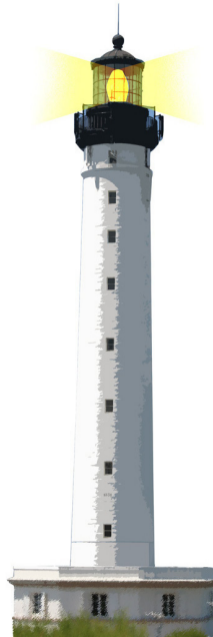


## An introduction to design patterns

S. Ducasse



<http://www.pharo.org>

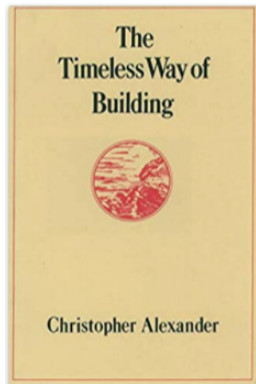


# Goal

- What are design patterns?
- Why they are good
- Patterns are not panacea
- Example



# Influenced by work on architecture



C. Alexander, *The Timeless Way of Building*, Oxford University Press, 1979

- tried to let people use patterns to build houses and other
- developed patterns and pattern languages
- failed in architecture but the idea is successful for object-oriented design

# Design Patterns

Design patterns are *recurrent* solutions to design *problems*

- Identification of similar problems at *design* level
- Experts solved *recurring problems* in similar ways
- There are pros and cons
- Design Patterns are literature: *Read them!*



# What are design patterns?

- Elegant solutions that a novice would not think of
- Generic
- Independent on specific type system or language
- Well-proven
  - Successfully tested in several systems
- Simple
- Can be combined for more complex solutions



# Watchout!

- There are really stupid patterns (e.g., supersuper) in some books
- Stay sceptical



# The most important point

Design Patterns are **names**

- They create a design **vocabulary**
  - Hook and Template, Factory, Composite, Visitor, Observer, Decorator...
- You can talk at lunch/blackboard about your design in an *abstract but precise* way



# What design patterns are not

- A design pattern is not *one* implementation
- A design pattern is *illustrated* using one *possible* implementation
- But alternate implementations may be better





# Elements of a pattern

- **Pattern name:** Increase design vocabulary
- **Pattern intent:** Describe the goal
- **Problem description:** When to apply it, in what context to use it
- **Solution description** (generic !):
  - The elements that make up the design
  - Their relationships, responsibilities, and collaborations
- **Consequences:** Results and trade-offs of applying the pattern



# Example: Strategy

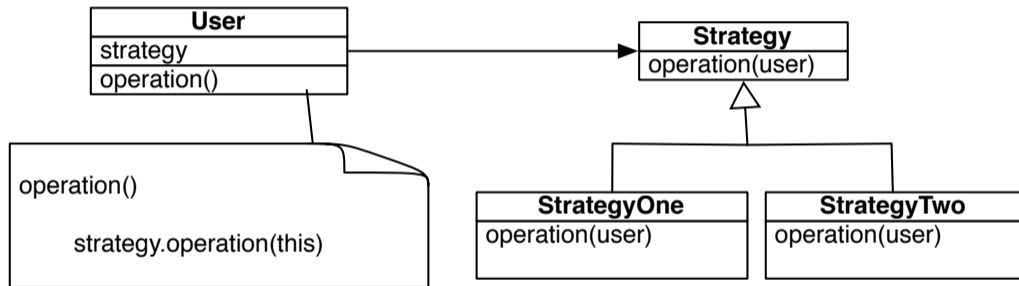
## Intent:

- Define a family of algorithms
- Encapsulate each in a separate class and
- Define each class with the *same* interface so that they can be *interchangeable*



