Tecniche di Progettazione: Design Patterns

Laurea Magistrale in Informatica, Pisa

Alcune recensioni …

- Che fatica sti gradini!
- Io credo che il rapporto alzata/pedata sia uno dei più scomodi che io abbia mai provato! Con e senza tacchi. Mi sembra che il ponte sia frequentato prevalentemente da persone che "hanno fretta", essendo il collegamento principe fra la … e … Gente "in corsa", gente che "deve andare" e che non appena fa il primo gradino è costretta, non senza impaccio, a ridimensionare il passo... a rallentare il tutto, diventando una goffaggine unica con il trolley e le borse. E non mi vengano a dire che … ha pensato al tema "della sosta", dell'arrivo, della visione contemplativa della bella … dall'alto perchè non ci credo. … La linea del ponte è senz'altro di indubbia eleganza. Ma un ponte che non assolve pienamente alla sua funzione di collegamento è un ponte misero.

Alcune recensioni …(cont'd)

- i gradini del ponte hanno un rapporto alzata e pedata troppo variabile tra loro che rende abbastanza fastidioso il transito.
- Il ponte si è dimostrato pericoloso, dal momento che la sua struttura a gradini irregolari crea delle barriere percettive che ingannano anche l'occhio di chi ci vede bene e la gamba di chi ha mobilità buona, e quasi dieci persone sono finite all'ospedale

Pattern: il gradino

- L'alzata in genere compresa tra i 13 e i 20 cm
- La pedata viene calcolata usando:
  - Formula di Blondel: 2A + P = 62 ÷ 64 cm
  - Al variare della pendenza, l'utente tende a modificare la lunghezza del passo di modo tale che il lavoro svolto per superare un gradino sia uguale al lavoro svolto per compiere lo stesso passo su un piano.

P = 50, A = 7 → 50+2x7=64

Pattern: il ponte veneziano

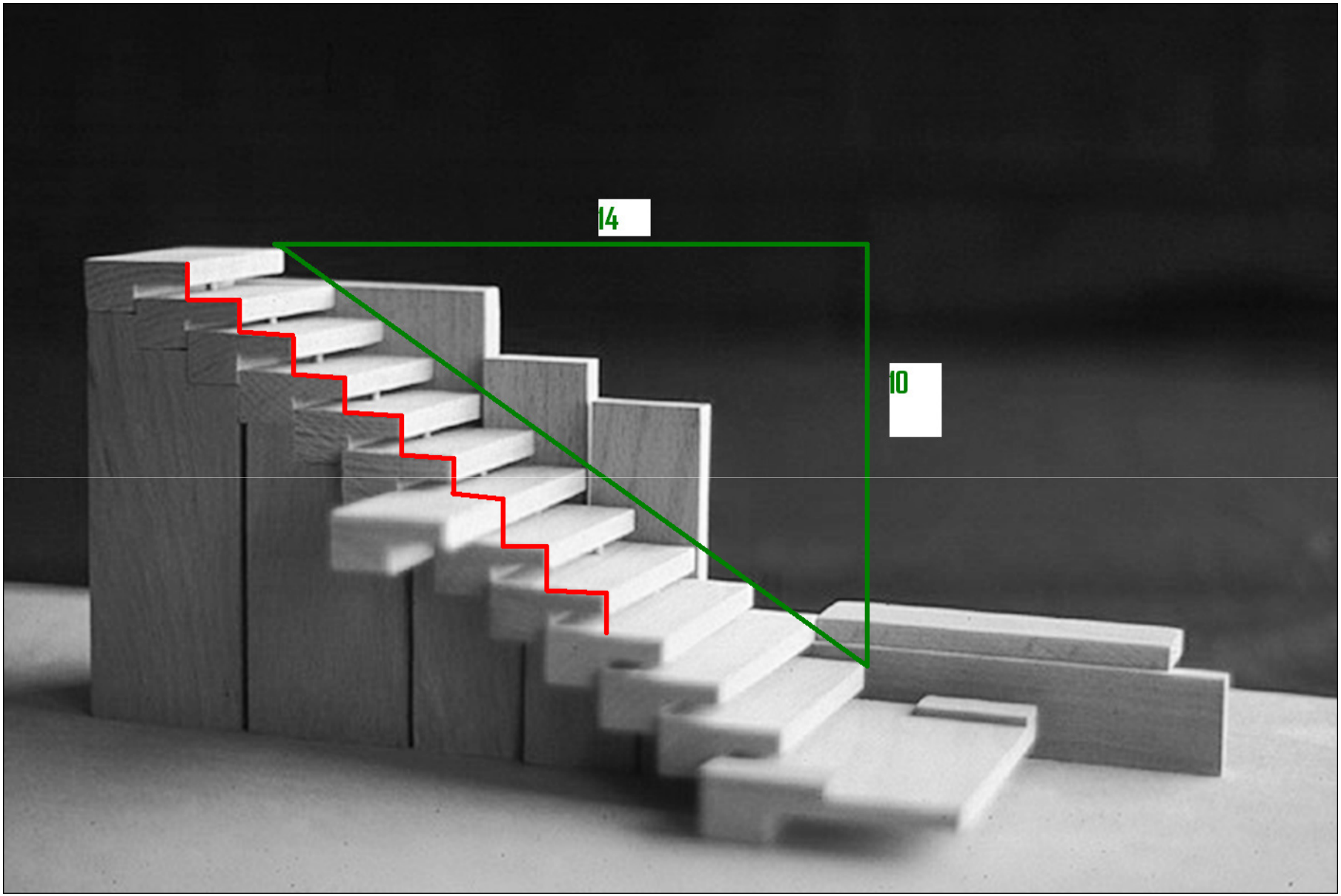
- Profilo a semiarco caratteristico dei ponti di Venezia per permettere il passaggio delle barche.
- Se Calatrava fosse più umile avrebbe chiesto ad un qualsiasi vero muratore (murer) di Venezia: perché tutti i ponti di Venezia hanno la "pedata variabile"? La risposta sarebbe stata: perché seguono una "legge fissa" imperniata sul "passo veneziano", antica unità di misura usata apposta per far si che si possa percorrere un ponte con la stessa cadenza, così che siano i gradini stessi a cercare di agevolarci.


