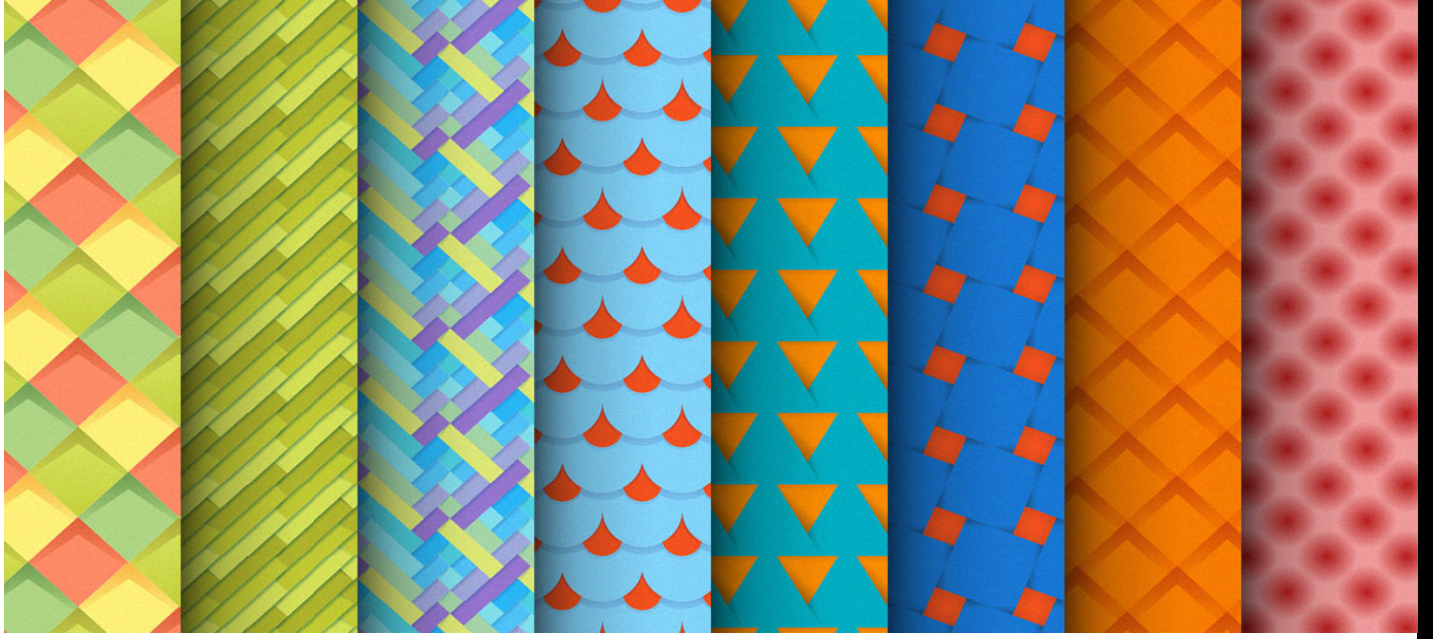


DESIGN PATTERNS

COURSE 1

OGANIZATION

- **Course**
  - Each week 2 hours, Room 050A
- **Laboratory**
  - Each odd/even school week, Room 050A
- **Presence**
  - Course: minimum 50%
  - Laboratory: minimum 50%
- **Grade**
  - Written exam 50%
  - Course activity 1%+ laboratory activity 24%
  - Presentation of a pattern 10%
  - Project 15%

ORGANIZATION

- **Course & laboratories**
  - available at http://staff.fmi.uvt.ro/~flavia.micota/
- **Contact**
  - e-mail: flavia.micota@e-uvt.ro
  - cab. 046B
- **Classroom**
  - 1rcgcs

COURSE CONTENT

- **Design patterns**
  - Creational
  - Structural
  - Behavioral
- **Refactoring**
- **Anti-patterns**
- **Students presentations of a pattern**

WAY YOU CHOSE THIS COURSE?

WAY YOU CHOSE THIS COURSE?

- **Some reasons from http://www.ida.liu.se/~chrke55/courses/SWE/bunus/DP01_1slide.pdf**
  - I could get some easy points.
  - Everybody is talking about so it must to be cool.
  - If I master this I can added it to my CV.
  - Increase my salary at the company.
  - Applying patterns is easier than thinking
  - A great place to pick up ideas to plagiarize.

