Diseño Basado en Componentes

Introducción

Ingeniería Informática
Universidad Carlos III de Madrid

Tabla de contenidos

Componentes y arquitectura (i)

- Los componentes se vienen utilizando en otras ramas de la ingeniería:
  - Electrónica: Circuitos Integrados
  - Automoción
- Permite la creación de nuevos elementos mediante la **suma de otras piezas** ya creadas.
- Permite alcanzar **altos niveles de abstracción**.
- En un sistema informático, la primera división en componentes es separar hardware y software. Esta es la primera fase (en muchos casos obvia) del **diseño arquitectónico** o **de alto nivel**.

Componentes y arquitectura (ii)

- Arquitectura (*IEEE 1471):*
  - 'Architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.'
- Arquitectura vs. Diseño:
  - Qué software hay que desarrollar (o comprar o reutilizar) vs. cómo hacerlo.
  - Problema vs. Solución.
  - Pensar en arquitectura implica un mayor nivel de abstracción y granularidad.
  - Ambos vienen determinados por la captura de requisitos funcionales del sistema.

Componentes y arquitectura (iii)

- Para un determinado desarrollo suelen existir varias arquitecturas posibles. Todas con sus **ventajas** y sus **inconvenientes**.
- Variables a tener en cuenta:
  - **Extensión:** facilidad para añadir nuevas propiedades.
  - **Facilidad de cambios:** importante dada la alta posibilidad de cambio en los requisitos.
  - **Simplicidad:** fácil de entender y de implementar.
  - **Eficiencia:** memoria, velocidad...
  - **Seguridad:** autenticación y autorización.

Componentes y arquitectura (iiii)

- Variables (...continuación):
  - **Costes extra**: es más difícil desarrollar un componente genérico → coste extra a corto plazo.
  - Pero una vez construido puede ser fuente de **reutilización** → ahorro a largo plazo, mejora de la **calidad**.
  - Posibilidad de abrir **mercado** → ingresos a largo plazo.
- Balanceo de todas las variables teniendo en cuenta el **¡¡retorno de inversión!! (ROI).**

Componentes y arquitectura (v)

- Descomposición...
  - Qué ha de ser implementado como hardware y qué como software.
  - Cuántos elementos de hardware van a ser necesarios (sus tipos).
  - Cuántos elementos software van a ser necesarios (con qué funcionalidad cada uno).
  - En qué elemento/s de hardware se ejecutará cada elemento de software.
- ... para posteriormente crear aplicaciones mediante el **ensamblado de distintos componentes**.

Componentes y arquitectura (vi)

- El principal problema de los sistemas informáticos es su creciente complejidad.
- Complejidad no en base al número de líneas de código, sino al número de **interrelaciones**.
- Solución:
  - Descomposición del problema en distintos componentes más simples.
  - Conseguir una **fuerte cohesión interna** de los mismos y un **bajo nivel de acoplamiento entre los distintos componentes**.

Componentes y arquitectura (vii)

- La arquitectura determinará el resto del trabajo, de ahí su importancia y la importancia de contar un qué, cómo y por qué.
- Los errores arquitectónicos:
  - No son defectos de programación, sino malas decisiones sobre protocolos, políticas de replicación, concurrencia...
  - Son más difíciles de identificar.
  - No radican en un único lugar.
  - Son más difíciles de solucionar. Su solución puede implicar un cambio drástico de una aplicación.
  - Todo esto redunda en el coste de su mitigación.

