Ingeniería de Sistemas y Computación

Arquitectura de Software
Modificabilidad

Universidad de los Andes

Agenda del día

- Introducción
- Tácticas arquitecturales
- Patrones arquitecturales
- Patrones de diseño detallado

Universidad de los Andes

¿Qué es modificabilidad?

- Atributo de calidad que tiene que ver con el **costo** del cambio y la **facilidad** con que un sistema se acomoda a los cambios.

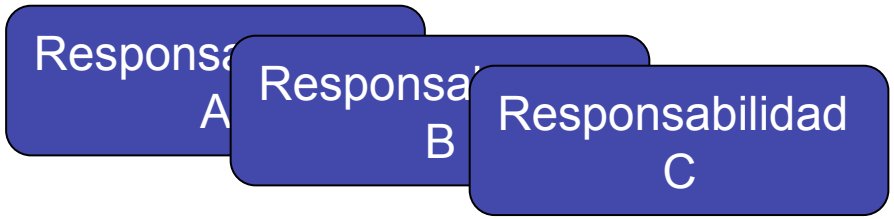
- Según ISO:
  - Mantenibilidad
  - Portabilidad

Modelo de modificabilidad

- **Supuestos**
  - **Un sistema cumple con responsabilidades.** Una responsabilidad es una acción, un conocimiento que debe ser mantenido, o una decisión a ser tomada por un sistema.
  - La relación entre responsabilidades se define como **acoplamiento** entre la dos. El **grado de acoplamiento** entre dos responsabilidades se mide como la probabilidad en la que el cambio de una responsabilidad implica un cambio en la otra.

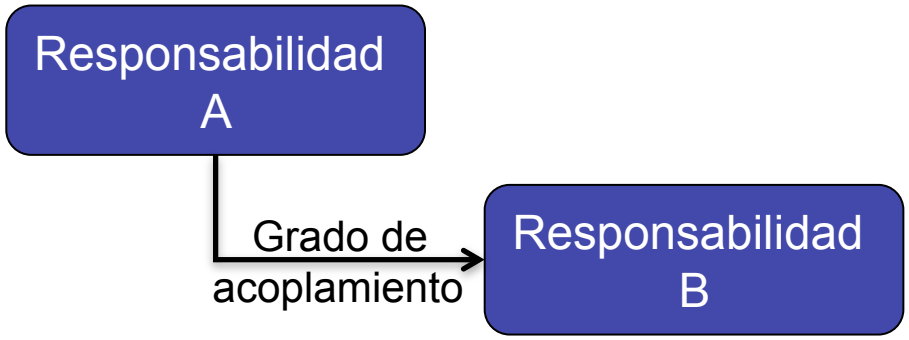
Parámetros a tener en cuenta (1)

- **Costo promedio de modificar una responsabilidad**
  - i.e. Reducir el costo de modificar una responsabilidad A y que esto no implique cambios en otras responsabilidades, reduce el costo de cualquier modificación que implique cambios en A.



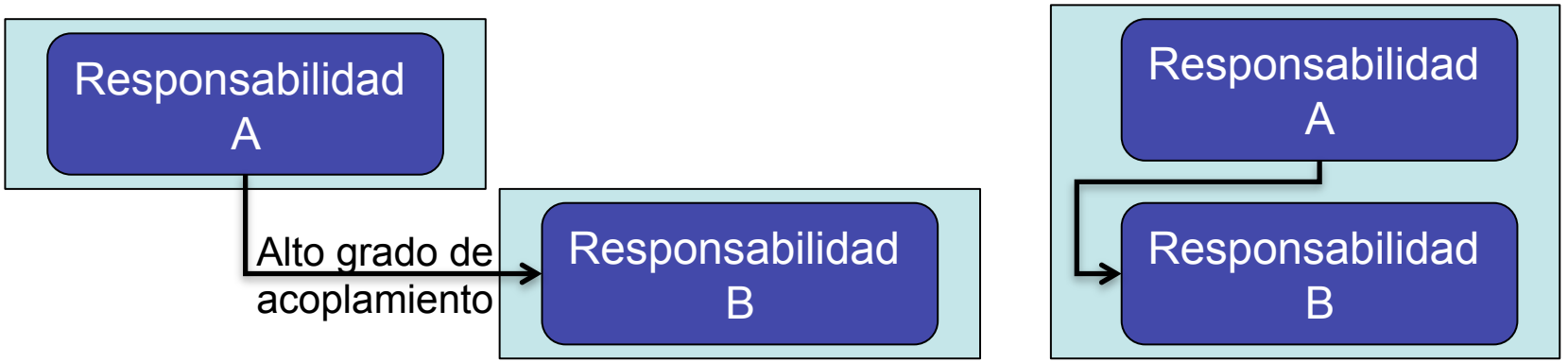
Parámetros a tener en cuenta (2)

- **Grado de acoplamiento**
  - i.e. Reducir el acoplamiento entre dos responsabilidades A y B reduce el costo de una modificación en la responsabilidad A.



Parámetros a tener en cuenta (3)

- **Cohesión**
  - Las responsabilidades son asignadas a módulos del sistema. Si una responsabilidad A tiene un alto grado de acoplamiento con la responsabilidad B, el costo de cambiar A es menor si las dos responsabilidades son asignadas al mismo módulo.



Parámetros a tener en cuenta (4)

- ¿Cuándo se hace la modificación en el ciclo de vida de construcción de la aplicación?
  - i.e. Los cambios tempranos en el sistema son menos costosos que los cambios hechos más adelante.