HOTEL
PALAZZO PRIULI
PONTE DEL
DIAVOLO


Fiorenzo De Luca Photography


Ferrovia
Ferrovia

L'uso della pietra d'Istria alternata alla trachite scura per marcare il gradino costituiscono un'ottima soluzione individuata già dai veneziani del XV secolo.


14
10


olivetti


Dino Buz

I pattern come osservazione

- I pattern non sono "creati"
- Nascono come accumulo di conoscenze
- A un certo punto, nella soluzione a un problema, ci può essere un passo "creativo", che porta ad una "rivoluzione"
  - Es: ascensore.
  - Con l'esperienza di progettazione di ascensori poi si arriva alla definizione di pattern per gli ascensori
  - Etc…
- Scienza rivoluzionaria: Thomas Kuhn
- Patterns aren't created or invented: they are discovered (or "mined") from empirical observation.

La progettazione come processo non solo creativo

- Dante Benini: … come Scarpa ha detto a me

  "Leggere cento pagine di architettura al giorno …".

Christopher Alexander

Who's Who
Archives
Buildings
Paintings
Books
Film
C Vitae
Computing
Wiki
Software
Patterns

Christopher Alexander is Professor in the Graduate School and Emeritus Professor of Architecture at the University of California, Berkeley.

He is the father of the Pattern Language movement in computer science, and *A Pattern Language*, a seminal work that was perhaps the first complete book ever written in hypertext fashion.

He has designed and built

What is a Pattern

- Current use comes from the work of the architect Christopher Alexander
  - Alexander studied ways to improve the process of designing buildings and urban areas
  - "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution."
  - Hence, the common definition of a pattern: "A solution to a problem in a context."
  - Patterns can be applied to many different areas of human endeavour, including software development
- *A Pattern Language* that was written by C. Alexander and five colleagues at the Center for Environmental Structure in Berkeley, California in the late 1970s.

