

# *Fundamentos de Ingeniería del Software*



## *Capítulo 11. Reutilización del software*

# *Reutilización del software.*

## *Estructura*



1. Reutilización del software
2. Beneficios de la reutilización
3. Dificultades para la reutilización
4. Assets
  - Caracterización de los assets
  - Tipos de assets
5. Niveles de reutilización
6. Modelo de procesos con reutilización
7. Ingeniería de dominios
  - Análisis del dominio
  - Fases del análisis del dominio
8. Desarrollo basado en componentes
  - Concepto de componente
  - Cuestiones técnicas
  - Estándares para el software de componentes
  - Cuestiones metodológicas
9. Clasificación y recuperación de assets o componentes
  - Clasificación enumerada
  - Clasificación por facetas
  - Clasificación de atributos y valores

# *Reutilización del software.*

## *Bibliografía*



- (Pressman 98) Roger S. Pressman. "Ingeniería del Software – Un enfoque práctico". 4<sup>a</sup> Edición. Ed. Mc Graw-Hill. 1998. Capítulo 26.

# 1. Reutilización del software

- Idea vieja (reutilización *ad hoc*).
- “Cualquier procedimiento que produce o ayuda a producir un sistema mediante el nuevo uso de algún elemento procedente de un esfuerzo de desarrollo anterior” (Freeman 87) ≠ **visión de (Meyer 98)**
- Inicialmente, simple combinación de componentes de código almacenados en una biblioteca
  - (reutilización del código, sin método)
  - ⇒ enfoque muy simple
  - ⇒ *¿Qué se reutiliza? ¿Cómo?*

## 2. Beneficios de la reutilización

- “La reutilización es la única aproximación realista para llegar a los índices de productividad y calidad que la industria del sw. necesita” (Mili et al. 95).
- Mejora de la productividad:
  - Disminución tiempo de desarrollo:  
*⇒ mejor adaptación requisitos cambiantes*  
*¡Los requisitos no son estables!*
  - Disminución de costes
- Mejora de la calidad del sw.:
  - Mayor fiabilidad
  - Mayor eficiencia (aunque al principio pueda parecer que no)

# *Beneficios de la reutilización (II)*



Para Bertrand Meyer (Meyer 98):

- Oportunidad
- Disminución de esfuerzos de mantenimiento
- Fiabilidad
- Eficiencia
- Consistencia (ósmosis estilo programación)
- Inversión (preservar el "know-how")

# 3. Dificultades para la reutilización

- En muchas empresas no existe plan de reutilización (no se considera prioritario)
  - Escasa formación
  - Resistencia del personal
  - Pobre soporte metodológico
    - uso de métodos que no promueven la reutilización (*p.ej. estructurados*)
    - Necesarios métodos para:
      - *desarrollo para reutilización*
      - *desarrollo con reutilización*
- ⇒ *¿Quién soporta los gastos adicionales de la reutilización?*

# 4. Assets



- Inicialmente, sólo se contemplaba la reutilización de código.
- El concepto de reutilización ha evolucionado hacia la idea de que todo el conocimiento y productos derivados de la producción de software son susceptibles de ser reutilizados en la construcción de nuevos sistemas.
- Se puede reutilizar mucho más que código fuente:
  - ⇒ beneficios mayores al reutilizar diseños y documentación asociada al código fuente reutilizable
- *Asset* o “elemento software reutilizable”:
  - cualquier producto software obtenido en el ciclo de vida del software, con independencia de su nivel de abstracción:
    - ⇒ p.ej. especificaciones, diseños, código, pruebas, documentación, etc.
- Nótese las diferencias con (Meyer 98):
  - Meyer propugna centrarse en el código, afirmando que los componentes han sido la mayor contribución a la reutilización
  - Propone reducir el salto entre diseño y codificación, asegurar la consistencia de forma natural al usar el mismo lenguaje

# *Caracterización de los assets* (García et al. 97)

## ■ *Atributos:*

- Identificador.
- Nombre.
- Versión.
- Fecha creación.
- Fecha modificación.

