|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **9** |
|  | TEMA |

Diseño de componentes

[9.1] ¿Qué es un componente?

[9.2] Diseño de componentes basados en clase

[9.3] Realización del diseño de componentes

[9.4] Diseño de componentes tradicionales

[9.5] Desarrollo basado en componentes

[9.6] Referencias

Esquema



Ideas clave

9.1. ¿Qué es un componente?



Dependiendo del punto de vista:

* **Orientación a objetos**: el componente es un conjunto de clases interrelacionadas.
* **Tradicional**: el componente es un elemento funcional del programa que contiene lógica de negocio, estructuras de datos y una interfaz ser invocado.
* **Proceso**: el componente es un módulo de software que se recoge en un catálogo de componentes para ser reutilizado.

9.2. Diseño de componentes basados en clases

Hay cuatro principios básicos aplicables al diseño de componentes:

* **Principio abierto-cerrado**: el componente debe ser abierto para la extensión pero cerrado para la modificación.
* **Principio de sustitución de Liskov**: las subclases deben ser sustituibles por sus clases de base.
* **Principio de inversión de la dependencia**: se debe depender de las abstracciones, no de las concreciones.
* **Principio de segregación de la interfaz**: es mejor tener muchas interfaces específicas del cliente que una sola de propósito general.

En referencia a la agrupación de los componentes en subsistemas o paquetes se debe tener en cuenta los siguientes principios:

* **Principio de equivalencia de la liberación de la reutilización**: el gránulo de reutilización es el gránulo de liberación.
* **Principio de cierre común**: las clases que cambian juntas deben pertenecer a la misma agrupación.
* **Principio de la reutilización**: las clases que no se reutilizan juntas no deben agruparse juntas.

Todo componente debe tener un **alto grado de cohesión**, es decir, que solo contendrá atributos y operaciones relacionadas con la funcionalidad que desarrolla.

Todo componente debe tener un **bajo grado de acoplamiento**. Esto se logra reduciendo la interdependencia entre clases.

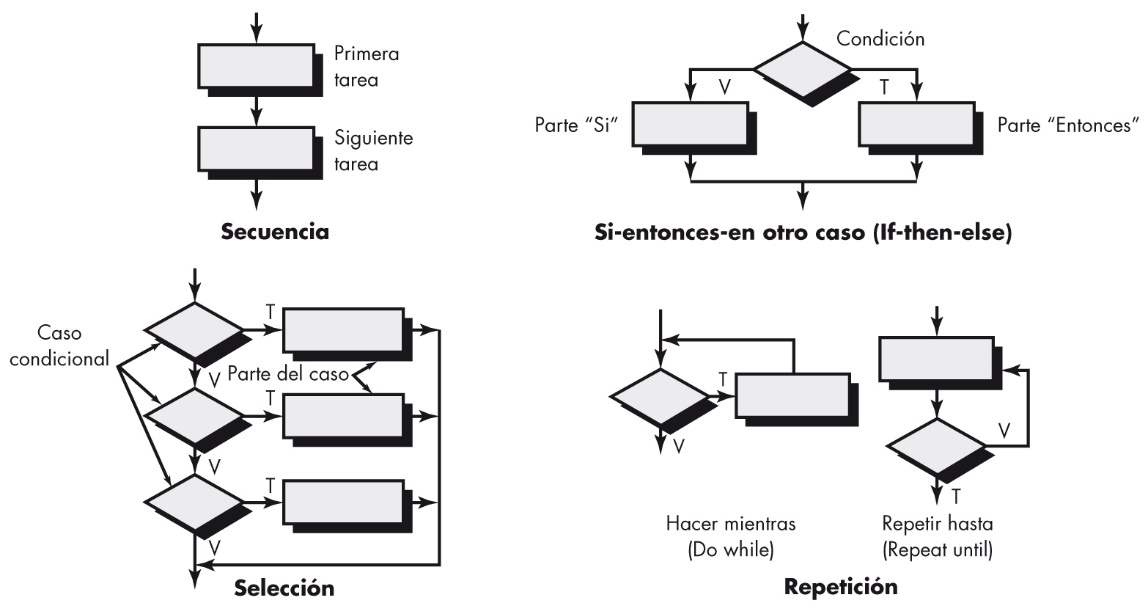
9.3. Realización del diseño en el nivel de componentes

Las tareas a llevar a cabo para el diseño de los componentes son:

1. Identificar las **clases de diseño** que correspondan con el **dominio del problema**.
2. Identificar las **clases de diseño** que correspondan con el **dominio de la infraestructura**.
3. **Elaborar** las **clases de diseño**:
   1. Especificar detalles de los mensajes cuando colaboren con otras clases o componentes.
   2. Identificar sus interfaces.
   3. Elaborar los atributos y definir tipos y estructuras de datos.
   4. Describir en detalle el flujo del procesamiento dentro de cada operación.
4. Describir las **fuentes persistentes de datos** (bases de datos y archivos) e identificar las **clases necesarias** para administrarlos.
5. Desarrollar y elaborar las **representaciones del comportamiento** para cada clase o componente.
6. Elaborar **diagramas de despliegue** para dar más detalles de la implementación.