Universidad de los Andes

Agenda del día

- Introducción
- Tácticas arquitecturales
- Patrones arquitecturales
- Patrones de diseño detallado

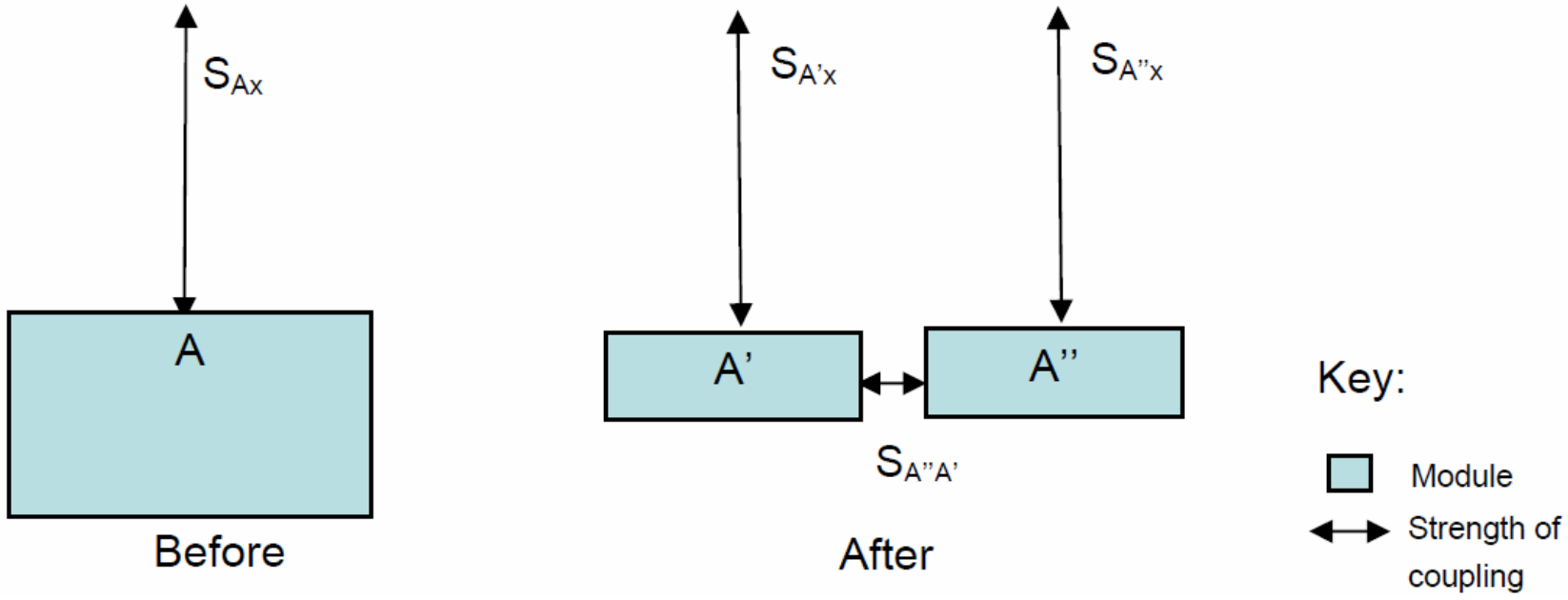
Universidad de los Andes

Tácticas arquitecturales

- Son decisiones de diseño que afectan los (4) parámetros en el modelo de modificabilidad

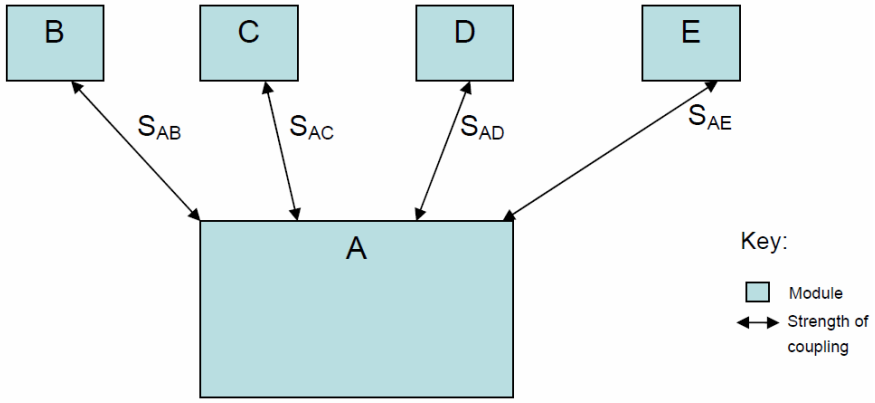
Reducir el costo promedio de modificar una responsabilidad

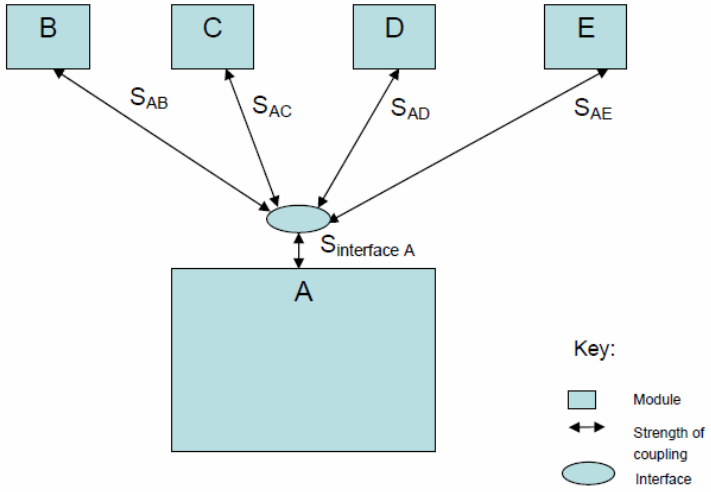
- Reducir el costo partiendo la responsabilidad en dos.



Reducir el grado de acoplamiento (1)

- **Encapsulation/Information Hiding**
  - Reducir la probabilidad que un cambio en un módulo se propague a otros por medio de una interface.