## ■ *Subsistemas:*

- Descriptor:
  - implementar el mecanismo de clasificación.
- Cualificación:
  - métricas de calidad
    - Complejidad
    - Cohesión
    - Acoplamiento
- Administrativo:
  - autores, accesos, etc.
- Técnico:
  - herramienta, formato, etc.
- Documentación.
- Pruebas.

# *Tipos de assets*



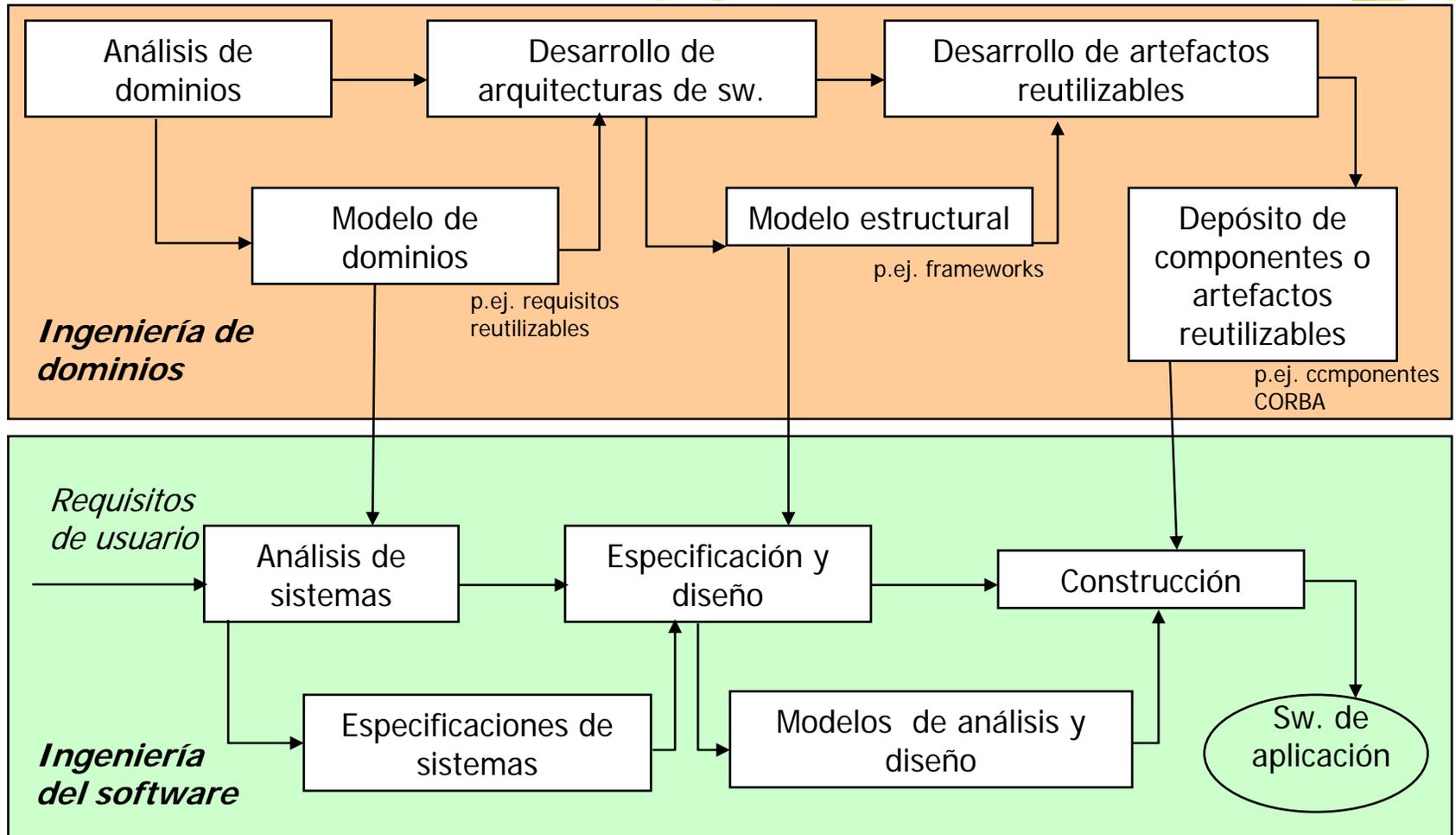
Un asset puede encapsular cualquier abstracción útil producida durante el desarrollo de software

- Planes de proyecto.
- Estimaciones de coste.
- Arquitectura.
- Especificaciones y modelos de requisitos.
- Diseños.
- Código fuente.
- Documentación de usuario y técnica.
- Interfaces persona-máquina.
- Datos.
- Casos de prueba.

