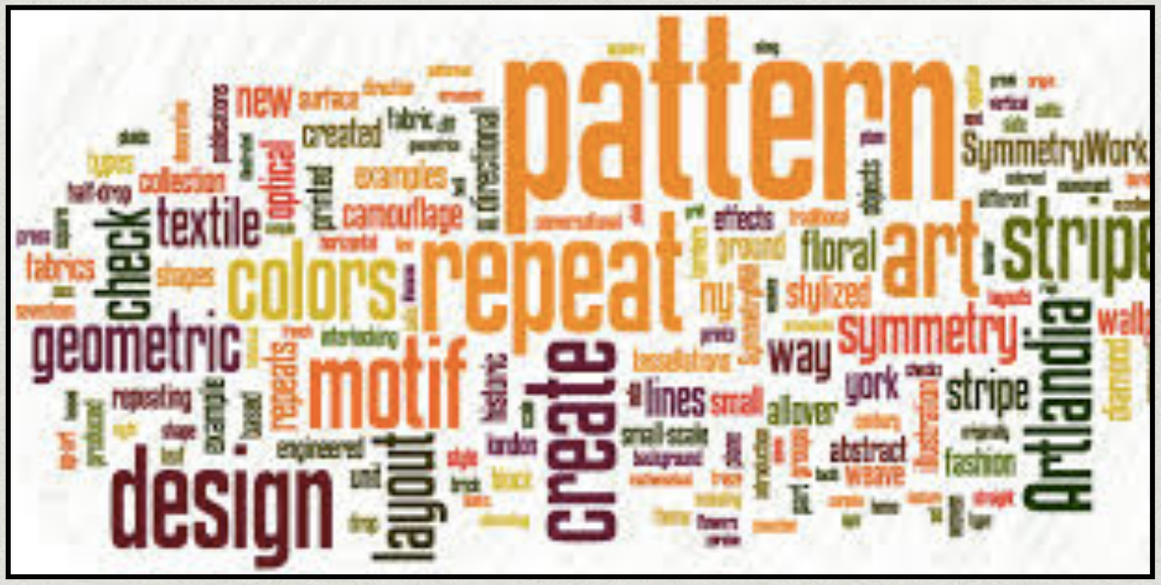
SOFTWARE DESIGN

MATERIALS: CHAPTERS 12-16 PRESSMAN'S BOOK SOFTWARE ENGINEERING: A PRACTITIONER'S APPROACH

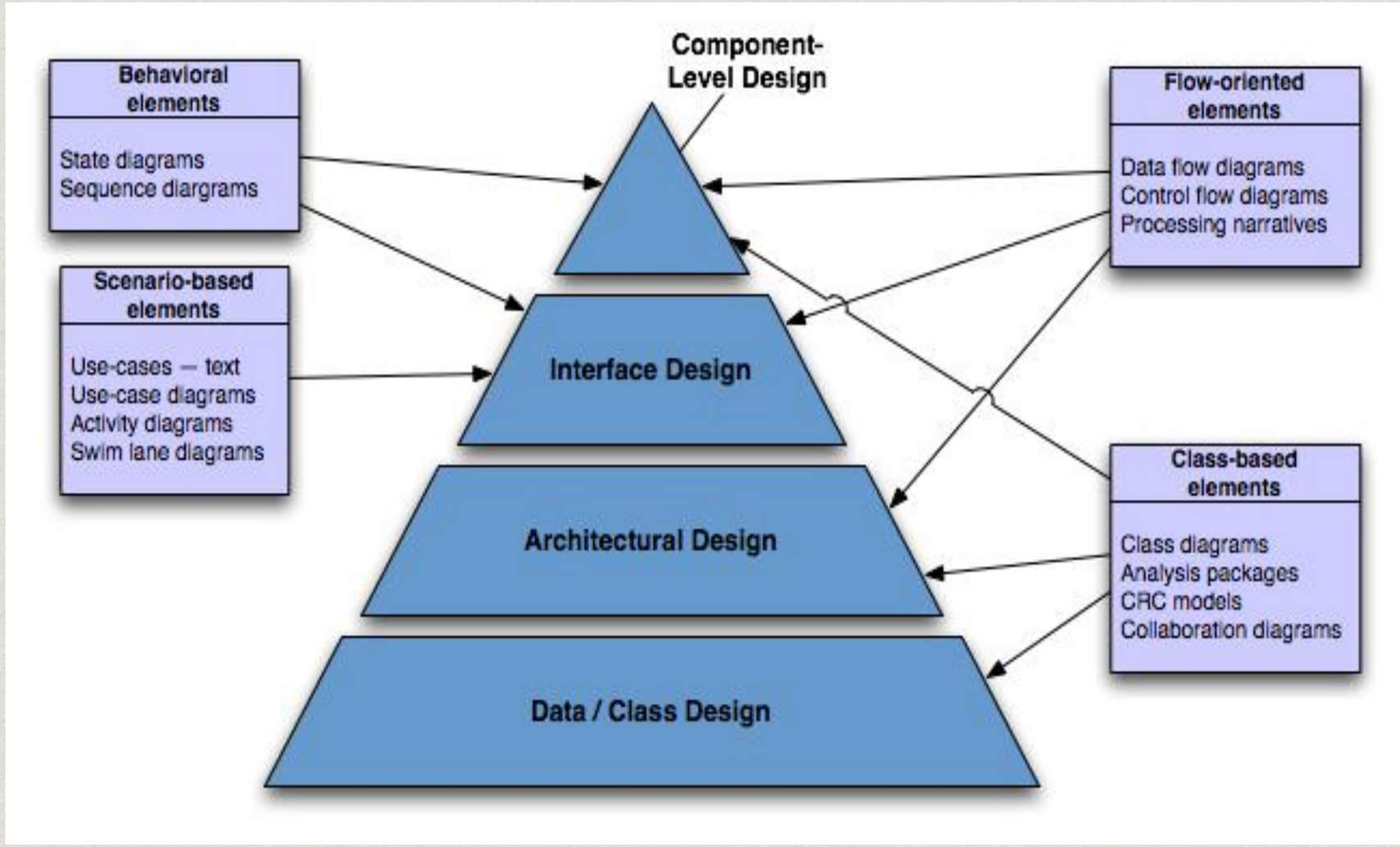
What is Design?