What is a Design Pattern?

- "Each 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"

  - -- Christopher Alexander A Pattern Language, 1977

Why Patterns?

- "Designing object-oriented software is hard and designing reusable object-oriented software is even harder."
  - - Erich Gamma
- Experienced designers reuse solutions that have worked in the past.
- Well-structured object-oriented systems have recurring patterns of classes and objects
- Knowledge of the patterns that have worked in the past allows a designer to be more productive and the resulting designs to be more flexible and reusable

Software Patterns History

- 1987 - Cunningham and Beck used Alexander's ideas to develop a small pattern language for Smalltalk
- 1990 - The Gang of Four (Gamma, Helm, Johnson & Vlissides) begin compiling a catalog of design patterns
- 1991 - First Patterns Workshop at OOPSLA
- 1993 - Kent Beck and Grady Booch sponsor the first meeting of what is now known as the Hillside Group
- 1994 – 1st Pattern Languages of Programs (PLoP) conf.
- 1995 - The Gang of Four (GoF) *Design Patterns book*

Benefits Of Design Patterns

- Capture expertise and make it accessible to non-experts in a standard form
- Facilitate communication among developers by providing a common language
- Make it easier to reuse successful designs and avoid alternatives that diminish reusability
- Facilitate design modifications
- Improve design documentation
- Improve design understandability

Wednesday, February 16, 2011 Login Text Size

THE HILLSIDE GROUP

Site Search

Search...

MAIN MENU

- Hilltop
  - News
  - History
  - Hillside Europe
  - Membership
  - Mission Statement
  - Vision
  - Board
- Conferences
- Patterns

Hilltop > History

Hillside History

In August of 1993, Kent Beck and Grady Booch sponsored a mountain retreat in Colorado where a group converged on foundations for software patterns. Ward Cunningham, Ralph Johnson, Ken Auer, Hal Hildebrand, Grady Booch, Kent Beck and Jim Coplien struggled with Alexander's ideas and our own experiences to forge a marriage of objects and patterns. The Group agreed that we were ready to build on Erich Gamma's foundation work studying object-oriented patterns, to use patterns in a generative way in the sense that Christopher Alexander uses patterns for urban planning and building architecture. We then used the term "generative" to mean "creational" to distinguish them from "Gamma patterns" that captured observations. The Group was meeting on the side of a hill when all this occurred, hence the name.

Since then, the **Hillside Group** has been incorporated as an educational non-profit. It has sponsored and helped run various conferences (PlopConference, EuroPlop, ChiliPlop, KoalaPlop, Mensore PLoP, SugarloafPLoP, and UP97) and has been responsible for getting the PatternLanguagesOfProgramDesign series of books put together and published.

More history can be found on the Portland's Pattern Repository

Types of Patterns

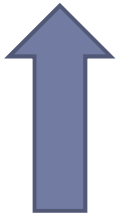
- Riehle and Zullighoven in "*Understanding and Using Patterns in Software Development*" *mention three types of software patterns*
- Conceptual Pattern
  - Pattern whose form is described by means of terms and concepts from the application domain
- Design Pattern
  - Pattern whose form is described by means of software design constructs, such as objects, classes, inheritance and aggregation
- Programming Pattern (Programming Idiom)
  - Pattern whose form is described by means of programming language constructs

Still others

- Organizational Patterns
  - Recurring structures of relationship, usually in a professional organization, that help the organization achieve its goals.
  - The patterns are usually inspired by analyzing multiple professional organizations and finding common structures in their social networks.
- Process Patterns
  - A process pattern is a pattern which describes a proven, successful approach and/or series of actions for developing software
  - E.g. software life cycle

Design Pattern Levels Of Abstraction

- Complex design for an entire application or subsystem
- Solution to a general design problem in a particular context
- Simple reusable design class such as a linked list, hash table, etc.