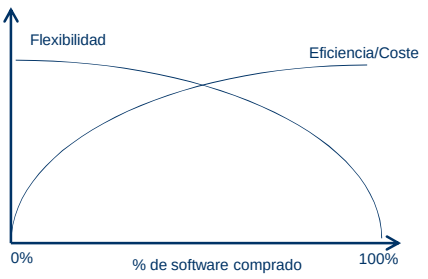
Fabricar todo vs. Adquirir todo

- Antes de desarrollar un nuevo software, hay que decidir en qué lado nos queremos ubicar.
- Fabricar todo el software:
  - Suele resultar un desarrollo más caro.
  - Posibles lagunas conceptuales en zonas no relacionadas con el *core-business* del desarrollador.
  - Módulos aparentemente sencillos como acceso a Web, sistemas de *backup*, conexión con otros sistemas estándar..., pueden convertirse en grandes problemas.
  - La adecuación a normas y estándares es tediosa.
  - Mayor control del desarrollo.
  - Posibilidad de venta.

Fabricar todo vs. Adquirir todo (ii)

- Adquirir todo el software en base a componentes existentes:
  - Precio/tiempos suelen estar negociados de ante-mano, por lo tanto, reducción de riesgos.
  - El mantenimiento del producto se deja al proveedor.
  - Incorpora la adecuación a normas y estándares.
  - La utilización de paquetes estándar aporta muchas ventajas a corto y largo plazo. La más importante: incorporación de las mejores prácticas del mercado.
    - Puede requerir reingeniería de procesos internos.
  - Este software no representa ninguna ventaja significativa sobre el resto de competidores.
  - Búsqueda de la mejor relación calidad/precio.

Fabricar todo vs. Adquirir todo (iii)



Mercado de los componentes (i)

- Un componente debe ser desarrollado mucho más genéricamente que otra pieza de software.
- Además, se requieren más pruebas, documentación, tutoriales y ayudas.
- El desarrollo de componentes para uso interno aporta beneficios a **largo plazo**: estos beneficios suelen no ser vistos por muchos gestores → **involucrar a la dirección**.
- Antes del desarrollo de un componente para su venta se ha de estudiar el mercado: ¿hay clientes dispuestos a comprar?, ¿bajo qué condiciones?

Mercado de los componentes (ii)

- Variables a estudiar del mercado:
  - ¿Este componente está creando nuevo mercado?
    - Implica una gran oportunidad, pero un gran riesgo.
    - Entrada 'evolucionaria' vs. 'revolucionaria'.
  - Si el mercado está maduro, ¿cuántos competidores y qué cualidades e importancia tiene cada uno?
  - ¿Cuál es el tamaño del mercado y qué porcentaje del mismo se espera cubrir?
  - ¿Se conoce suficientemente a los clientes? Generalizar un único (aunque gran) desarrollo puede llevar a desconocer parte de mis potenciales clientes.

Definiciones

- Los términos objeto y componentes son a menudo confundidos.
- Ambos ofrecen funcionalidad mediante una *interfaz*, siguiendo la filosofía de caja negra.

Definiciones. Componente (i)

- Propiedades (Clemens Szyperski):
  - Un componente es una unidad de 'despliegue' independiente.
  - Un componente podría ser utilizado por cualquier organización sin necesidad de ningún conocimiento previo.
  - Un componente no tiene estado persistente en sí mismo, lo tienen (opcionalmente) sus objetos.
- Filosofía: **"Alta cohesión interna, bajo acoplamiento externo"**.
- Si las cumple, podrá ser utilizado para **ensamblado de componentes**.

Definiciones. Componente (ii)

- Otras definiciones:
  - Reutilizable y reemplazable, lo que requiere una interfaz bien definida, y bajo acoplamiento con otras interfaces (Perdita Stevens).
  - Parte física y reemplazable de un sistema que conforma a un conjunto de interfaces y proporciona la realización de dicho conjunto (Los Tres Amigos).

Definiciones. Objeto

- Propiedades (Clemens Szyperski):
  - Es la unidad de **instanciación** con identificador único.
  - Un objeto tiene un estado, el cuál puede almacenarse de forma persistente.
  - Un objeto encapsula su **estado** y **comportamiento**.
- Otras definiciones:
  - Manifestación concreta de una abstracción; entidad con unos **límites bien definidos** e **identidad** que encapsula **estado** y **comportamiento**; instancia de una clase (Los Tres Amigos).