- A blue-print for the software product
- Designing is **NOT** Coding
- Should not start coding before having a good idea of the design, or will need to step back to refactor



Components of the Design Model

- Architecture Design
- Interface Design
- Component Design
- Class Design



Why is Design Important?

Can we skip this step?

Design Process: Goal

- Establish the design model
- Should be complete
- Should satisfy requirements
- Should be documented, readable and clear

Design Quality

- **Functionality**: A design should support all the functions needed given the requirements. It should be complete and functional.
- **Modularity**: A design should be modular; that is, the software should be logically partitioned into elements or subsystems
- **Modifiability**: A design should enable change easily and effectively
- **Extensibility**: A design should be flexible to enable additions of new features easily
- **Reliability/Maintainability**: A design should enable quick identification of problems and fixes.
- **Reusability**: A design should enable the reuse of its components

Characteristics of good designs

- A design should exhibit an architecture that
  - as been created using recognizable architectural styles or patterns,
  - is composed of components that exhibit good design characteristics and
  - can be implemented in an evolutionary fashion

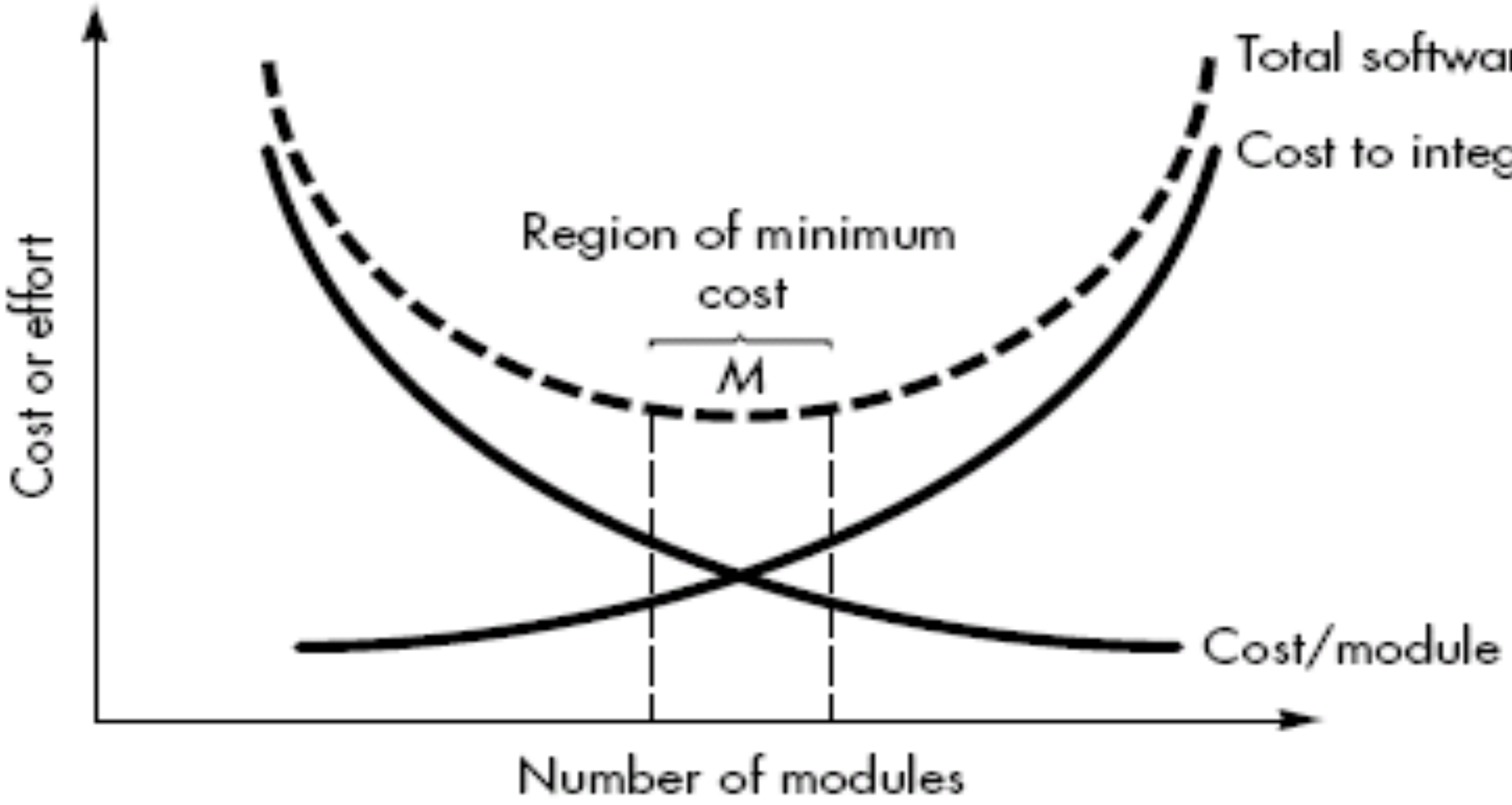
- A design should contain distinct representations of data, architecture, interfaces, and components.

- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.

- A design should lead to components that exhibit independent functional characteristics

Let’s take a look at the Code and Design Review Document

Modularity



So how do you design for good modularity?