More Abstract

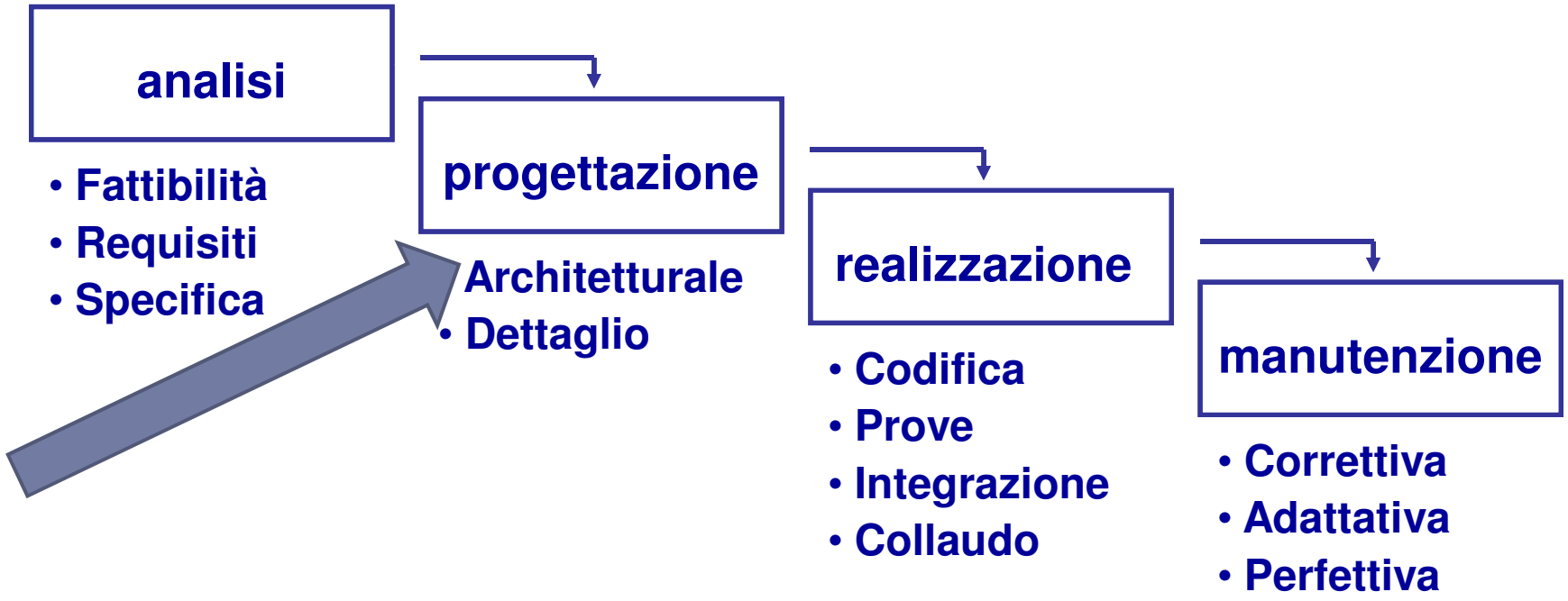
More Concrete

Architecture-Design-Code

- Architectural Design Patterns
- Design Patterns
- Idioms o Coding Design Patterns