Universidad de los Andes

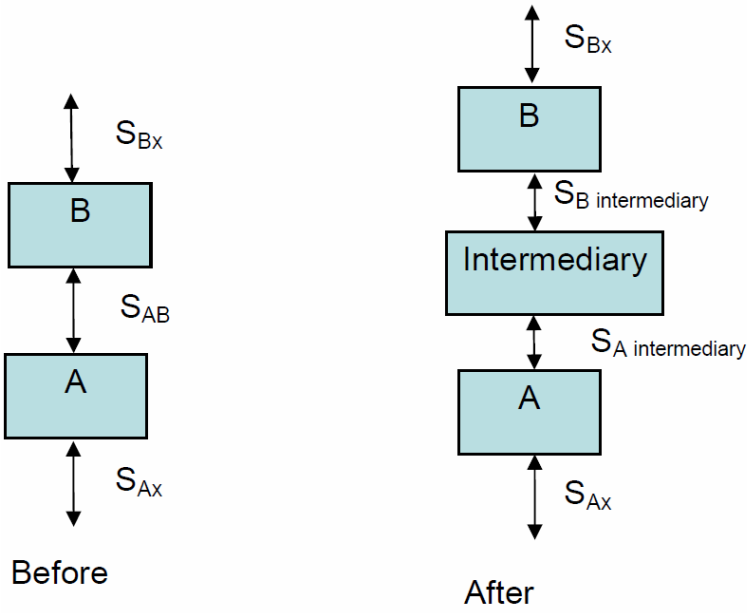
Reducir el grado de acoplamiento (2)

- **Wrappers**
  - Una forma de encapsulamiento
  - Es una interface para un módulo que transforma o controla información para un módulo.
  - “Encapsulation is intended to hide information, and transformations may be used as a portion of the hiding strategy”

Reducir el grado de acoplamiento (3)

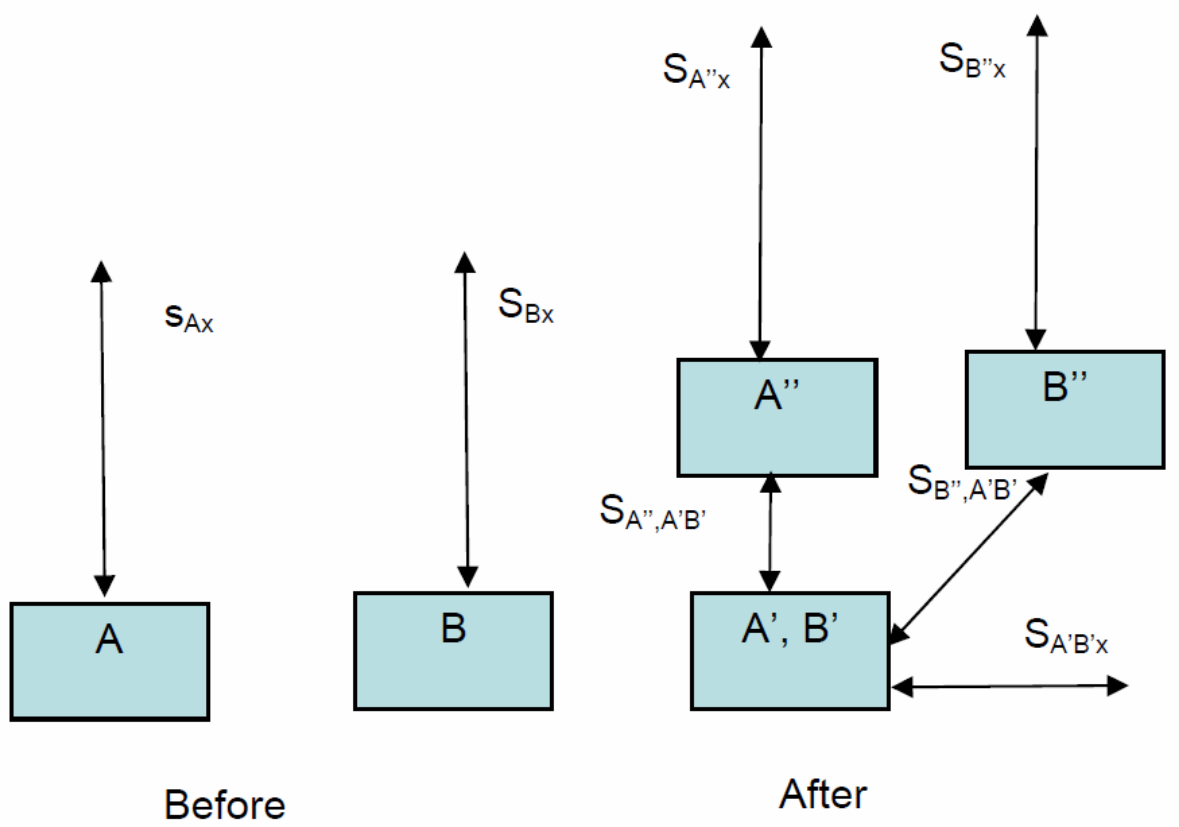
- **Subir de nivel de abstracción**
  - Parametrizar las actividades de una responsabilidad

- **Usar intermediarios**



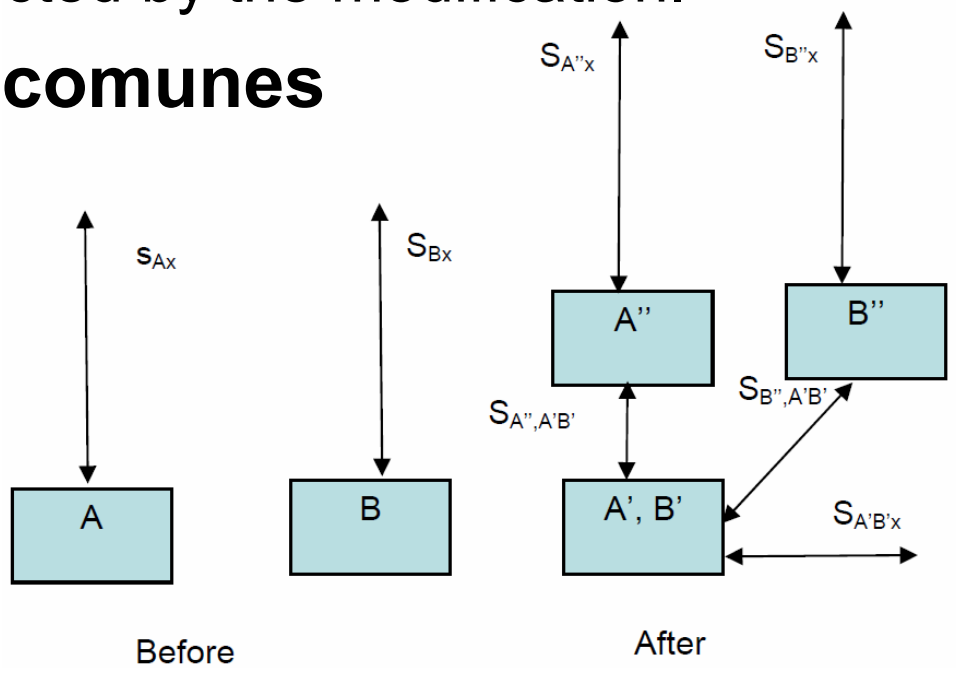
Aumentar cohesión (1)