Definiciones. Componente y Objeto

- Un componente suele utilizarse mediante sus objetos y, por lo general, suele estar constituido por más de una clase.
- Los objetos del componente pueden (suelen) atravesar sus fronteras y ser accesible por otros componentes.
- Un componente, además de clases, puede contener otros módulos de código.
- Al igual que una clase puede depender de otra (herencia, asociación,...) un componente puede depender de otro/s.

Definiciones. Módulo

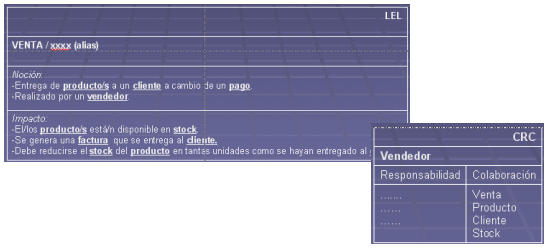
- Populares en lenguajes como Modula-2 o ADA.
- Soportados en algunos lenguajes más modernos (VB 6.0 ó .Net).
- A diferencia de una clase, no puede instanciarse.
- En algunos contextos, un módulo puede verse como un componente que ofrece funcionalidad estática.
- Por esta razón, los antiguos módulos suelen ser vistos en los modernos lenguajes OO como clases con métodos estáticos.

Definiciones. Clase (i)

- Descripción de un conjunto de objetos que comparten los mismos **atributos**, **relaciones** y **semántica** (Los Tres Amigos).
- Forman parte del **vocabulario (dominio) del problema** o de la **solución:**
  - Fichas LEL (Léxico Extendido del Lenguaje).
- Es importante determinar exactamente las **responsabilidades** de una clase dentro de un componente:
  - Fichas CRC (Clase-Responsabilidad- Colaboración).

Definiciones. Clase (ii)

- **LEL:** Léxico Extendido del Lenguaje
- **CRC:** Clase – Responsabilidad – Colaboración



Definiciones. Interfaz (i)

- Se aplican a distintas ‘abstracciones’ como **clases** o **componentes**.
- Colección de un conjunto de **operaciones** que se utiliza para especificar un **servicio** de una **clase** o **componentes** (Los Tres Amigos).
- Marcan la separación de **qué** labor se lleva a cabo y de **cómo** se lleva a cabo.
- Evitan la necesidad de grandes cambios ante un pequeño cambio en otro componente/clase.
- Permiten el **trabajo en grupo** con sólo definir las distintas interfaces.

Definiciones. Interfaz (ii)

- Un componente puede soportar múltiples interfaces:
  - Por soportar múltiples responsabilidades (servicios).
  - Por soportar múltiples versiones de la misma responsabilidad.
- Pueden ser vistos como un **contrato** entre un cliente y un suministrador de software.
- Un componente debe minimizar el número de **dependencias** hacia otros componentes. Pero esto puede llevar a componentes muy pesados.
- Las interfaces llevan al bajo acoplamiento entre componentes.

Definiciones. Caja negra y blanca

- Se refieren a cómo de visible es la implementación que se esconde detrás de un *interfaz*.
- En una caja negra, no se aporta (ni se necesita) ningún detalle sobre su implementación.
- Caja negra o blanca influyen en la futura reutilización del componente. Una nueva versión del mismo software podría no ser fácil de implantar en los clientes si era visto como una caja blanca.

Bibliografía

- Component Software. Beyond Object-Oriented Programming (Capítulos 1, 2 y 4)
  Clemens Szyperski
  Addison-Wesley, 1998
- Software Engineering. An Object-Oriented Perspective (Capítulo 5)
  Eric J. Braude
  John Wiley & sons, Inc, 2001
- El Lenguaje Unificado de Modelado
  Grady Booch, James Rumbaugh, Ivar Jacobson
  Addison Wesley, 1999

Diseño Basado en Componentes

Introducción

Ingeniería Informática
Universidad Carlos III de Madrid