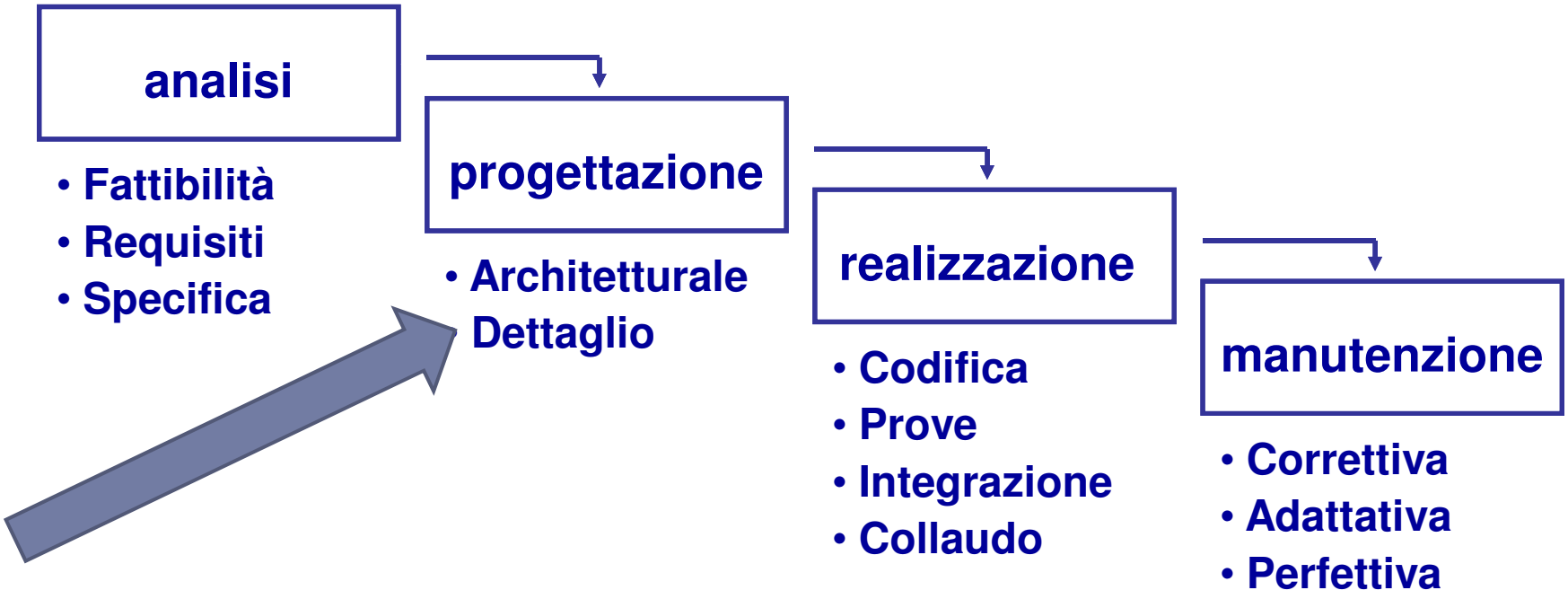
Architecture-Design-Code

- Architectural Design Patterns
  - Thay address the architecture of a sw system
  - E.g. Layers, Pipes and Filters, Blackboard, Broker, Model-View-Controller, …



Architecture-**Design**-Code

- Design Patterns
  - They address the design and refinement of components.
  - E.g. abstract factory, decorator, …



Architecture-Design-**Code**

- Idioms o Coding Design Patterns
  - An idiom is a low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language.
  - An Idiom is more restricted than a design pattern
    - Still describes a recurring problem
    - Provides a more specific solution, with fewer variations
    - Applies only to a narrow context
      - e.g., the C++ language
  - E.g. Naming conventions, Source code formats, Memory management, ... Counted Pointer (C++), Implementation of Singleton design pattern (C++, Smalltalk)

GoF Design Patterns

- The GoF design patterns are in the middle of these levels of abstraction
- "A design pattern names, abstracts, and identifies key aspects of a common design structure that makes it useful for creating a reusable object-oriented design."
- The GoF design patterns are "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context."