- Mover responsabilidades de un módulo a otro

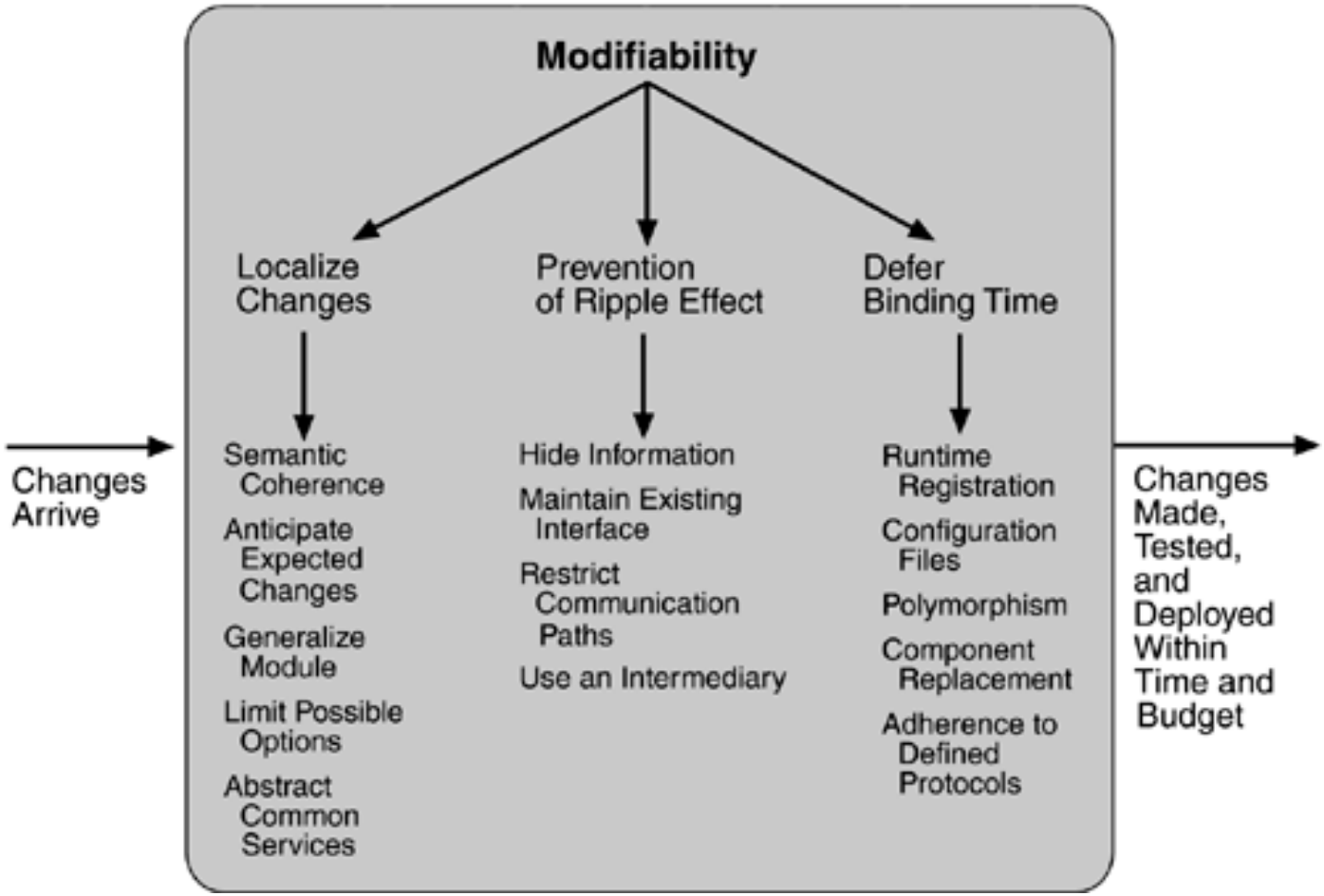


Aumentar cohesión (2)

- **Mantener la coherencia semántica**
  - The cost of modifying A' (B') is less than the cost of modifying A (B).
  - A'' and B'' are unaffected by the modification.
- **Abstraer servicios comunes**



Tácticas arquitecturales (Otra visión)



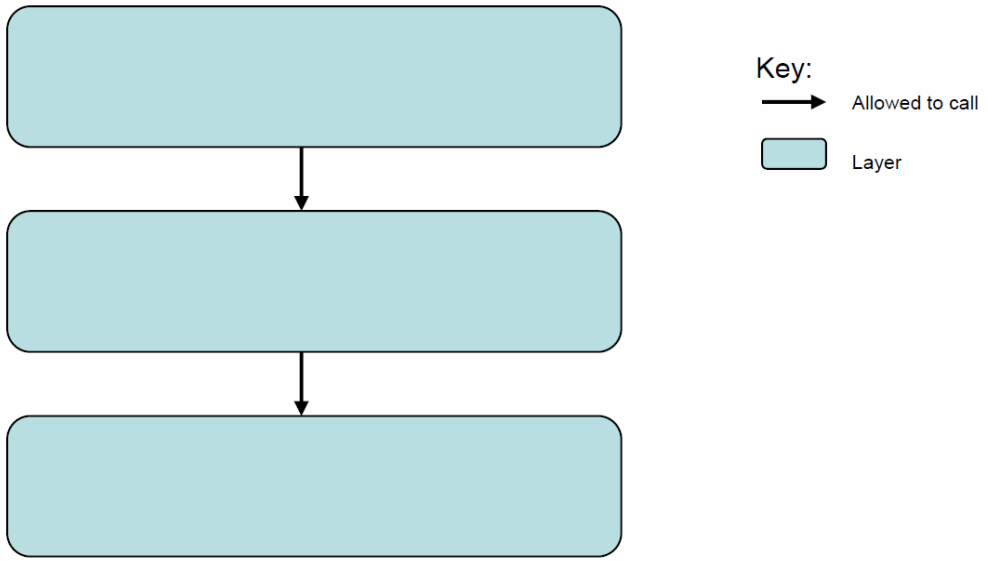
Universidad de los Andes

Agenda del día

- Introducción
- Tácticas arquitecturales
- Patrones arquitecturales
- Patrones de diseño detallado

