



DESIGN PATTERNS

Course 1

ORGANIZATION

- Course: Each week 2 hours, Room 032

- Laboratory: Even weeks, Room 032

- Grade

- Written exam

- Laboratory activity

- Project

- $\text{Mark} = 0.5 \text{ written exam} + 0.2 \text{ laboratory activity} + 0.3 \text{ project}$

ORGANIZATION

□ Courses & Laboratories available at

□ web.info.uvt.ro/~zflavia

□ Contact

□ E-mail: zflavia@info.uvt.ro

□ 050B

COURSE CONTENT

- ☐ Design patterns

 - ☐ Creational

 - ☐ Structural

 - ☐ Behavioral

- ☐ Refactoring

- ☐ Anti-patterns

- ☐ Other design patterns

 - ☐ J2EE patterns

SOURCE CODE QUALITY

What characteristics should be respected in order to deliver a quality source code for a software project?

SOURCE CODE QUALITY

What characteristics should be respected in order to deliver a quality source code for a software project?

- Easy to read/understood – clear
- Easy to modify – structured
- Easy to reuse
- Simple (complexity)
- Easy to test
- Implements patterns for standard problems

SOURCE CODE QUALITY

What influence source code quality?

- ☐ Development time
- ☐ Costs
- ☐ Programmer experience
- ☐ Programmer abilities
- ☐ Specifications clarity
- ☐ Solution complexity
- ☐ Requirements change rate, team, ...

PATTERNS

- ❑ A *pattern* is a recurring **solution** to a standard **problem**, in a context.
- ❑ A *Design Pattern* systematically names, explains, and evaluates an important and recurring design.
- ❑ Christopher Alexander, a professor of architecture...
 - *Why would what a prof of architecture says be relevant to software?*
 - “A pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”

PATTERNS

❑ Patterns solve software structural problems

- ❑ Abstraction,
- ❑ Encapsulation
- ❑ Information hiding
- ❑ Separation of concerns
- ❑ Coupling and cohesion
- ❑ Separation of interface and implementation
- ❑ Single point of reference
- ❑ Divide and conquer

PATTERNS

❑ Patterns also solve non-functional problems

- ❑ Changeability
- ❑ Interoperability
- ❑ Efficiency
- ❑ Reliability
- ❑ Testability
- ❑ Reusability

PATTERNS

☐ Advantages

- ☐ Allow the standard solution reusability at source code/architectural level
- ☐ Allow the source code/architecture documentation
- ☐ Facilitate architecture and code understanding
- ☐ Known solutions – common vocabulary
- ☐ Well documented solutions

PATTERNS TYPES

- ❑ Architectural Patterns: MVC, Layers etc.
- ❑ Design Patterns: Singleton, Observer etc
- ❑ GUI Design Patterns: Window per task, Disabled irrelevant things, Explorable interface etc
- ❑ Database Patterns: decoupling patterns, resource patterns, cache patterns etc.
- ❑ Concurrency Patterns: Double buffering, Lock object, Producer-consumer, Asynchronous processing etc.
- ❑ Enterprise (J2EE) Patterns: Data Access Object, Transfer Objects etc.
- ❑ GRASP (General Responsibility Assignment Patterns): Low coupling/high cohesion, Controller, Law of Demeter (don't talk to strangers), Expert, Creator etc.
- ❑ Anti-patterns = bad solutions largely observed: God class, Singletonitis, Basebean etc

DESIGN PATTERNS TYPES

3 types of patterns ...