9.4. Diseño de componentes tradicionales

Para la representación gráfica de la funcionalidad que debe desarrollar un componente se utilizan los **diagramas UML de actividad** o los **diagramas de flujo**.



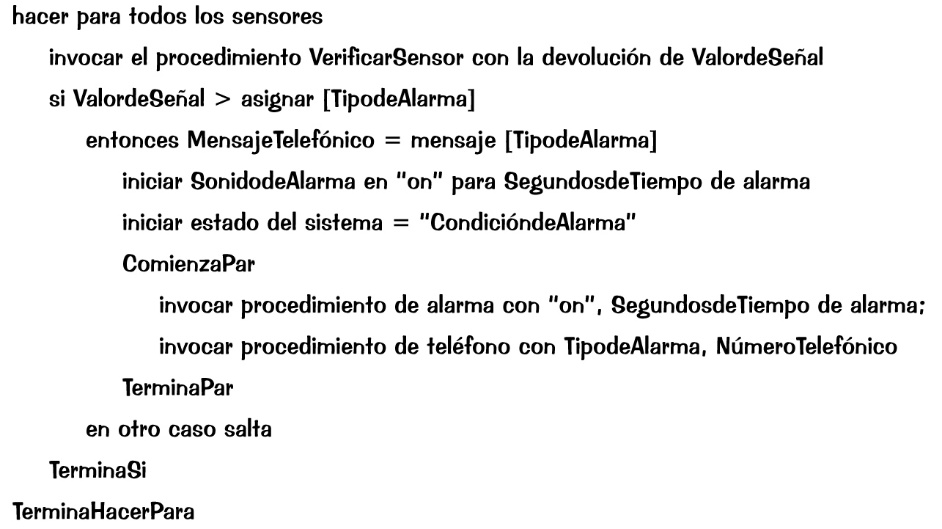
Fuente: Pressman, R. S. (2010). *Ingeniería del software.   
Un enfoque práctico* (7ª ed.) (p. 254). México: McGraw Hill.

Las **tablas de decisión** permiten representar las combinaciones complejas de condiciones y las acciones a realizar en cada caso.



Fuente: Pressman, R. S. (2010). *Ingeniería del software.   
Un enfoque práctico* (7ª ed.) (p. 255). México: McGraw Hill.

El **seudocódigo** permite representar, con un lenguaje intermedio ente el lenguaje que entendemos las personas y el que entienden las computadoras, la funcionalidad del componente.



Fuente: Pressman, R. S. (2010). *Ingeniería del software.   
Un enfoque práctico* (7ª ed.) (p. 256). México: McGraw Hill.

9.5. Desarrollo basado en componentes

La ingeniería de *software* basada en componentes propone el diseño y construcción de sistemas reutilizando componentes ya existentes.

Para que esto sea viable, es necesario:

* Realizar ingeniería de dominio de componentes para identificar, catalogar y difundirlos.
* Calificar la idoneidad del componente, identificar las adaptaciones necesarias y las posibles combinaciones con otros componentes.
* Realizar el análisis y diseño para la reutilización del componente.
* Tener una biblioteca que permita la clasificación y recuperación de componentes.

9.7. Referencias

Pressman, R. (2010). *Ingeniería del software* (7ª ed.). México: McGrawHill.

Lo + recomendado

No dejes de leer…

**Desarrollo de software basado en componentes**

Artículo de carácter tutorial en el que se presentan los conceptos fundamentales de la reutilización de software, los modelos de proceso y los aspectos metodológico que caracterizan la producción de software basada en componentes.

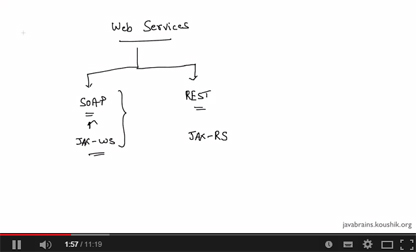
Accede al artículo desde el aula virtual o a través de la siguiente dirección web:

<http://juancol.me/rsrc/sw-basado-en-comp-CAC2003.pdf>

No dejes de ver…

**Introducción a los servicios web**

Interesante tutorial de iniciación sobre servicios web.



Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://www.youtube.com/watch?v=mKjvKPlb1rA>

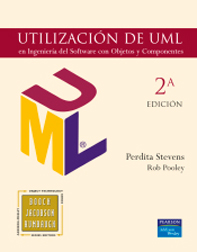
+ Información

A fondo

**UML con objetos y componentes**

Perdita, S. & Pooley, R. (2007). *Utilización de UML en ingeniería del software con objetos y componentes*. Madrid: Prentice Hall.

Libro que presenta las mejores prácticas en el diseño y desarrollo de software, con una visión amplia de la materia, que permite a los lectores ver por sí mismos cómo pueden ser apropiados para diferentes situaciones diferentes prácticas. El libre se divide en cuatro bloques que comprenden: conceptos de introducción, el lenguaje UML, casos de estudio y la aplicación de UML en la práctica.



**W3C. Guía breve de servicios web**

Los servicios web son componentes web interoperables basados en estándares XML (SOAP, WSDL, UDDI, etc.). En esta guía rápida se presentan los principales elementos participantes en la tecnología de servicios web.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:

<http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb>

Recursos externos

PSeInt. Herramienta de seudocódigo y diagramas de flujo

PSeInt es una herramienta gratuita *opensource* que utiliza un seudolenguaje en castellano y dispone de un editor de diagramas de flujo.

Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<http://pseint.sourceforge.net/>

StarUML

StarUML es una herramienta de modelado UML gratuita compatible con UML 2 que soporta once tipos de diagramas UML: clases, objetos, casos de uso, componentes, despliegue, secuencia, comunicación, actividad, etc.



Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<http://sourceforge.net/projects/staruml/>

Documentación: <https://docs.staruml.io/>

Test

**1.** Un componente es:

A. Un módulo de *software*.

B. Un módulo que puede desplegarse o sustituirse con independencia del resto de la arquitectura *software*.

C. Un módulo accesible mediante un conjunto de interfaces.

D. Todas las anteriores son correctas.

**2.** El principio abierto-cerrado para el diseño de componentes propone:

A. Que debe ser abierto para los cambios y cerrado para las rigideces.

B. Que debe ser abierto para los usuarios de confianza y cerrado para potenciales intrusos.

C. Que debe ser abierto para la extensión y cerrado para la modificación.

D. Que debe ser abierto para el *software* y cerrado para el *hardware*.

**3.** En referencia a la agrupación de componentes:

A. El gránulo de reutilización debe ser el gránulo de liberación.

B. Las clases que cambian juntas deben de separarse en diferentes agrupaciones.

C. Las clases que se no se reutilizan juntas deben agruparse juntas.

D. La agrupación de componentes solo se recomienda para componentes que se reutilicen internamente en una organización.

**4.** Señala cuál es la respuesta correcta respecto al diseño correcto de un componente:

A. Alta cohesión y alto acoplamiento.

B. Alta cohesión y bajo acoplamiento.

C. Baja cohesión y alto acoplamiento.

D. Baja cohesión y bajo acoplamiento.

**5.** Cuando se elabora una clase de diseño se debe:

A. Documentar con comentarios a pie de página.

B. Describir atributos, operaciones, mensajes e interfaces.

C. Enumerar el listado de potenciales objetos.

D. Representar las acciones que conlleven polimorfismo.

**6.** En el diseño de componentes tradicionales, para representar las combinaciones complejas de condiciones y las acciones a realizar en cada caso se utiliza:

A. Tablas de decisión.

B. Diagramas de flujo.

C. Diagramas de actividad.

D. Diagramas de estado.

**7.** La ingeniería de *software* basada en componentes propone:

A. El desarrollo del mayor número posible de componentes.

B. La creación de un catálogo de usuarios de cada componente.

C. La construcción de sistemas reutilizando componentes ya existentes.

D. La implementación de los componentes en diferentes lenguajes de programación para que sean más reutilizables.

**8.** El seudocódigo:

A. Debe evitarse en la fase de diseño para evitar condicionar a los programadores.

B. Es como se denomina a un fragmento de código, que contiene errores, escrito en un determinado lenguaje de programación.

C. Permite representar la funcionalidad de un componente.

D. Debe compilarse y ejecutarse en máquinas con una configuración semejante a las que se despliegue la aplicación final.