DESIGN CHALLENGES

- Designing software with good modularity is hard!
- Designs often emerge from a lot of trial and error
- *Are there solutions to common recurring problems?*
  - A Design Pattern is *a Tried and True Solution To a Common Problem*
  - Basically, people, who have done this a lot, are making a suggestion!

SOURCE CODE QUALITY

- What characteristics should be respected in order to deliver a quality sorce code for a project?

SOURCE CODE QUALITY

- **What characteristics should be respected in order to deliver a quality sorce code for a 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
    - The capability of the software product to enable a specified modification to be implemented.
  - Interoperability
    - The ability of two or more systems or components to exchange information and to use the information that has been exchanged.
  - Efficiency
    - Efficiency signifies a level of performance that describes using the least amount of input to achieve the highest amount of output. Efficiency requires reducing the number of unnecessary resources used to produce a given output including personal time and energy.
  - Reliability
    - Probability of failure-free operation for a specified time in a specified environment for a given purpose
  - Testability
    - The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met
  - Reusability
    - The degree to which a software module or other work product can be used in more than one computing program or software system

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 solution

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

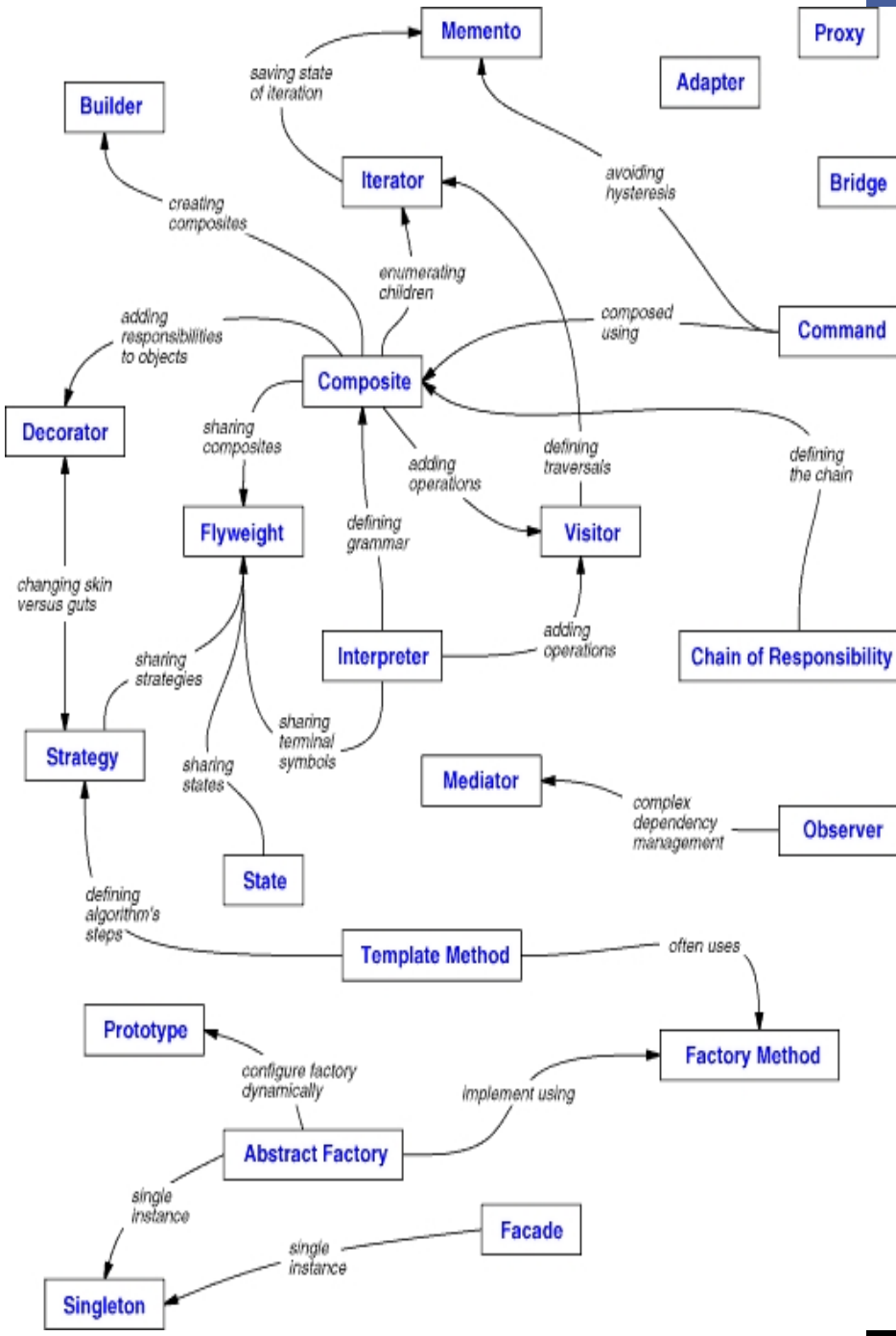
DESIGN PATTERNS HISTORY

- **1979: Christopher Alexander, architect, "The Timeless Way of Building",Oxford Press**
  - 253 patterns that collectively formed what the authors called a pattern language
- **1987: OOPSLA (Object Oriented Programming System), Orlando, presentation of design pattern to the community OO by Ward Cunningham and Kent Beck**
- **1995:Group of Four alias E. Gamma, R. Helm, R. Johnson and J. Vlissides : "Design Pattern: Elements of Reusable OO software"**
  - 23 design patterns in three categories

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 OO 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 PATERNS STRUCTURAL

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

- **Adapter Pattern**
- **Bridge Pattern**
- **Composite Pattern**
- **Decorator Pattern**
- **Facade Pattern**
- **Flyweight Pattern**
- **Proxy pattern**