Creational

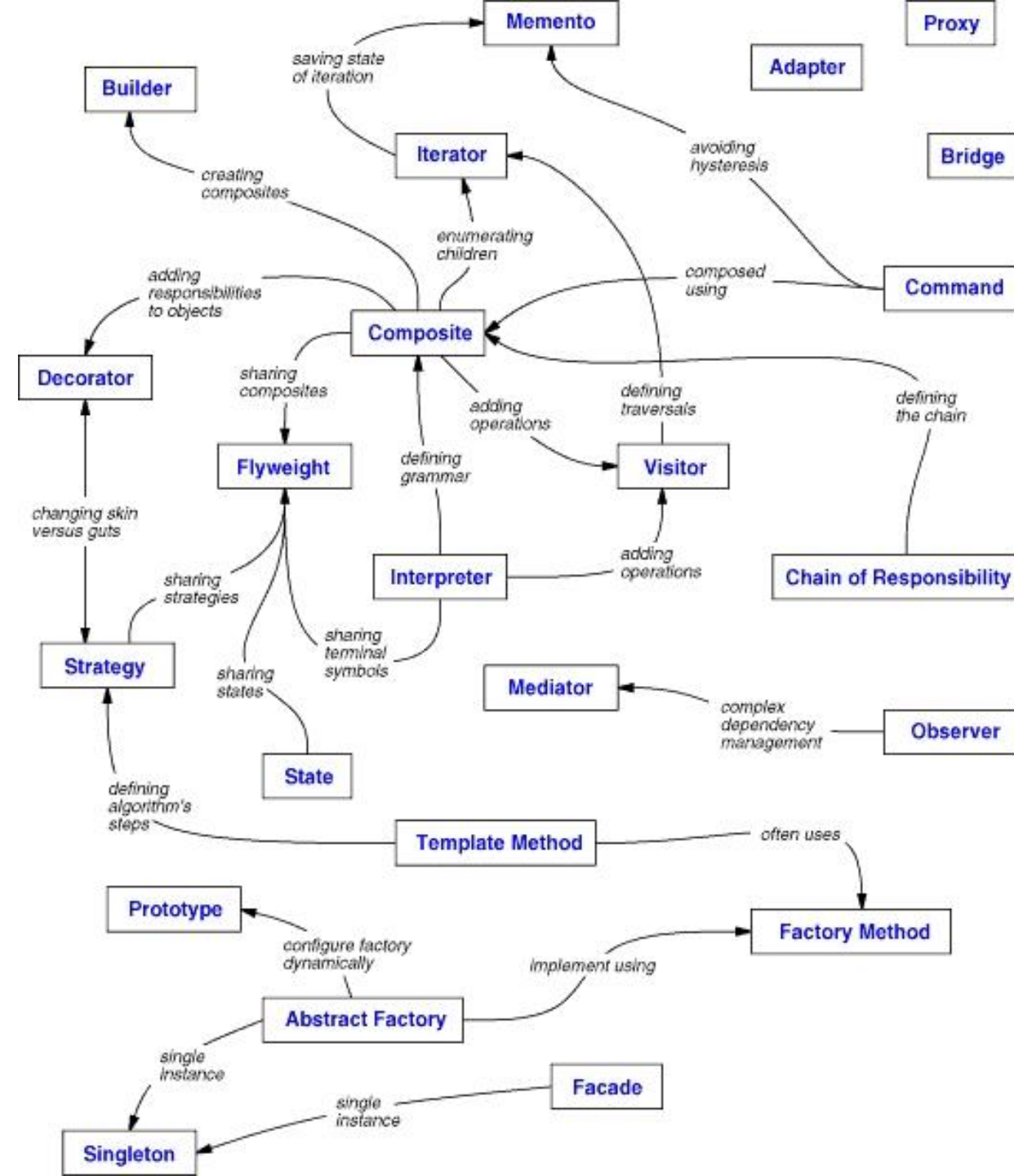
- address problems of creating an object in a flexible way. Separate creation, from operation/use.