Achieving good modular design

Effective modular design consist of three things:

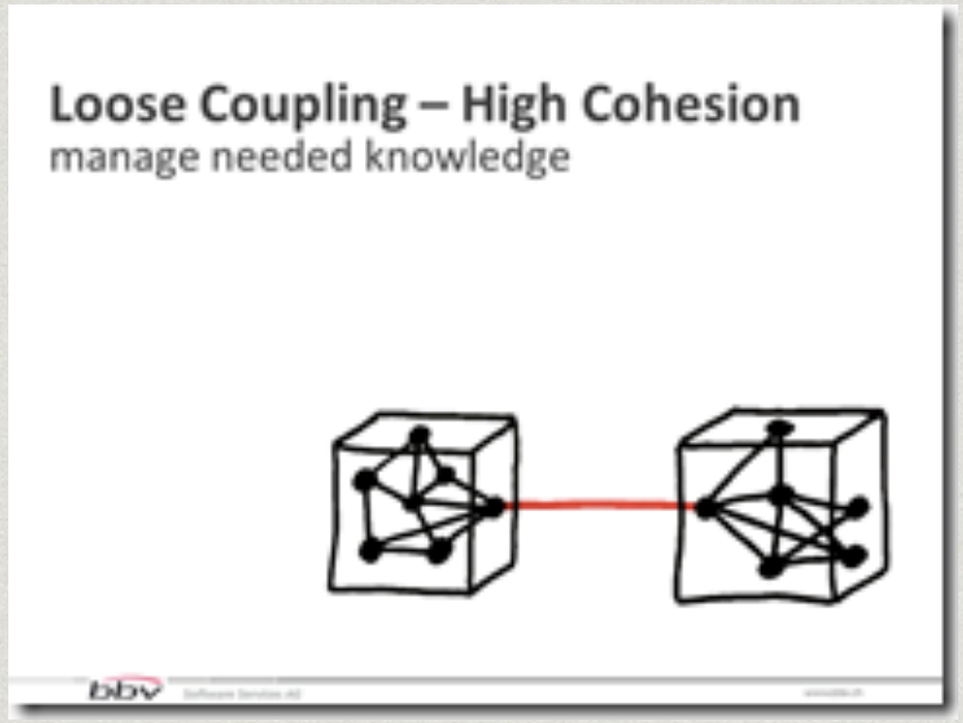
- Functional Independence
- Cohesion
- Coupling

Functional Independence

- Functional independence is achieved by developing modules with "single-minded“ function and an "aversion" to excessive interaction with other modules.

- In other words - each module addresses a specific sub-function of requirements and has a simple interface when viewed from other parts of the program structure.

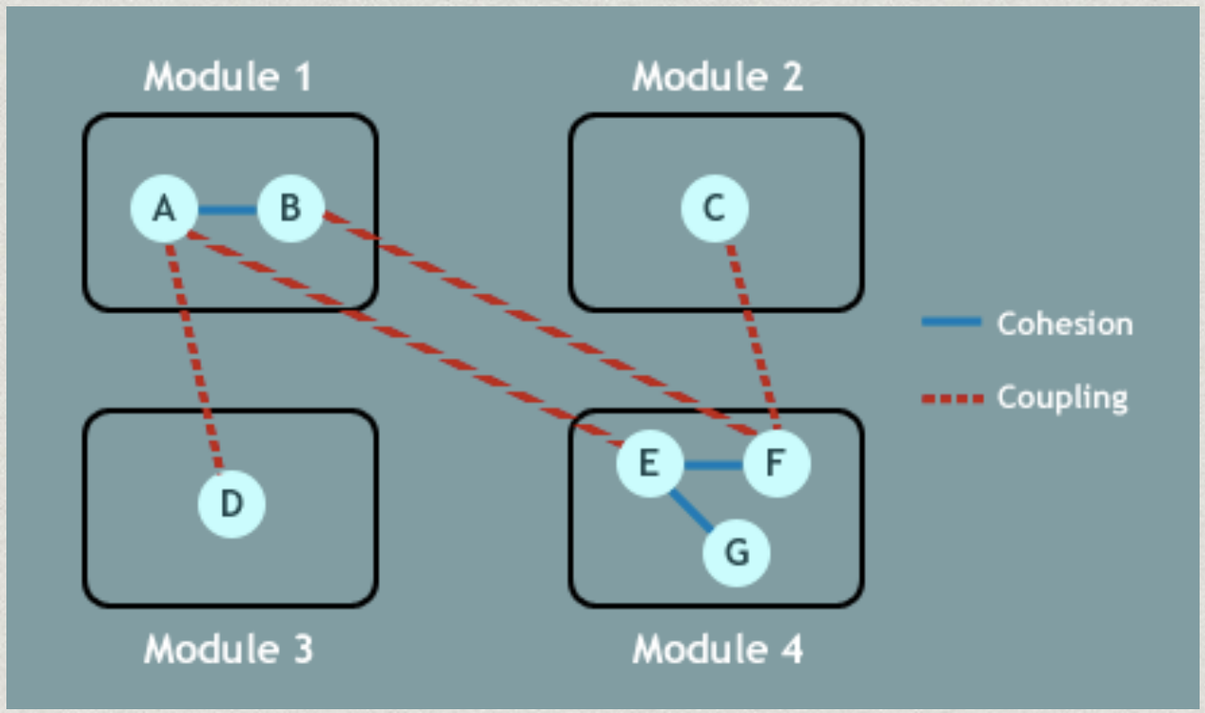
- Independence is important –
  - Easier to develop
  - Easier to Test and maintain
  - Error propagation is reduced
  - Reusable module



Coupling and Cohesion

Coupling: is the degree of interdependence between modules or software components

Cohesion: is the degree to which elements in the module belongs together



Architecture Design

- Software Architecture is: “The software architecture of a program or computing system is the structure or structures of the system, which comprise the software components, the externally visible properties of those components, and the relationships among them.”

