Trusted Components
Reuse, Contracts and Patterns

Prof. Dr. Bertrand Meyer
Dr. Karine Arnout
Lecture 5: Design patterns
Agenda for today

- Overview
- Benefits of patterns
- The limits
“A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design.”

Erich Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, 1995, p 3.
A design pattern is a set of domain-independent architectural ideas — typically a design scheme describing some classes involved and the collaboration between their instances — captured from real-world systems that programmers can learn and apply to their software in response to a specific problem.

Karine Arnout, From Patterns to Components, 2004
A design pattern is given by one or more of
- A description of the pattern’s intent
- Use cases
- A software architecture for typical implementations
GoF’s description of a design pattern

- Pattern name and classification
- Intent
- Also known as
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Sample code
- Known uses
- Related patterns

Categorization by intent
The GoF design patterns

- **Creational**
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton

- **Structural**
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy

- **Behavioral**
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor
Creational design patterns (1/3)

- Creational
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton

- Behavioral
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

- Structural
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
Creational design patterns (2/3)

- **Goal:**
  - Put more flexibility into the instantiation process

- **How:**
  - Through inheritance or delegation

- **What:**
  - Defer parts of object creation
Creational design patterns (3/3)

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

Competitors

Complementary
Structural design patterns (1/3)

- **Creational**
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton

- **Structural**
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy

- **Behavioral**
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor
Goal:
- Compose software elements into bigger structures

How:
- Through inheritance (static binding) or composition (flexibility)
Structural design patterns (3/3)

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

Similar class diagrams

Rely on each other
Behavioral design patterns (1/3)

- **Creational**
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton

- **Structural**
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy

- **Behavioral**
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor
Behavioral design patterns (2/3)

- **Deal with:**
  - Algorithms
  - Assignment of responsibilities between objects
  - Communication between objects

- **How:**
  - Through inheritance or composition
Behavioral design patterns (3/3)

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Memento
  - Mediator
  - Observer
- State
- Strategy
- Template Method
- Visitor

Mediator implementation may use Observer pattern
Delegation vs. inheritance, but same intent
Example: A text editor

- The problem:
  - Enable users of an interactive system to cancel the effect of the last command ("Ctrl-Z")
  - Should support multi-level undo-redo
A text editor

- Notion of “current line”

- Example commands:
  - Insert line after current position
  - Insert line before current position
  - Delete current line
  - Replace current line
  - ...
Finding the right abstractions

- Here: The notion of **COMMAND**
deferred class

COMMAND

feature -- Command pattern

execute (args: TUPLE)
    -- Execute command with args.
    deferred
    end

undo (args: TUPLE)
    -- Undo last action.
    deferred
    end

redo (args: TUPLE)
    -- Redo last undone action.
    deferred
    end

end
Keeping the history of the session

\[ \text{history: LINKED\_LIST [COMMAND]} \]

- Insert
- Insert
- Remove
- Insert
- Remove

Oldest \quad \text{Most recent}
How to use the Command pattern

- Create a descendant of `COMMAND`
- and effect its features `execute`, `undo`, and `redo`

```
* COMMAND

+ INSERTION_COMMAND

+ DELETION_COMMAND

...```

Trusted Components: Reuse, Contracts and Patterns - Lecture 5
More design patterns

- Model View Controller (MVC)

- Specialized patterns (e.g. distributed systems, networking, Web services)

“Design patterns series”

- Publisher-Subscriber (≈Observer)
- Master-Slave (for parallel computation)
- Broker (for distributed systems)
- Layers (for architecture)
- ...

Chair of Software Engineering
Goal: Build robust and extendible GUIs
How: Decouple the view (GUI) from the model

(From: Java BluePrint Patterns - MVC)
Model View Controller (2/2)

**Model**
- Encapsulates application state
- Exposes application functionality
- Notifies view of changes

**View**
- Renders the model
- Sends user gestures to controller
- Allows controller to select view

**Controller**
- Defines application behavior
- Maps user actions to model updates
- Selects view for response
- One for each functionality

Change Notification → State change

User gestures → View selection
Agenda for today

- Overview
- Benefits of patterns
- The limits
Benefits of design patterns

- Capture the knowledge of experienced developers
- Publicly available “repository”
- Newcomers can learn them and apply them to their design
- Yield a better structure of the software (modularity, extendibility)
- Common pattern language
- Facilitate discussions between programmers and managers
Agenda for today

- Overview
- Benefits of patterns
- The limits
Solution to a particular recurring design issue in a particular context:

“Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to this problem in such a way that you can use this solution a million times over, without ever doing it the same way twice.”

Erich Gamma et al., Design Patterns, 1995
A step backward from reuse

- No available “pattern libraries”
- Programmers need to implement them each time anew
- A pedagogical tool, not a reuse tool

“One can hope that many of the “patterns” currently being studied will soon cease to be mere ideas, yielding instead directly usable library classes.”

Bertrand Meyer: OOSC2, 1997
1. Read the pattern once through for an overview.
2. Go back and study the Structure, Participants, and Collaborations sections.
3. Look at the Sample Code section to see a concrete example of the pattern in code.
4. Choose names for pattern participants that meaningful in the application context.
5. Define the classes.
6. Define application-specific names for operations in the pattern.
7. Implement the operations to carry out the responsibilities and collaborations in the pattern.
“A successful pattern cannot just be a book description: it must be a software component, or a set of components”.

Software reuse vs. design reuse

“Reuse of architectural and design experience is probably the single most valuable strategy in the basket of reuse ideas”

Clemens Szyperski, Component software, 1998

- Software reuse vs. design reuse:
  - Not much different with seamless development

- Combining both worlds:
  - From patterns to Eiffel components...
Cookbook after componentization

1. Look up the pattern componentizability classification.

2. If the pattern you seek belongs to the “componentizable” category:
   - Download the componentized version.
   - Write the descendant or client classes needed in your application.

else:
   - Download the Pattern Wizard.
   - Use it to generate skeleton classes for this pattern.
   - Fill in the skeleton classes according to your needs.
Complementary material

- From Patterns to Components:
  - Chapter 3: Design patterns

- Further reading:
    - Chapter 1: Introduction
End of lecture 5