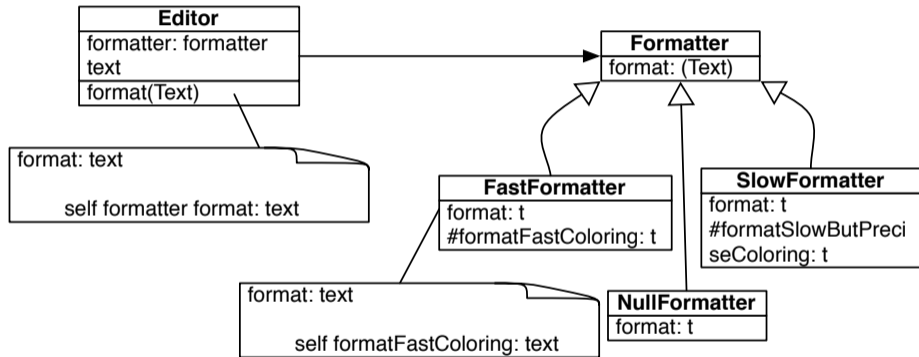
# Essence of Strategy

A possible implementation



- Variation: Passing or not the user to the strategy?

# Strategy application



# Some pattern categories

## **Creational Patterns**

- Instantiation and configuration of classes and objects

## **Structural Patterns**

- Usage of classes and objects in larger structures, separation of interfaces and implementation

## **Behavioral Patterns**

- Algorithms and division of responsibility



# Some creational patterns

- **Abstract factory** (how to create objects)
- **Builder** (provide a programmatic way to create objects)
- Factory Method
- Prototype
- **Singleton** (Watch out most people get it wrong)



# Some structural patterns

- Adapter
- Bridge
- **Composite** (support nested structure and common api)
- Decorator
- Façade (it is only useful in super limited case)
- **Flyweight** (how to avoid creating too many objects)
- Proxy



# Some behavioral patterns

- Chain of responsibility
- **Command** (reify operations)
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- **Strategy** (delegate behavior)
- **Template Method** (hook/template)
- **Visitor** (first class operation on composite)
- **Null Object**





# Alert!

Design Patterns may be a **plague!**

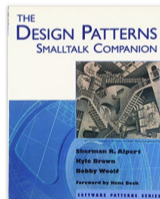
- Do not apply them when you do not need them
- Design Patterns make the software more complex
  - More classes
  - More indirections, more messages
  - Try to understand when NOT applying them!

# Alert!!

- Patterns are about **intent** and **tradeoffs**
- Do not confuse **intent** and **implementation**

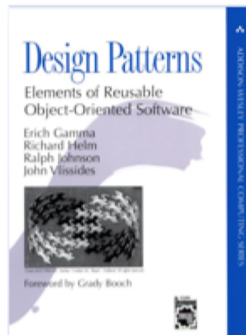
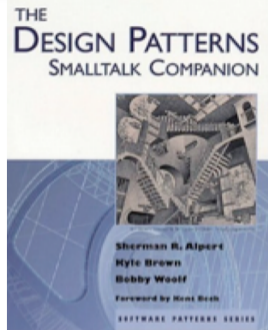


# Some books



- [Seminal one] **Design Patterns** (Gamma et al), also known as GoF book (Gang of Four)
- [Excellent] **Smalltalk Design Pattern Companion** (Alpert et al)

# What readers said



12 of 12 people found the following review helpful:

★★★★★ **Easier to understand than the original GoF**, February 4, 2000

Reviewer: [Nicolas Weidmann](#) (Zurich, Switzerland) - [See all my reviews](#)

This book gives you a better understanding of the patterns than in its original version (the GoF one). I am not a SmallTalk programmer but a 9 years C++ one. At work I had to use the GoF book and never liked reading it. In contrast to this, the SmallTalk companion is easy to read and you can understand the patterns within the first few lines of their description. Take the Bridge pattern and compare their discussions in the two books. If you really like the Gof one then buy it. But according to me, it would be a big mistake buying the GoF in favour of the SmallTalk companion. Trust a C++ programmer :-)

# Conclusion

## **Reusable solutions to common problems based on experiences from real systems**

- Names of abstractions creating a common vocabulary for developers
- Often support modularity (separation of interface/implementation)
- A basis for frameworks and toolkits basic elements to improve reuse



A course by

S. Ducasse, G. Polito, and Pablo Tesone



Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France  
<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>