Architectural Styles and Patterns

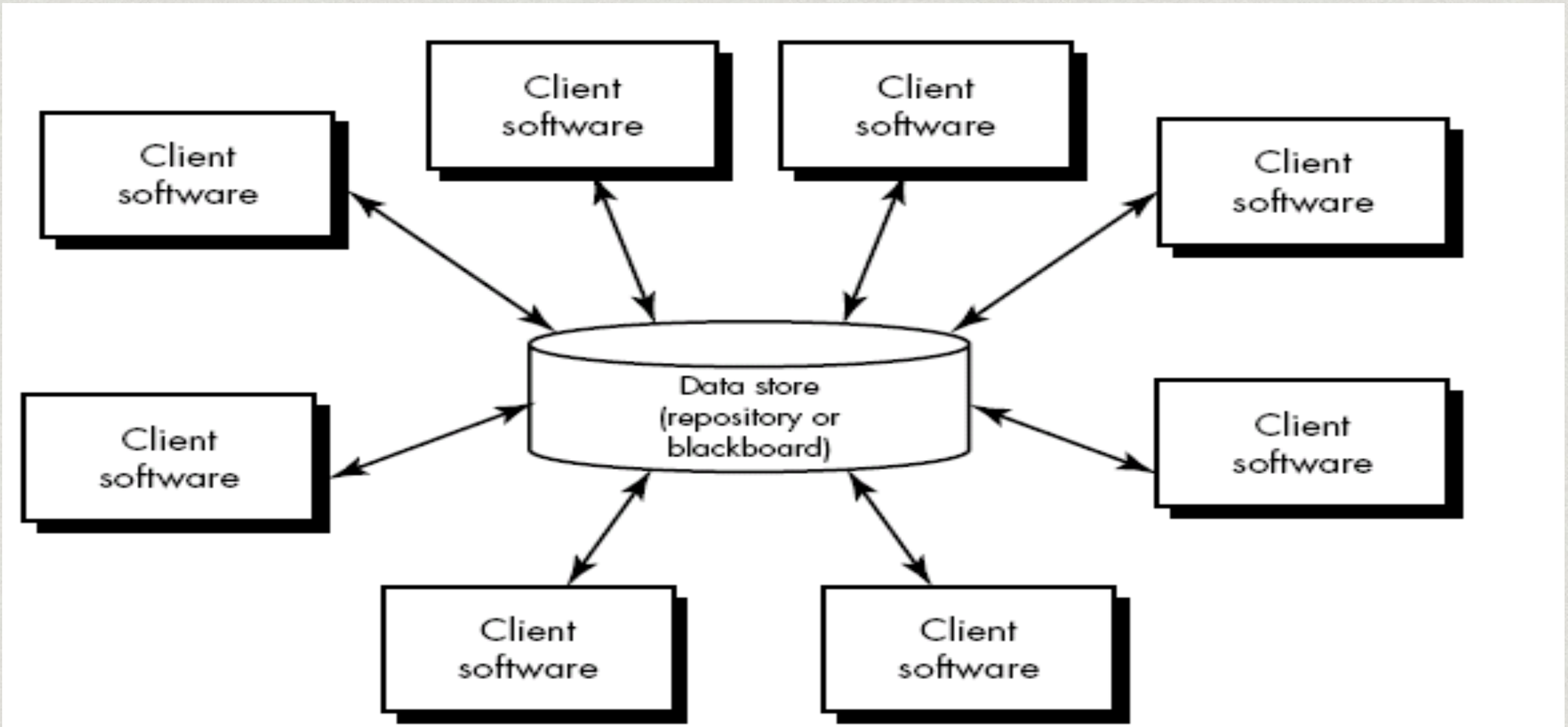
Style describes a system category that encompasses

- A set of *components* (e.g., a database, computational modules) that perform a function required by a system;
- a set of *connectors* that enable "communication, co-ordinations and cooperation" among components;
- *constraints* that define how components can be integrated to form the system
- *semantic models* that enable a designer to understand the overall properties of a system

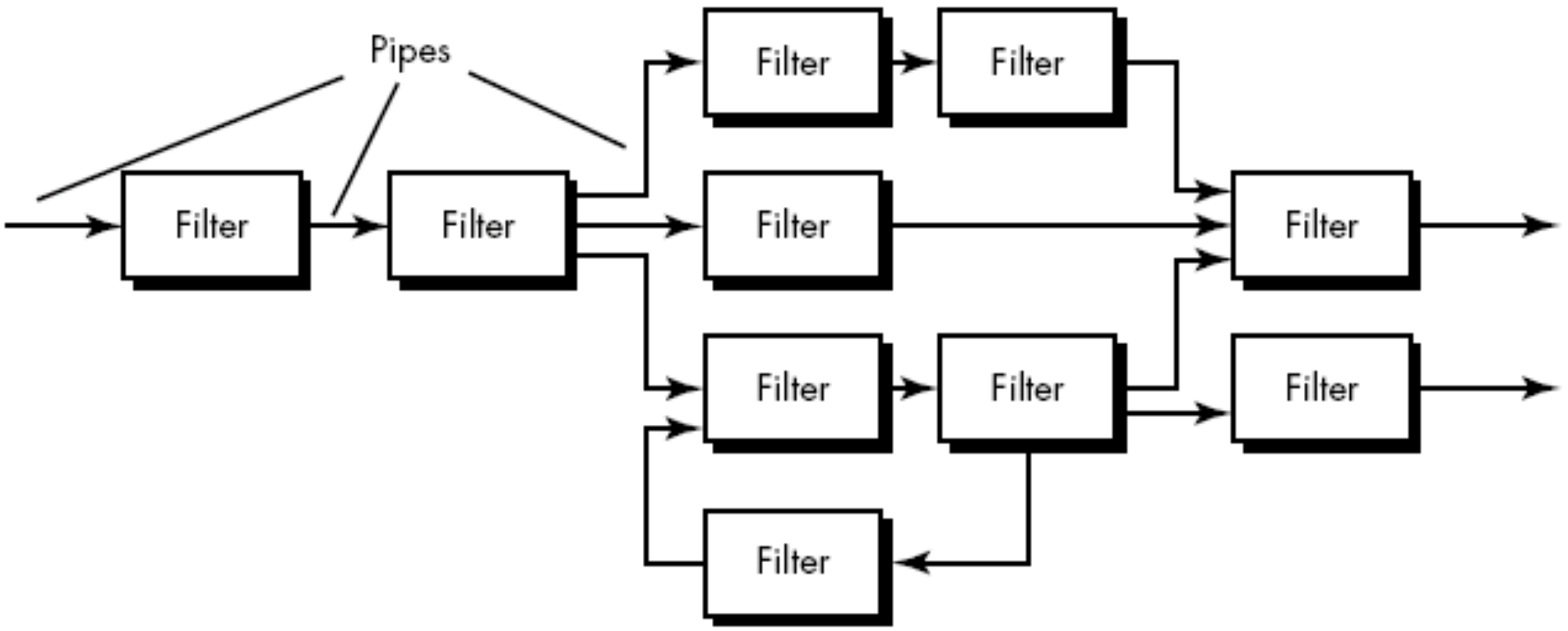
Pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context