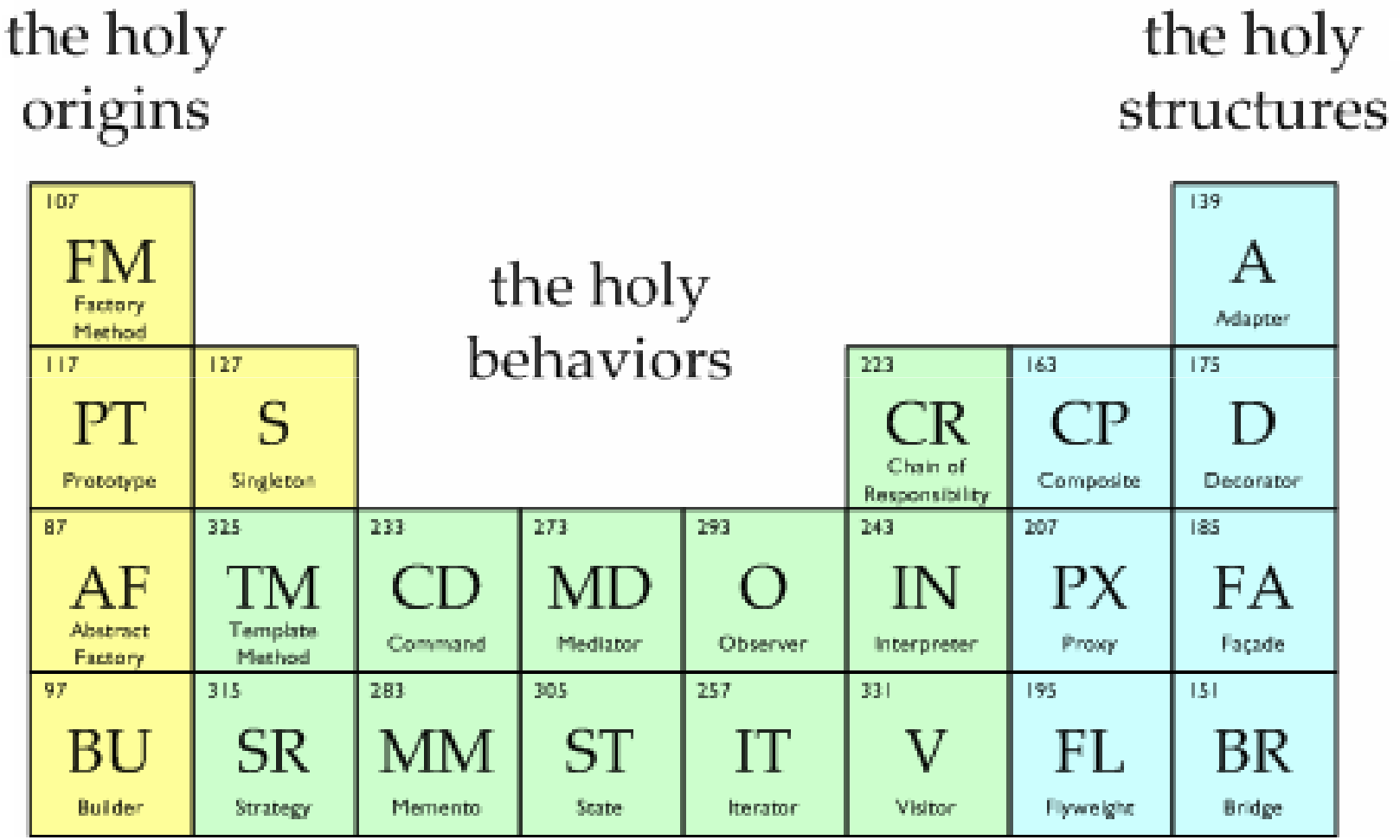
Best known families of patterns

- GRASP
  - General Responsibility Assignment Software Patterns (or Principles) [Larman]
  - Information Expert, Creator, Controller, Low Coupling, High Cohesion, Polymorphism, Pure Fabrication, Indirection, Protected Variations
- SOLID
  - Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion
- GoF
  - 23 design patterns
- POSA
  - A System of Patterns: Pattern-Oriented Software Architecture
  - Volumes 1—5

GoF Classification Of Design Patterns

- Purpose - what a pattern does
  - Creational Patterns
    - Concern the process of object creation
    - *Abstract Factory, Builder, Factory Method, Prototype, Singleton.*
  - Structural Patterns
    - Deal with the composition of classes and objects
    - *Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy.*
  - Behavioral Patterns
    - Deal with the interaction of classes and objects
    - *Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template, Visitor.*

The Sacred Elements of the Faith



GoF Classification Of Design Patterns (Continued)

- Scope - what the pattern applies to
  - Class Patterns
    - Focus on the relationships between classes and their subclasses
    - Involve inheritance reuse
  - Object Patterns
    - Focus on the relationships between objects
    - Involve composition reuse

GoF Essential Elements Of Design Patterns

- Pattern Name
  - Having a concise, meaningful name for a pattern improves communication among developers
- Problem
  - What is the problem and context where we would use this pattern?
  - What are the conditions that must be met before this pattern should be used?