Capas (Layer Pattern)

- "The Layers architectural pattern helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a **particular level of abstraction**"



Universidad de los Andes

Capas (Layer Pattern)

- Mantener la consistencia semántica
- Aumentar el nivel de abstracción
- Abstraer servicios comunes
- Usar encapsulamiento
- Usar un intermediario

Pipes and Filter Pattern

- “The Pipes and Filters architectural pattern provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component. Data is passed through pipes between adjacent filters. Recombining filters allows you to build families of related systems.”

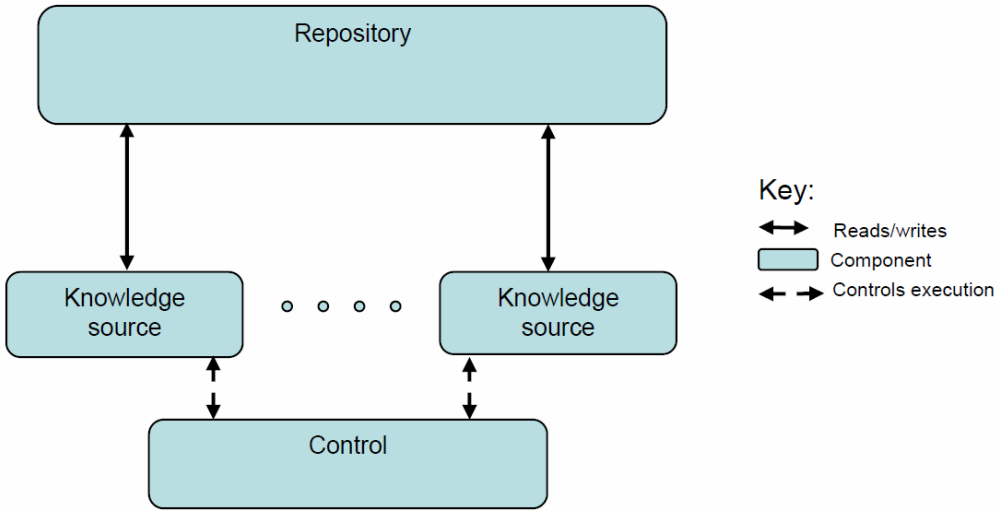
Universidad de los Andes

Pipes and Filter Pattern

- Mantener la consistencia semántica
- Usar encapsulamiento
- Usar un intermediario
- Usar lazy instantiation

Blackboard Pattern

- “The Blackboard architectural pattern is useful for problems for which no deterministic solution strategies are known. Several specialized subsystems assemble their knowledge to build a possibly partial or approximate solution.”



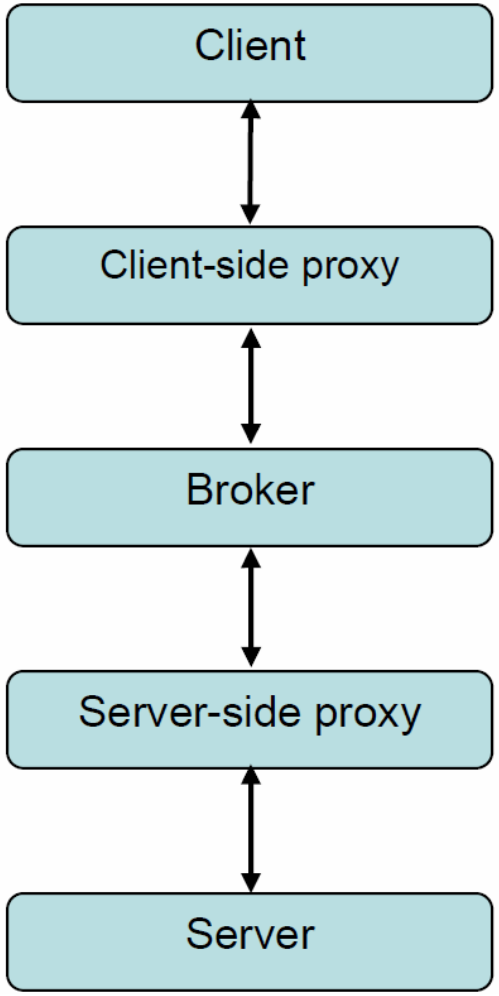
Universidad de los Andes

Blackboard Pattern

- Mantener la coherencia semántica
- Usar un intermediario
- Subscripción en tiempo de ejecución

Broker Pattern

- “The Broker architectural pattern can be used to structure distributed software systems with decoupled components that interact by remote service invocations.
- A broker component is responsible for coordinating communication, such as forwarding requests, as well as for transmitting results and exceptions.”



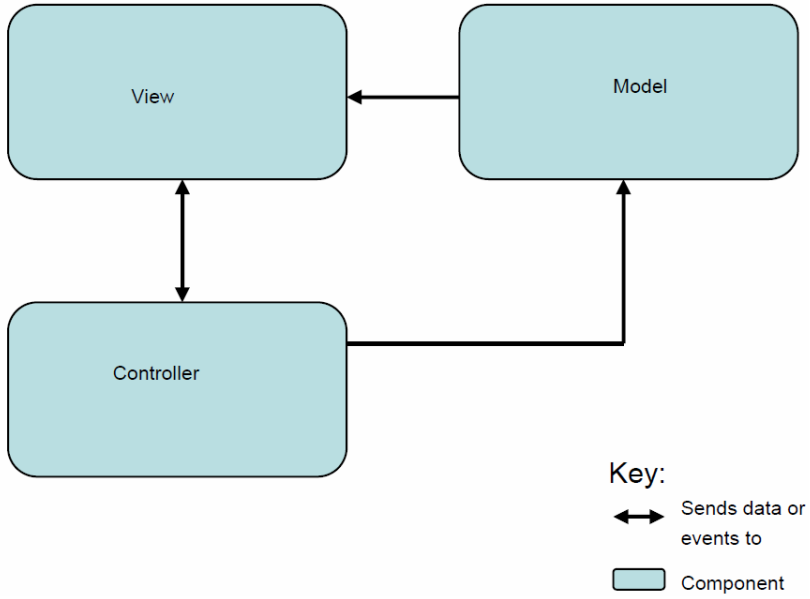
Universidad de los Andes

Broker Pattern

- Mantener la coherencia semántica
- Usar encapsulamiento
- Usar un intermediario
- Subscripción en tiempo de ejecución