Architectural Styles

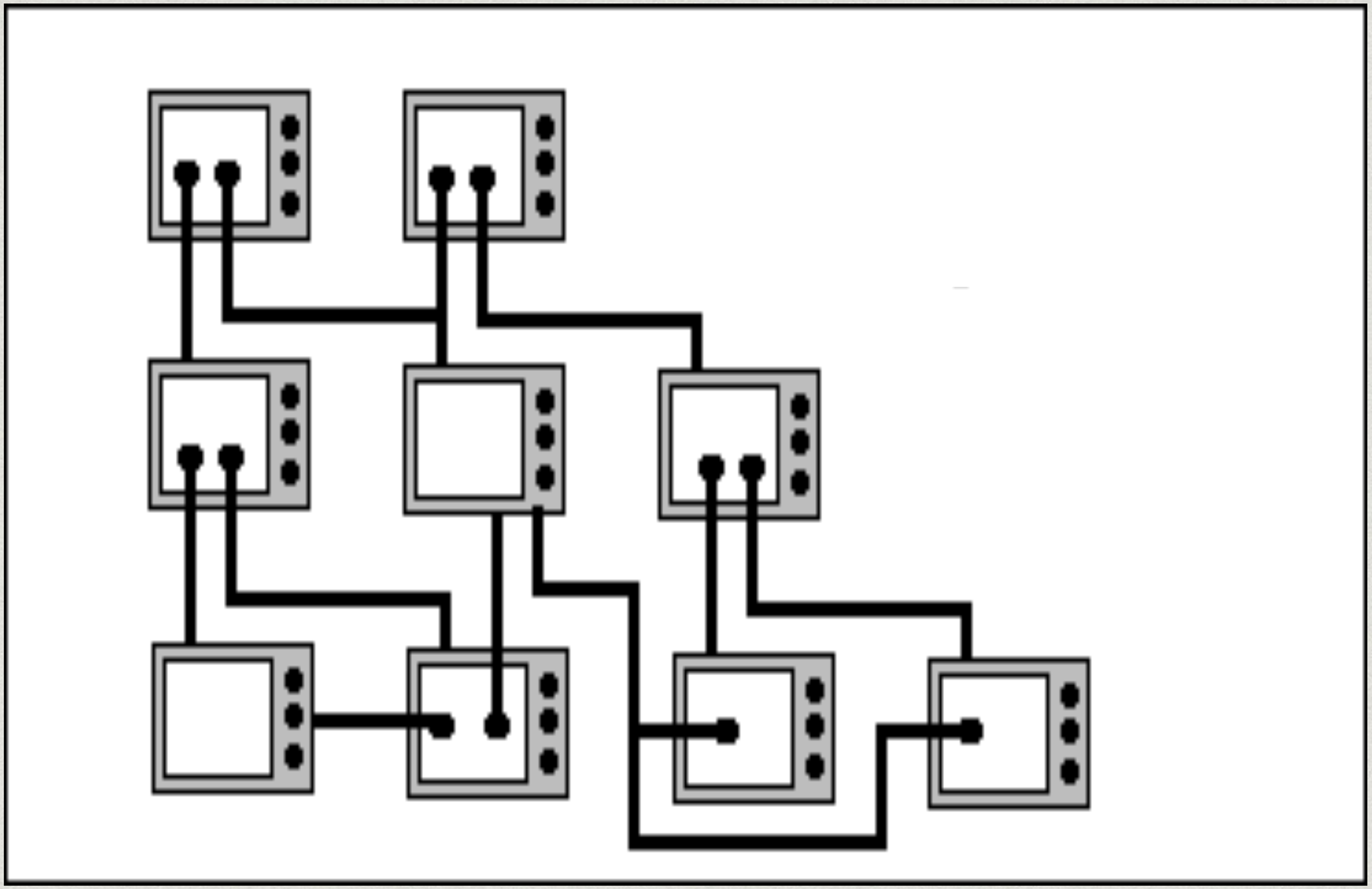
Data Centered Architecture



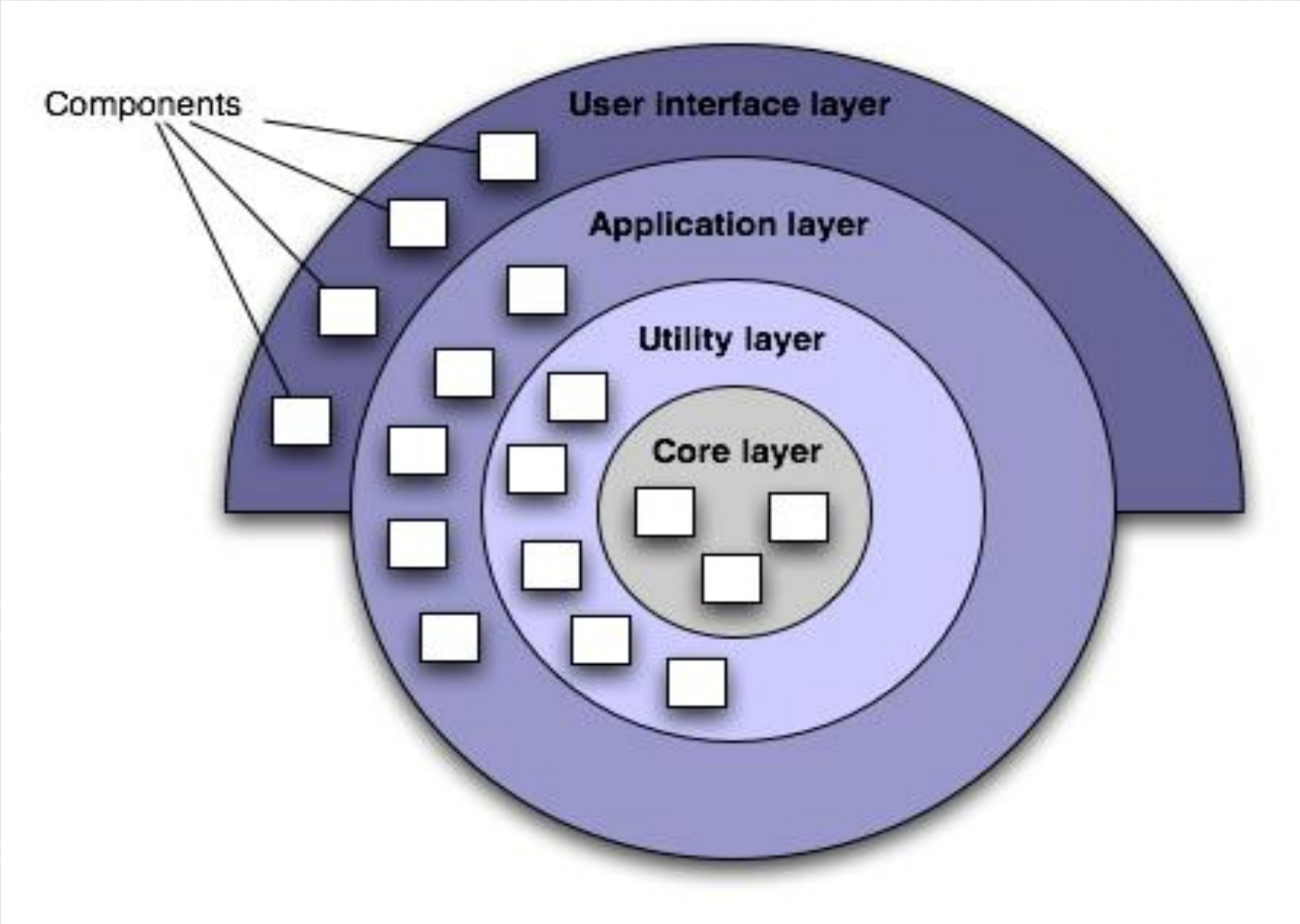
Pipe-Filter Architecture



