un diseño de un sistema software suele involucrar un salto de imaginación, una síntesis de experiencia y conocimiento como la que requiere cualquier artista
  - Una vez que el ingeniero ha articulado un diseño como artista, debe analizarlo técnicamente aplicando la sistemática de una metodología.
- **El significado del diseño**
  - El diseño es la aproximación disciplinada que se utiliza para concebir una solución para algún problema, suministrando así un camino desde los requisitos hasta la implementación.
  - Los productos del diseño son modelos que permiten razonar sobre las estructuras, hacer concesiones cuando los requisitos entran en conflicto y proporcionar un anteproyecto para la implementación
    - **La importancia de construir un modelo**
      - Un modelo es una representación simplificada de la realidad, que permite simular bajo situaciones tanto previstas como improbables el comportamiento del sistema
    - **Los elementos de los métodos de diseño de software**
      - **Notación**: El lenguaje para expresar cada modelo
      - **Proceso**: Actividades que se siguen para la construcción ordenada de los modelos del sistema
      - **Herramientas:** Los artefactos que facilitan la construcción del modelo, eliminando las tareas tediosas y comprobando errores e inconsistencias
    - **Los modelos del desarrollo orientado a objetos**



Los modelos del desarrollo orientado a objetos.

1.5 Resumen

- Las técnicas de análisis y diseño orientado a objetos están pensadas para desarrollo de software de dimensión industrial.
- El software es complejo de forma innata. La complejidad de los sistemas excede frecuentemente la capacidad intelectual humana
- La complejidad puede atacarse por medio del uso de la descomposición, abstracción y jerarquía (complejidad organizada)
- Los sistemas complejos pueden evolucionar desde formas intermedias estables
- El diseño orientado a objetos define una descomposición, abstracción y jerarquía basada en clases y objetos
- El diseño orientado a objetos define notaciones y procesos que conducen a modelos que permiten razonar sobre diferentes aspectos del sistema que se está simulando.
- Conviene volver a leer este tema cuando se tenga más experiencia en Análisis y Diseño

1.6 Test de Autoevaluación

1.6.1 Las técnicas de análisis y diseño orientadas a objetos **A)** Sólo se aplican para desarrollo de software artesanal o individual **B)** Están pensadas para desarrollo de software de dimensión industrial **C)** Sólo se aplican cuando se utilizan bases de datos **D)** Sólo se aplican en sistemas de tiempo real **E)** Ninguna afirmación es correcta.

1.6.2 La crisis del software es **A)** La perdida del monopolio de IBM en el campo del software **B)** El dominio de Microsoft en el mercado mundial de software **C)** Son los sucesivos fracasos de las distintas metodologías para dominar la complejidad del software **D)** Todas las afirmaciones anteriores son correctas **E)** Ninguna respuesta anterior es correcta.

1.6.3 Si se nos pide el análisis y diseño de un producto de software que ha sido creado y mantenido por una sola persona en un periodo corto de tiempo. Se puede decir que **A)** Es un sistema de software simple **B)** No necesita grandes esfuerzos de análisis y diseño **C)** Se pueden construir aplicaciones alternativas en un periodo razonable de tiempo **D)** Todas las afirmaciones anteriores son correctas **E)** Ninguna respuesta anterior es correcta

1.6.4 Si se nos pide construir un software determinado, el seguimiento de una metodología **A)** Facilita la producción de software de calidad **B)** Una forma de dominar la complejidad del software **C)** Es una colección de métodos aplicados a lo largo del ciclo de vida del desarrollo del software y unificados por alguna aproximación general o filosófica **D)** Todas las afirmaciones anteriores son correctas **E)** Ninguna respuesta anterior es correcta

Respuestas: 1.6.1 B), 1.6.2 C), 1.6.3 D), 1.6.4 D)

1.7 Lecturas recomendadas

La mayor parte de los conceptos de este tema están explicados en el capítulo 1 del libro [Booch 94]. También se recomienda la lectura de la sección I (The Problem), capítulos 1 a 9 del libro [Beck 2000].

1.8 Referencias

- [ARIAN501] *Rapport de la Comission d'enquête Ariane 501*. `http://www.cnes.fr/actualites/news/rapport_501.html`
- [Beck 2000] K. Beck. *Extreme Programming explained. Embrace change.*Addison-Wesley, 2000.
- [Booch 94] G.Booch. *Object-Oriented Analysis and Design with Applications. Second Edition*. Addison-Wesley (1994). Versión Castellana: *Análisis y diseño orientado a objetos con aplicaciones.* 2ª Edición. Addison-Wesley/ Díaz de Santos (1996).
- [Meyer 97] J.-M. Jézéquel, B. Meyer. *Design by Contract: the lessons of Ariane*. IEEE Computer. January 1997, Vol. 30, No. 1, pp 129-130.
- [Pressman 97] R.S.Pressman. *Software engineering: a practitioner´s approach*. McGraw-Hill 4 Ed. (1997). Versión castellana: *Ingeniería del software: Un enfoque práctico.* 4ª Edición. McGraw-Hill(1998).
- [Tribuna Astronomía 130, 1996] Comisión de Investigación del fallo del primer vuelo del Ariane 5. TRIBUNA DE ASTRONOMÍA, nº 130, pp.81, 1996