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

- **Chain of Responsibility Pattern**
- **Command Pattern**
- **Interpreter Pattern**
- **Iterator Pattern**
- **Mediator Pattern**
- **Memento Pattern**
- **Observer Pattern**
- **State Pattern**
- **Strategy Pattern**
- **Template Pattern**
- **Visitor Pattern**
- **Null Object**

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

- **Factory Method Pattern**
- **Abstract Factory Pattern**
- **Singleton Pattern**
- **Prototype Pattern**
- **Builder Pattern**
- **Object Pool Pattern**

DESIGN PATTERNS - EXAMPLES

- **Observer** in Java AWT and Swing for components actions callbacks
- **Observer** in Java watches file for changes (Java 7 NIO)
- **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 main 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 TEMPLATES. COMPLETE

- **Intent**
  - short description of the pattern & its purpose
- **Also Known As**
  - any aliases this pattern is known by
- **Motivation**
  - motivating scenario demonstrating pattern's use
- **Applicability**
  - circumstances in which pattern applies
- **Structure**
  - graphical representation of the pattern using modified UML notation
- **Participants**
  - participating classes and/or objects & their responsibilities

PATTERNS TEMPLATES. COMPLETE

- **Collaborations**
  - how participants cooperate to carry out their responsibilities
- **Consequences**
  - the results of application, benefits, liabilities
- **Implementation**
  - pitfalls, hints, techniques, plus language-dependent issues
- **Sample Code**
  - sample implementations in C++, Java, C#, Smalltalk, C, etc.
- **Known Uses**
  - examples drawn from existing systems
- **Related Patterns**
  - discussion of other patterns that relate to this one

PATTERNS, ARHITECTURE 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.**

HOW TO USE DESIGN PATTERNS?

- **Part "Craft"**
  - Know the patterns
  - Know the problem they can solve
- **Part "Art"**
  - Recognize when a problem is solvable by a pattern
  - Part "Science"
- **Look up the pattern**
  - Correctly integrate it into your code

DESIGN PATTERNS PROVIDE A SHARED VOCABULARY

- Dev 1: “I made a Broadcast class. It keeps track of all of its listeners and anytime it has new data it sends a message to each listener. The listeners can join the Broadcast at any time or remove themselves from the Broadcast. It’s really dynamic and loosely-coupled!”

- Dev 2: “Why didn’t you just say you were using the Observer pattern?

BIBLIOGRAPY

http://www.oodesign.com/