Object Oriented Architecture



Layered Architecture



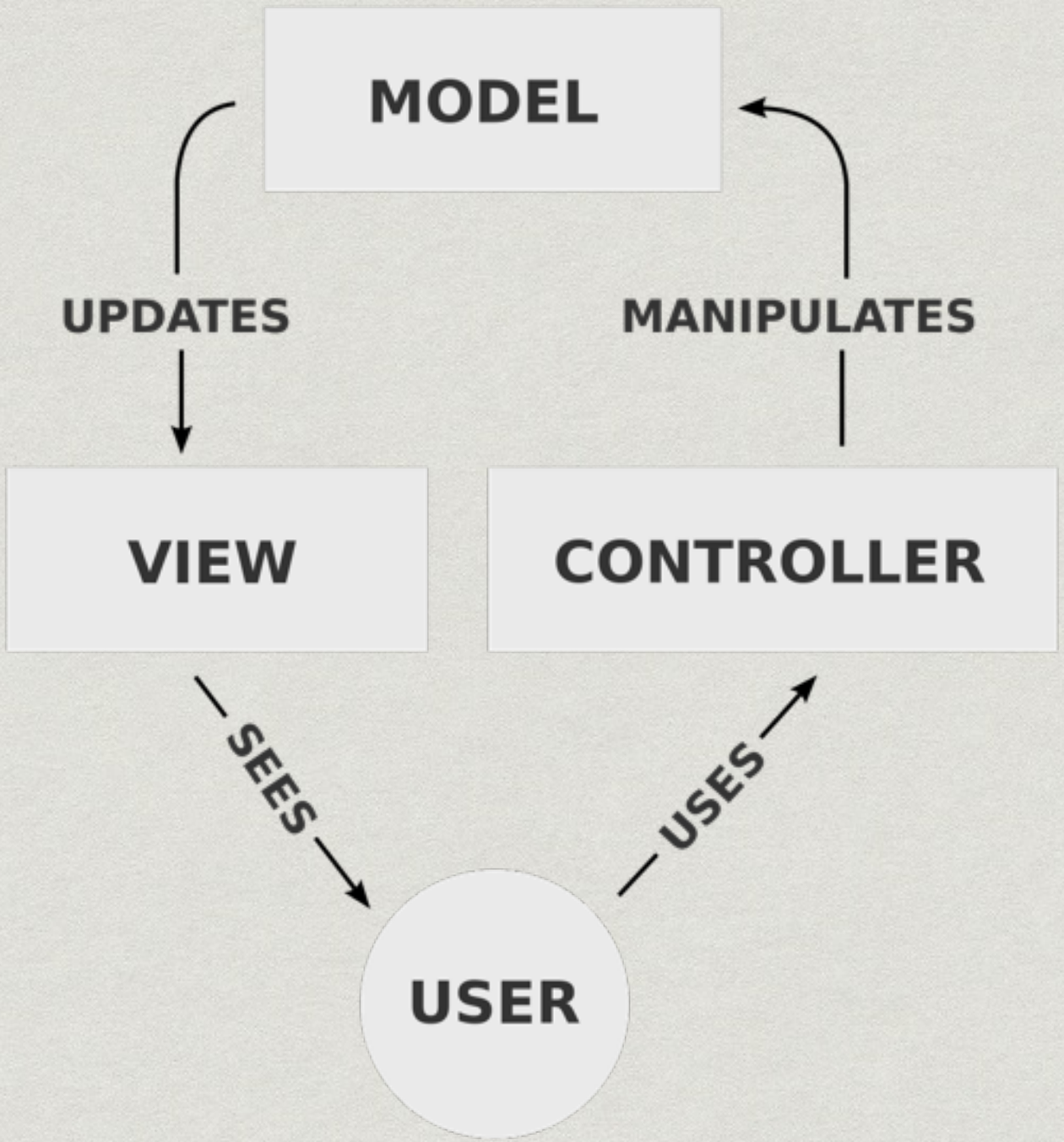
Architecture Design and the Agile Process