MVC Pattern

- "The Model-View-Controller architectural pattern (MVC) divides an interactive application into three components. The model contains the core functionality and data. Views display information to the user. Controllers handle user input. Views and controllers together comprise the user interface. A change-propagation mechanism ensures consistency between the user interface and the model."



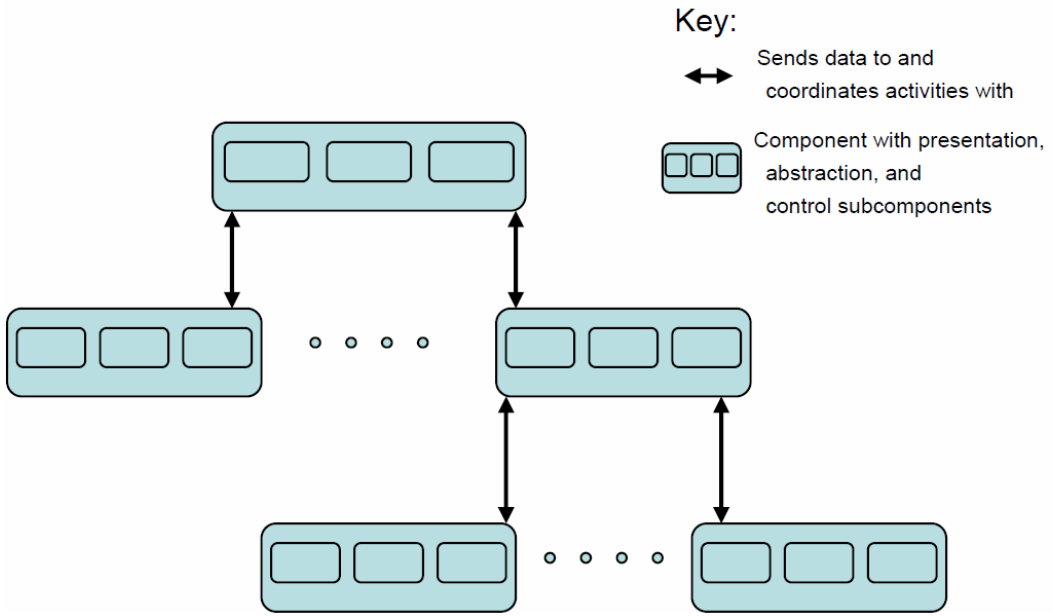
Universidad de los Andes

MVC Pattern

- Mantener coherencia semántica
- Usar encapsulamiento
- Usar un intermediario
- Runtime binding

Presentation-Abstraction-Control Pattern

- “This pattern defines a structure for interactive software systems in the form of a hierarchy of cooperating agents. Every agent is responsible for a specific aspect of the application’s functionality and consists of three components: presentation, abstraction, and control. This subdivision separates the human-computer interaction aspects of the agent from its functional core and its communication with other agents.”



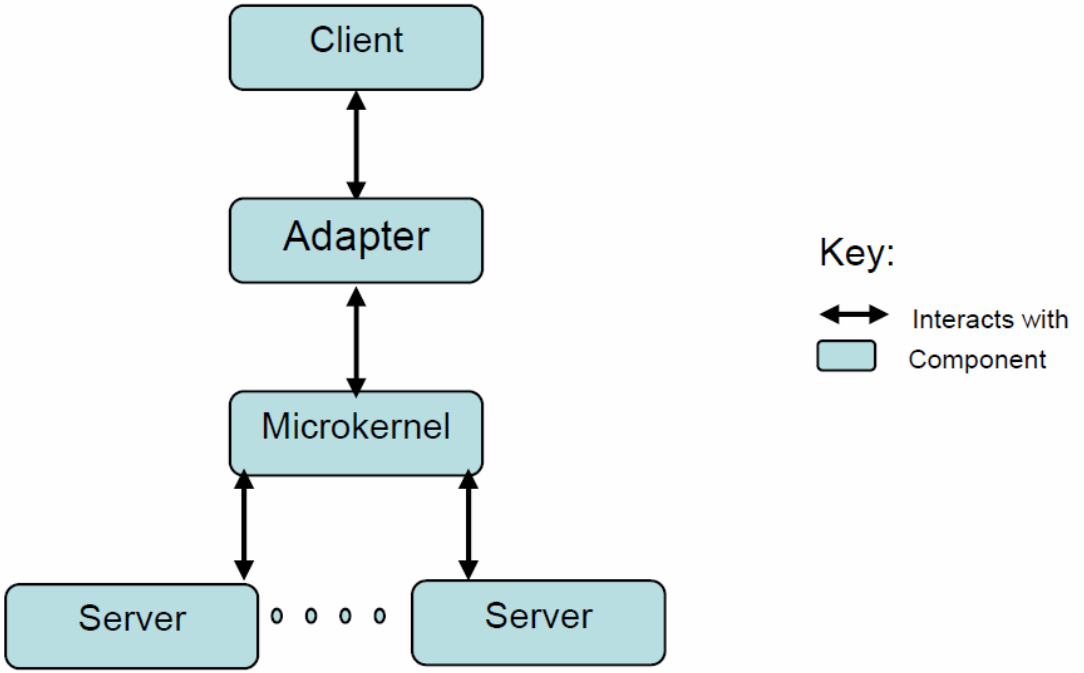
Universidad de los Andes

PAC Pattern

- Mantener la coherencia semántica
- Aumentar el nivel de abstracción
- Usar encapsulamiento
- Usar un intermediario

Microkernel Pattern

- "The Microkernel architectural pattern applies to a software system that must be able to adapt to changing system requirements. It separates a minimal functional core from extended functionality and customer-specific parts. The microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration."