# 5. Niveles de reutilización

- Reutilización de código
  - librerías de funciones, editores, inclusión de ficheros, mecanismos de herencia en POO, **componentes**, etc.
- Reutilización de diseños
  - no volver a inventar arquitecturas
    - p.ej. patrones de diseño
    - p.ej. patrones arquitectónicos (C/S, pipeline, OO, etc.)
- Reutilización de especificaciones
  - reutilización de las abstracciones del dominio
  - podría estar asociada a la generación (semi)automática de los elementos de diseño e implementación.
- Elevar el nivel de abstracción  $\Rightarrow$  reutilización  $\uparrow\uparrow$ 
  -  Asset como subsistema agregación de varios componentes atómicos a distintos niveles de abstracción  $\Rightarrow$  *mecano*

# 6. Modelo de procesos con reutilización (Pressman 98) p.490



# 7. *Ingeniería de dominios*



- Objetivo: identificar, construir, catalogar y diseminar un conjunto de artefactos de sw. que tienen interés dentro de un dominio de aplicación.
- Dominio: conjunto de sistemas relacionados.
- No ligado a ningún proyecto de sw.
- Ingeniería de dominios:
  - Análisis
  - Construcción
  - Diseminación

# *Análisis del dominio*



- El proceso de identificar y crear un conjunto de componentes reutilizables que puedan ser usados en los sistemas desarrollados en un dominio.
- “El análisis de dominio del sw. es la identificación, análisis y especificación de **requisitos comunes de un dominio** de aplicación específico, normalmente para su reutilización en múltiples proyectos dentro del mismo dominio de reutilización” (Firesmith 93).
- Interesante a largo plazo en áreas de producto consideradas como estratégicas:
  - Bajo costo, mejor calidad y menor tiempo de comercialización.

# *Fases del análisis del dominio* (Berard 93) (Pressman 98) p.391



- Definir el dominio a investigar.
- Extraer elementos del dominio:
  - OO: especificaciones, diseños y código; bibliotecas de componentes ya desarrolladas; casos de prueba.
  - No OO: políticas, procedimientos, planes, estándares, métricas, y componentes no OO.
- Clasificar los elementos extraídos del dominio.
- Recolectar una muestra representativa de aplicaciones del dominio.
  - La aplicación debe tener elementos dentro de las categorías definidas.
- Analizar cada aplicación dentro de la muestra.
  - Identificar objetos candidatos reutilizables.
- Desarrollar un modelo de análisis para los objetos.
  - Servirá como base para el diseño y construcción de los objetos del dominio.

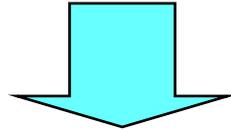
# 8. Desarrollo basado en componentes (Pressman 98) p.494

- Los sistemas de software actuales deben ser *abiertos*:
  - distintas redes
  - distintas plataformas hardware y software
  - requisitos cambian constantemente



OO por si solo no es suficiente

*"los sistemas OO han fracasado en su promesa de mejorar la reutilización de software"*



Preferible considerar cada aplicación como una instancia de una clase genérica de aplicaciones, construida a partir de un conjunto de *componentes de software reconfigurables*.

# Componente



- Concepto más general que el de objeto:
  - “Unidades binarias de producción, adquisición y uso independiente, que interaccionan para formar un sistema”  
(Szyperski 98)
  - “Abstracción estática con conectores (*plugs*)”  
(Nierstrasz Tsichritzis 95)
- No todas las abstracciones que deberían evolucionar con los cambios en los requisitos de una aplicación son necesariamente objetos
  - ⇒ se postula que es posible adaptarse con más facilidad a los requisitos cambiantes en un paradigma orientado a componentes que en uno OO, “desconectando” y reconfigurando sólo los componentes afectados.