- Responsibility Driven Architecture. A process by which an architect is assigned to work with the team and product owner to decide when to make architectural decisions or changes

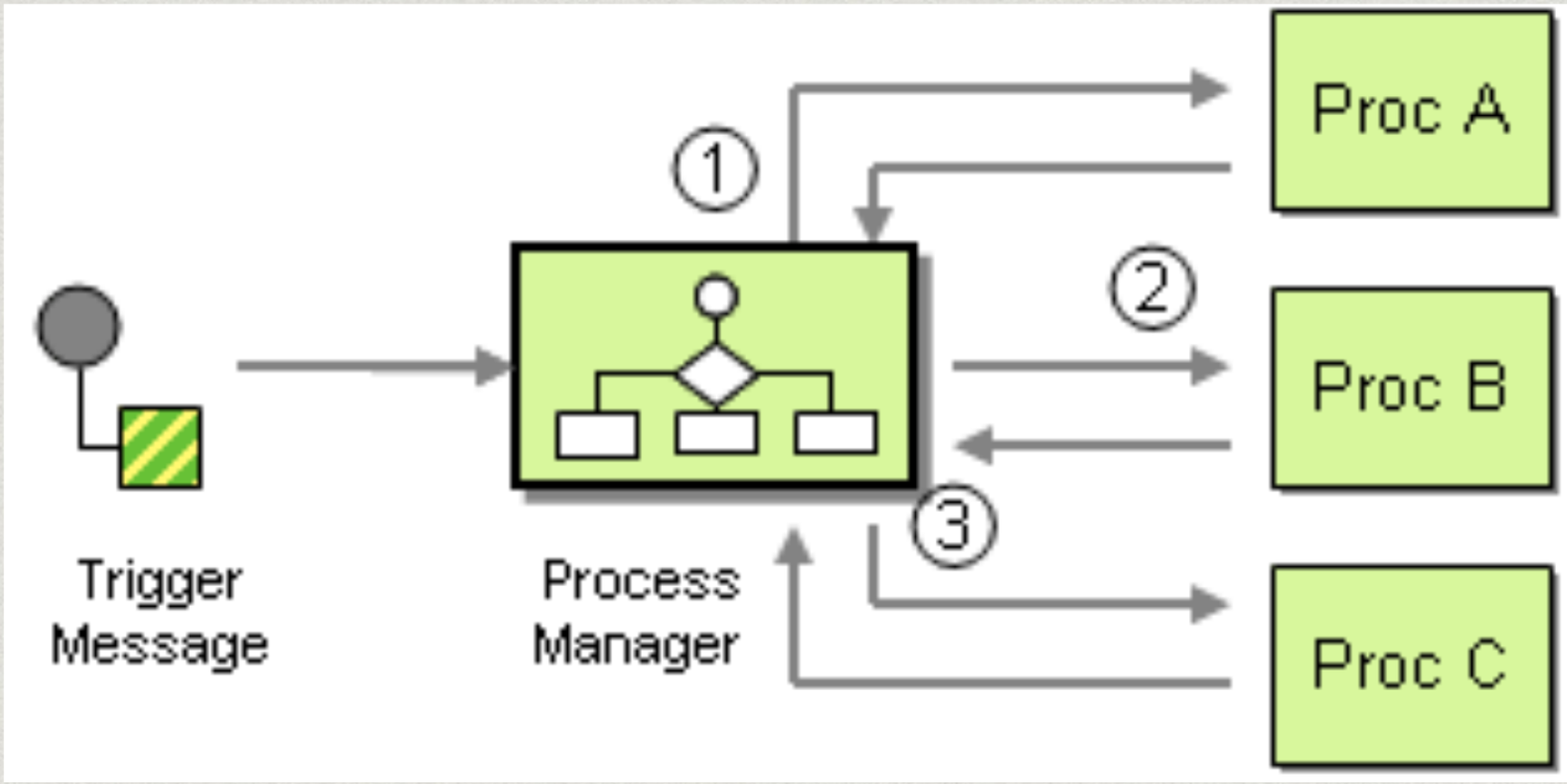
Resource: Stuart Blair, Richard Watt, Tim Cull, "Responsibility-Driven Architecture", IEEE Software, vol.27, no. 2, pp. 26-32, March/April 2010,