GoF Essential Elements Of Design Patterns (Continued)

- Solution
  - A description of the elements that make up the design pattern
  - Emphasizes their relationships, responsibilities and collaborations
  - Not a concrete design or implementation; rather an abstract description
- Consequences
  - The pros and cons of using the pattern
  - Includes impacts on reusability, portability, extensibility

GoF Pattern Template

- Pattern Name and Classification
  - A good , concise name for the pattern and the pattern's type
- Intent
  - Short statement about what the pattern does
- Also Known As
  - Other names for the pattern
- Motivation
  - A scenario that illustrates where the pattern would be useful
- Applicability
  - Situations where the pattern can be used

GoF Pattern Template (Continued)

- Structure
  - A graphical representation of the pattern
- Participants
  - The classes and objects participating in the pattern
- Collaborations
  - How to do the participants interact to carry out their responsibilities?
- Consequences
  - What are the pros and cons of using the pattern?
- Implementation
  - Hints and techniques for implementing the pattern

GoF Pattern Template (Continued)

- Sample Code
  - Code fragments for a sample implementation
- Known Uses
  - Examples of the pattern in real systems
- Related Patterns
  - Other patterns that are closely related to the pattern

GoF Notation

- The GoF book uses the Object Modeling Technique (OMT) notation for class and object diagrams:
- OMT has proposed three main types of models:
  - *Object model* : represents the static phenomena in the modeled domain. Main concepts are classes and associations, with attributes and operations, aggregations and generalizations (with multiple inheritance).
  - *Dynamic model* : represents a state/transition view on the model. Main concepts are states, transitions, and events.
  - *Functional model* : handles the process perspective of the model, corresponding roughly to data flow diagrams. Main concepts are process, data store, data flow, and actors.