# *Desarrollo basado en componentes.*

## *Cuestiones técnicas*

---

- *¿Qué mecanismos deben utilizar los lenguajes y entornos de programación para soportar el desarrollo basado en componentes?*
  - objetos, clases, herencia, sistema de tipos, concurrencia y persistencia
- *¿Cuál es la estructura de un componente?*
- *¿Cuáles son los mecanismos para combinar componentes, adquiridos de distintas fuentes, internas y externas?*
  - ↳ necesidad de estándares para componentes

# *Estándares para el software de componentes*



- Imprescindibles para desarrollar mercados de componentes:
  - Modelos comunes que permiten interoperar
  - Especificaciones de interfaz concretas
    - ⇒ permiten la composición
  - Arquitecturas generales
- Estándares:
  - OMG (*Object Management Group*): OMA (*Object Management Architecture*) y CORBA
  - Microsoft: COM / DCOM / OLE 2.0 / ActiveX / .NET
  - Sun: Java y JavaBeans
  - Linux: modelos de componentes KParts y Bonobo para los escritorios KDE y GNOME, respectivamente

# *Desarrollo basado en componentes. Cuestiones metodológicas*



- *¿Cómo y dónde se obtienen los componentes en el ciclo de vida del software?*
  - Ingeniería de componentes/Ingeniería de dominios
- *¿Cómo deben modificarse los métodos y modelos de proceso?*
- *¿De dónde provienen los componentes?*
  - Un componente debe haber sido diseñado para ser utilizado en composición con otros componentes.
  - Un componente normalmente no se diseña por separado, sino que forma parte de un *framework* de componentes que colaboran entre sí.

# 9. Clasificación y recuperación de assets o componentes

- Aspecto crítico en reutilización.
- Repositorio: BD de información compartida sobre los elementos que se producen o se usan en un desarrollo software.
- Tendencia actual: repositorios que permitan el trabajo conjunto de equipos de desarrollo localizados en diferentes lugares geográficos a través de Internet.

⇒ Supongamos un repositorio con miles de componentes reutilizables...

- ¿Cómo puede hallar en él el ingeniero el elemento que necesita?
- ¿Cómo se describen los componentes de sw. en términos no ambiguos y fácilmente clasificables?
- Se han desarrollado varios esquemas de clasificación y recuperación...

# Clasificación enumerada

- Estructura jerárquica que define clases y subclases de componentes
- Los componentes en sí son las hojas
- Necesaria labor de ingeniería de dominio previa
- Sencillo de comprender y utilizar

## Ejemplo:

(Pressman 98)

```
operaciones de ventana
  visualización
    abrir
      basados en menú
        open Window
      basados en sistema
        sysWindow
    cerrar
      a través de puntero
        ...
  cambio de tamaño
    a través de órdenes
      setWindowSize, stdResize, shrinkWindow
    por arrastre
      pullWindow, stretchWindow
  ...
```

# *Clasificación por facetas*



- Se analiza una cierta área de dominio y se identifica un conjunto de características descriptivas básicas (priorizadas): *facetas*
- Se define un descriptor de facetas:
  - p.ej. {función, tipo de objeto, tipo de sistema}
- Para cada faceta, conjunto de palabras reservadas:
  - p.ej. *función*=(copiar, desde) ó  
*función*=(copiar, sustituir, todo)
- Se puede incorporar *tesauro*  $\Rightarrow$  uso de sinónimos
- Más fácil de extender y adaptar que clasificación enumerada

# *Clasificación de atributos y valores*



- Se define un conjunto de atributos para todos los componentes de una cierta zona del dominio.
- Parecido a una clasif. por facetas, pero:
  - no hay límite para el n° de atributos que se pueden utilizar.
  - no se asignan prioridades a los atributos.
  - no se utiliza la función de tesaurus.