Structural

- address problems of using O-O constructs like inheritance to organize classes and objects

Behavioral

- address problems of assigning responsibilities to classes. Suggest both static relationships and patterns of communication (use cases)



DESIGN PATTERNS CREATIONAL

Creational patterns

- Abstract the instantiation process
- Make a system independent to its realization
- Class Creational use inheritance to vary the instantiated classes
- Object Creational delegate instantiation to an another object

DESIGN PATTERNS STRUCTURAL

Structural patterns

- Class Structural patterns concern the aggregation of classes to form largest structures
- Object Structural pattern concern the aggregation of objects to form largest structures

DESIGN PATTERNS BEHAVIORAL

Behavioral patterns

- Concern with algorithms and assignment of responsibilities between objects
- Describe the patterns of communication between classes or objects
- Behavioral class pattern use inheritance to distribute behavior between classes
- Behavioral object pattern use object composition to distribute behavior between classes

DESIGN PATTERNS TYPES

Creational	Structural	Behavioral
Factory Method	Adapter	Interpreter
Abstract Factory	Bridge	Template Method
Builder	Composite	Chain of Responsibility
Prototype	Decorator	Command
Singleton	Façade	Iterator
	Flyweight	Mediator
	Proxy	Memento
		Observer
		State
		Strategy
		Visitor

DESIGN PATTERNS - EXAMPLES

- ❑ Observer in Java AWT and Swing for callbacks
- ❑ Iterator in C++ STL and Java Collection
- ❑ Façade in many Open-Source library to hide the complexity of the internal runtime
- ❑ Bridge and proxy in frameworks for distributed applications
- ❑ Singleton in Hibernate and NHybernate

PATTERNS TEMPLATES

- ❑ Design patterns are described by 4 characteristics

- ❑ Pattern name

- ❑ Meaningful text that reflects the problem e.g. Brige, Mediator

- ❑ Problem

- ❑ intent of the pattern, context, when to apply

- ❑ Solution

- ❑ UML-like structure, abstract code
 - ❑ Static and dynamic relationships among the components

- ❑ Consequences

- ❑ Results and tradeoff

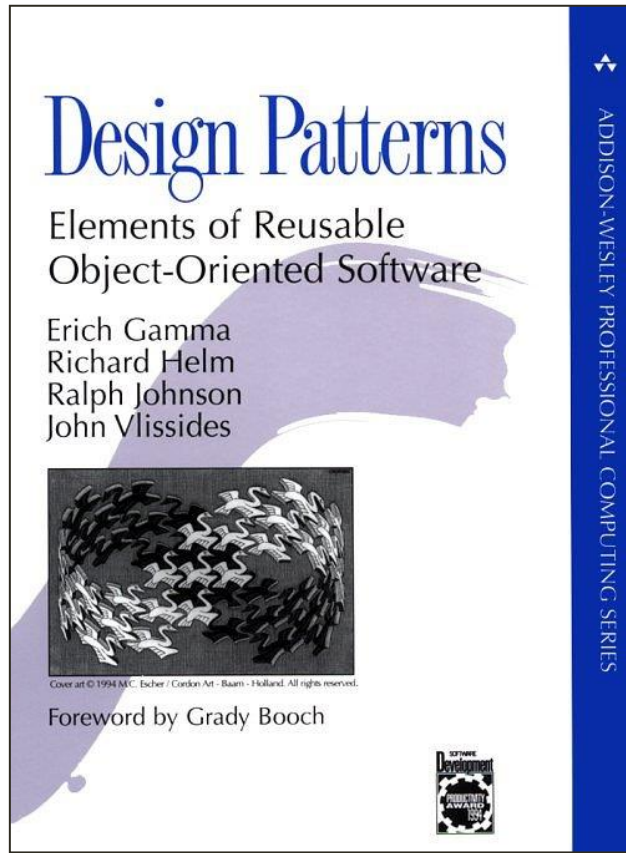
PATTERNS, ARCHITECTURE AND FRAMEWORK

Architectures model software structure at the highest possible level, and give the overall system view. An architecture can use many different patterns in different components

Patterns are more like small-scale or local architectures for architectural components or sub-components

Frameworks are partially completed software systems that may be targeted at a particular type of application. These are tailored by completing the unfinished components.

BIBLIOGRAPY



<http://www.oodesign.com/>