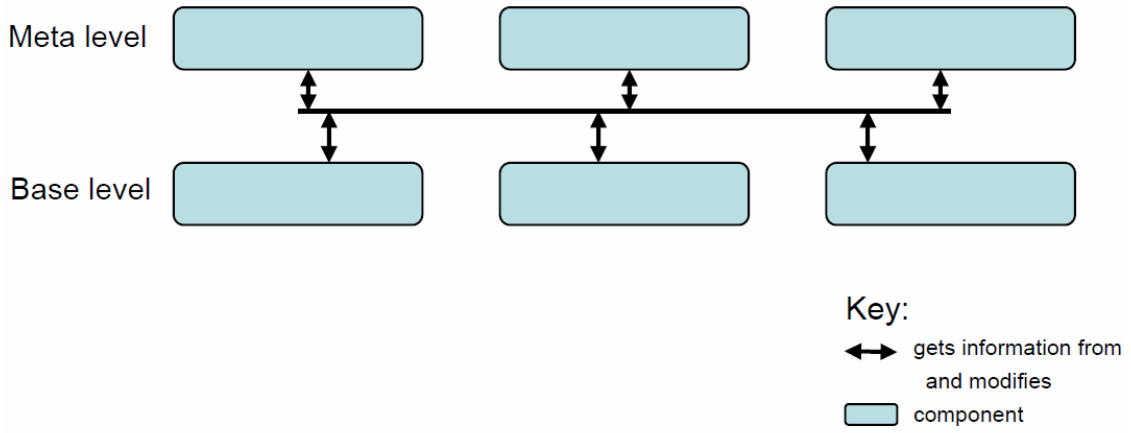
Universidad de los Andes

Microkernel Pattern

- Mantener la coherencia semántica
- Abstraer servicios comunes
- Usar encapsulamiento
- Usar un intermediario

Reflection Pattern

- "The Reflection architectural pattern provides a mechanism for changing the structure and behavior of a software system dynamically. It supports the modification of fundamental aspects, such as type structures and function call mechanisms. In this pattern, an application is split into two parts. A meta level provides information about selected system properties and makes the software self-aware. A base level includes the application logic. Its implementation builds on the meta level. Changes to information kept in the meta level affect subsequent base-level behavior."



Universidad de los Andes

Reflection Pattern

- Usar coherencia semántica
- Usar encapsulamiento

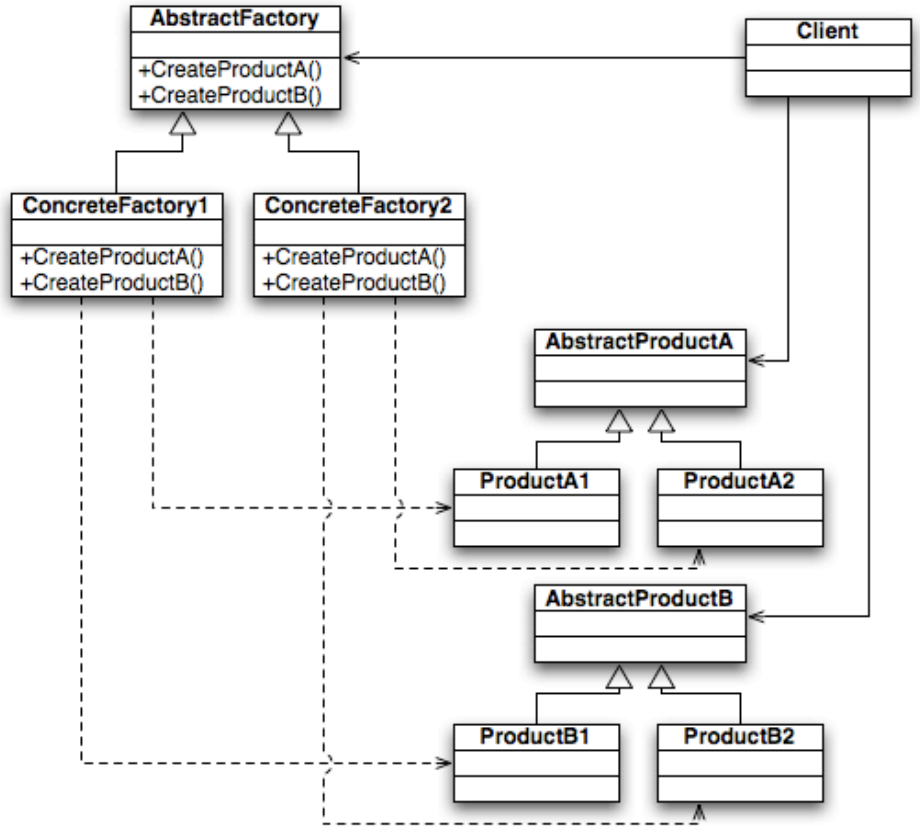
Universidad de los Andes

Agenda del día

- Introducción
- Tácticas arquitecturales
- Patrones arquitecturales
- Patrones de diseño detallado