Architectural Patterns

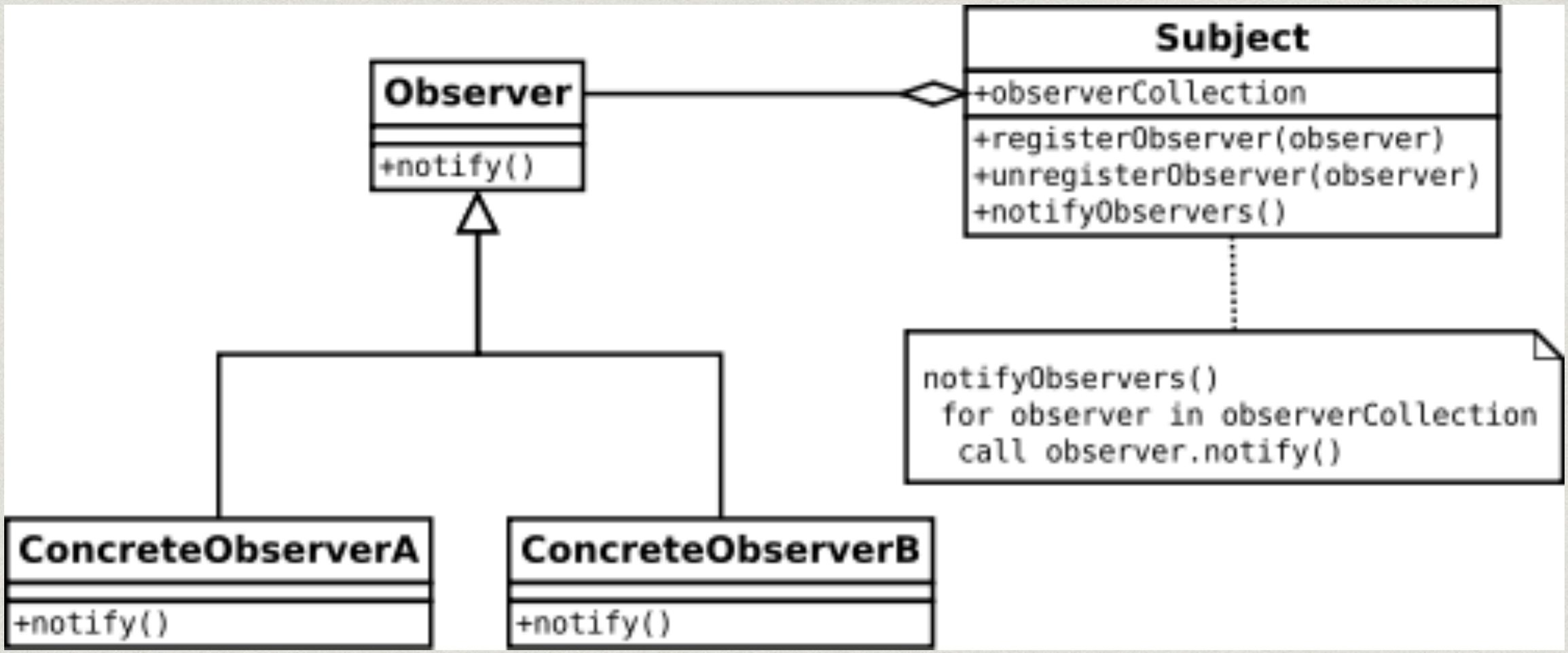
Model-View-Controller



Process Manager



Observer Pattern



More …

- **Layers, Pipes & Filters, Broker – POSA book series**
  Resource: Buschmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M., Pattern Oriented Software Architecture – a System of Patterns. Wiley, 1996.

- **Patterns of Enterprise Application Architecture such as Model-View-Controller, Domain Model, Active Record**
  Resource: Fowler M., Patterns of Enterprise Application Architecture. Addison Wesley, 2003

- **Domain-Driven Design patterns such as Bounded Context**
  Resource: Evans E., Domain-Driven Design. Tackling Complexity in the Heart of Software. Addison Wesley, 2003.

- **Enterprise Integration Patterns such as Messaging, Channel**
  Resource: Hohpe G., Woolf, B., Enterprise Integration Patterns. Addison Wesley, 2004.

- **Adapter, Observer and other design patterns**
  Resource: Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

Materials

- “Handbook of Software Architecture” collects such architectural patterns: http://handbookofsoftwarearchitecture.com/?cat=14

	Database	Application	Implementation	Infrastructure
Data/Content				
Problem statement ...	PatternName(s)		PatternName(s)	
Problem statement ...		PatternName(s)		PatternName(s)
Problem statement ...	PatternName(s)			PatternName(s)
Architecture				
Problem statement ...		PatternName(s)		
Problem statement ...		PatternName(s)		PatternName(s)
Problem statement ...				
Component-level				
Problem statement ...		PatternName(s)	PatternName(s)	
Problem statement ...				PatternName(s)
Problem statement ...		PatternName(s)	PatternName(s)	
User interface				
Problem statement ...		PatternName(s)	PatternName(s)	
Problem statement ...		PatternName(s)	PatternName(s)	
Problem statement ...		PatternName(s)	PatternName(s)	

Component Design Guidelines

- Open Closed Principle (OCP) “A module should be open for extension but closed for modification” - Example handling sensors with an abstraction Detector
- Liskov Substitution Principle (LSP) “Subclasses should be substitutable for their base classes”
- Dependency inversion Principle (DIP) “Depend on abstractions. Do not depend on concretions” abstractions are what you use to extend
- Common Closure Principle (CCP) “Classes that change together belong together”
- The Common Reuse Principle (CRP) “Classes that aren’t reused together should not be grouped together”

Interface Design Guidelines

- Place user in control
- Reduce user's memory and cognitive load
- Make interface consistent and intuitive

Design Patterns used for component and UI Design

INFO

Design Pattern Repositories

There are many sources for design patterns available on the Web. Some patterns can be obtained from individually published pattern languages, while others are available as part of a patterns portal or patterns repository. The following Web sources are worth a look:

Hillside.net **http://hillside.net/patterns/** One of the Web's most comprehensive collections of patterns and pattern languages

Portland Pattern Repository
http://c2.com/ppr/index.html Contains pointers to a wide variety of patterns resources and collections

Pattern Index
http://c2.com/cgi/wiki?PatternIndex
An "eclectic collection of patterns"

Handbook of Software Architecture
http://researcher.watson.ibm.com/researcher/view_project.php?id=3206/
Bibliographic reference to hundreds of architectural and component design patterns

UI Patterns Collections

UI/HCI Patterns
http://www.hcipatterns.org/patterns/borchers/patternIndex.html

Jennifer Tidwell's UI patterns
http://designinginterfaces.com/patterns/

Mobile UI Design Patterns
http://profs.info.uaic.ro/~evalica/patterns/

Pattern Language for UI Design
www.maplefish.com/todd/papers/Experiences.html

Interaction Design Library for Games
http://www.eelke.com/files/pubs/usabilitypatternsingames.pdf

UI Design Patterns
www.cs.helsinki.fi/u/salaakso/patterns/

Specialized Design Patterns:

Aircraft Avionics
http://g.oswego.edu/dl/acs/acs/acs.html

Business Information Systems
www.objectarchitects.de/arcus/cookbook/

Distributed Processing
www.cs.wustl.edu/~schmidt/

IBM Patterns for e-Business
www-128.ibm.com/developerworks/patterns/

Yahoo! Design Pattern Library
http://developer.yahoo.com/ypatterns/

WebPatterns.org
http://www.welie.com/index.php