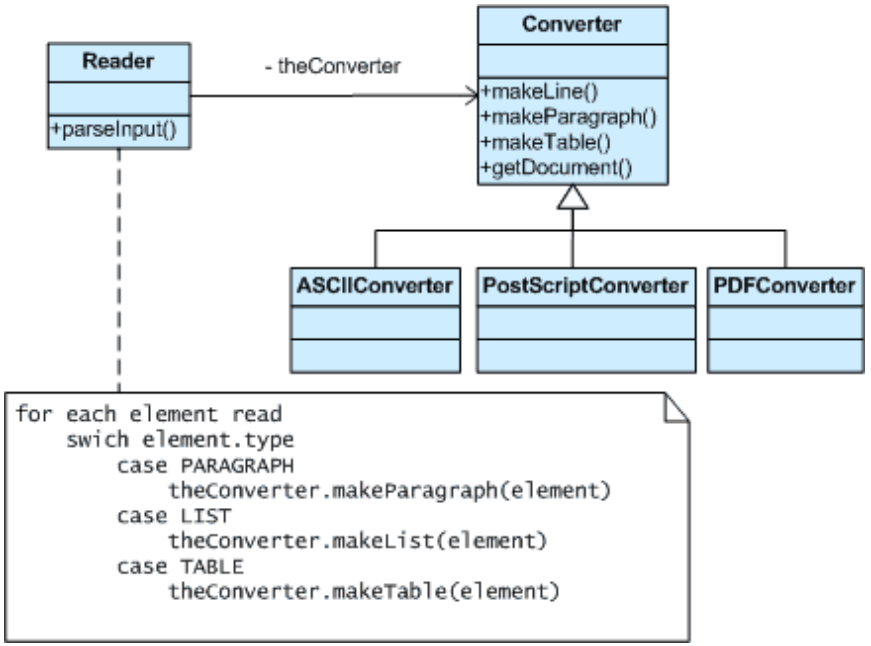
Abstract Factory

- Provee una interface para crear familias de objetos relacionados sin especificar la clase a la que pertenecen



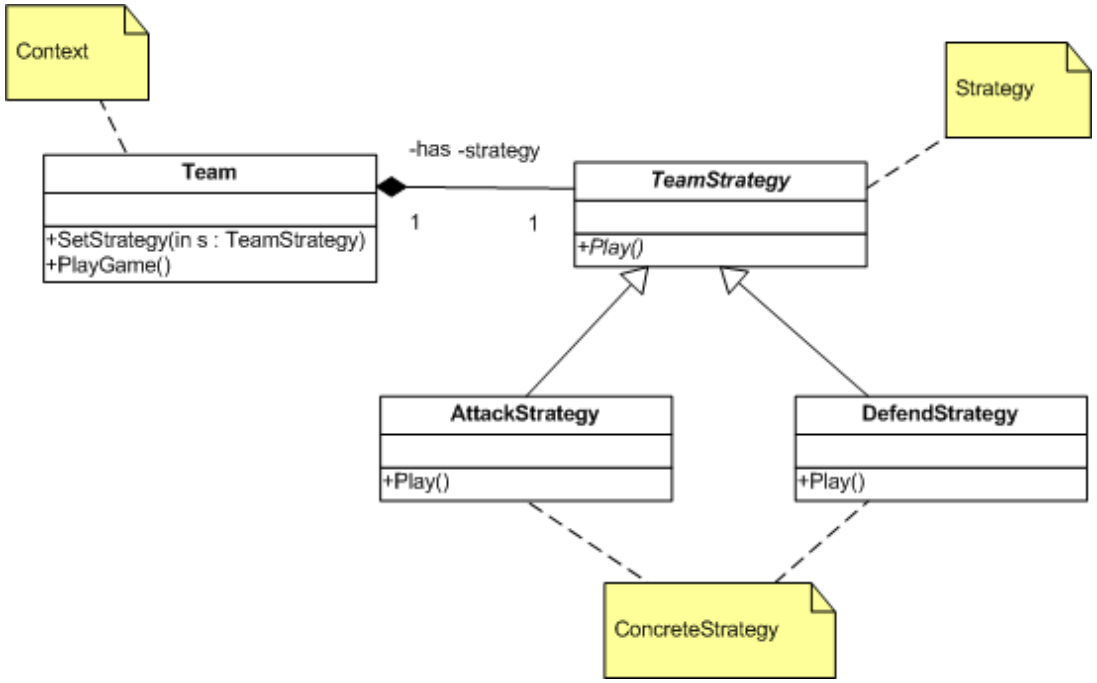
Builder

- Separa la construcción de objetos complejos de su representación, para que el mismo proceso de construcción pueda crear diferentes representaciones.



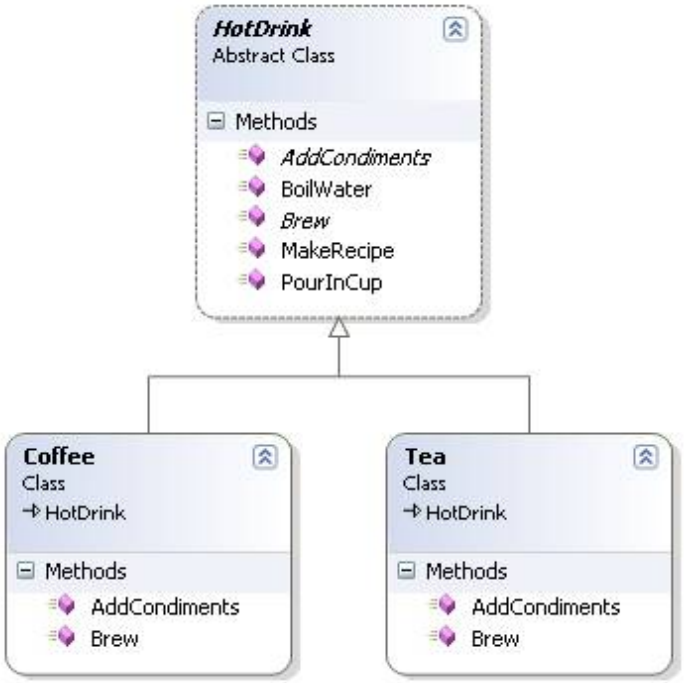
Strategy Pattern

- Define una familia de algoritmos, encapsulados, e intercambiables



Template Method

- Definir el esqueleto de un algoritmo en métodos y darle responsabilidad a diferentes clases de la implementación.



¿Preguntas?

Universidad de los Andes

Referencias

- Len Bass, Paul Clements, Rick Kazman. **Software Architecture in Practice**, Second Edition
- Len Bass, Paul Clements, Rick Kazman. **Aspectos Avanzados en Arquitectura de Software.** Universidad de los Andes, Curso de Verano 2010, Bogotá, Colombia
- Frank Buschmann, Kevlin Henney, Douglas C. Schmidt. **Pattern-Oriented Software Architecture (POSA), A Pattern Language for Distributed Computing**. Volume 4. 2007