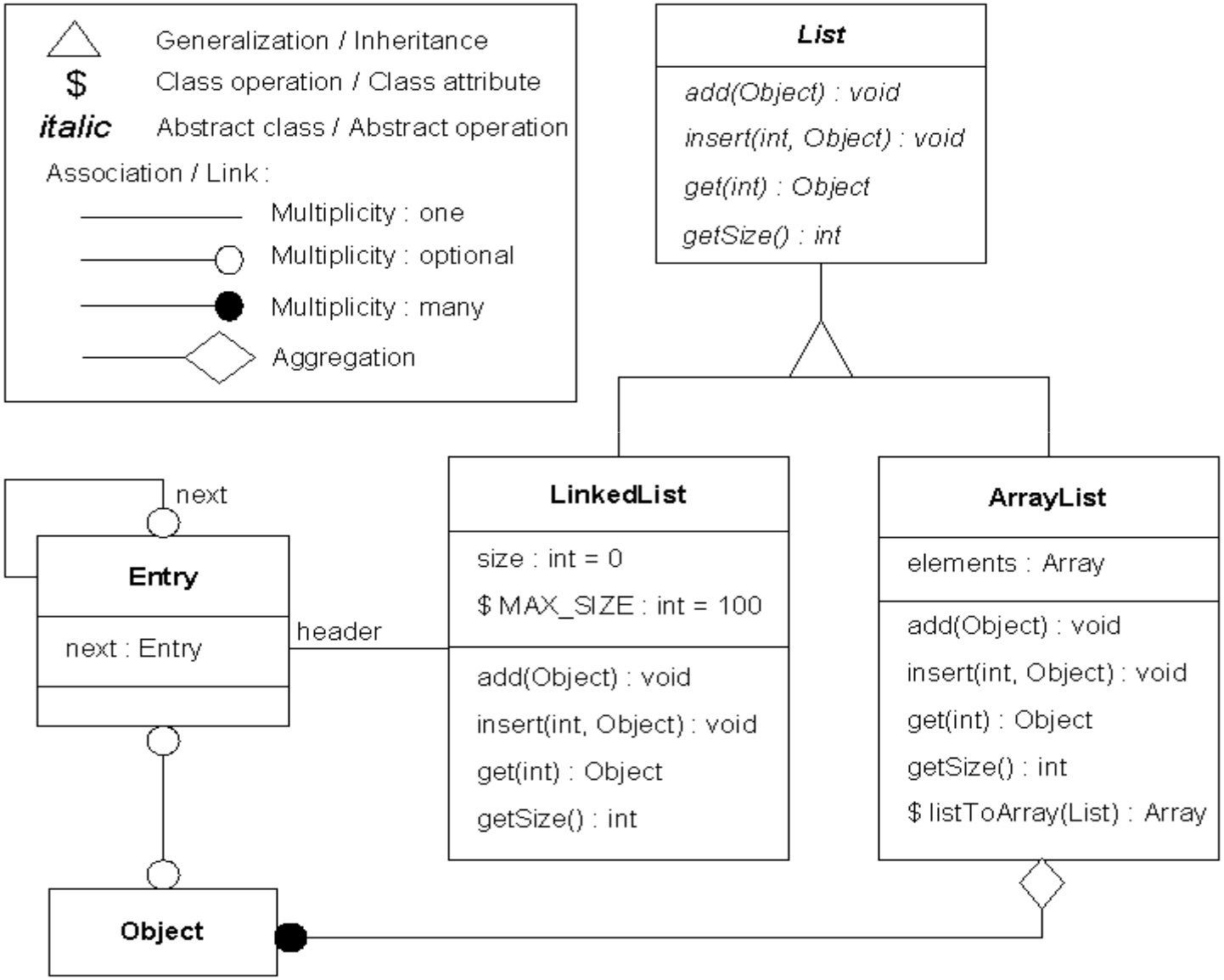
OMT object model

- Appendix B of the GoF book.

(a) Abstract and concrete classes

(b) Participant Client class (left) and implicit Client class (right)

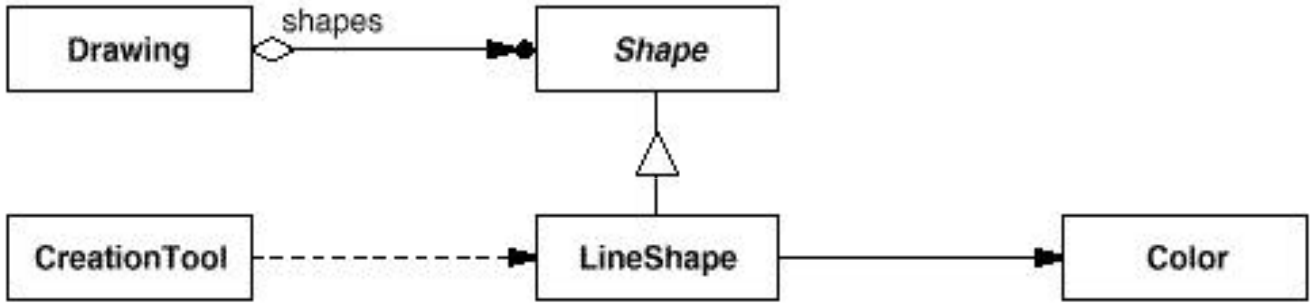
OMT object model (continued)



OMT object model (continued)

reference (do not use associations) when describing DP

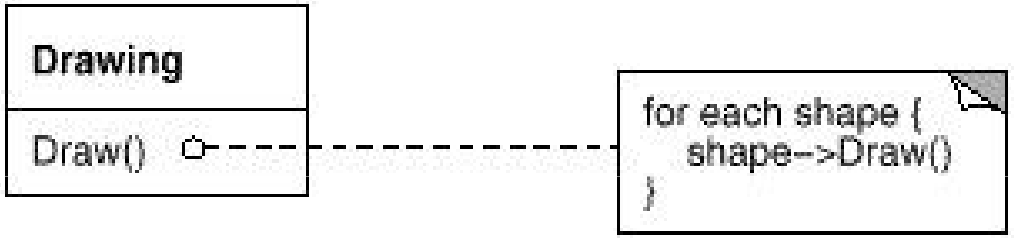
instantiation relation



(c) Class relationships

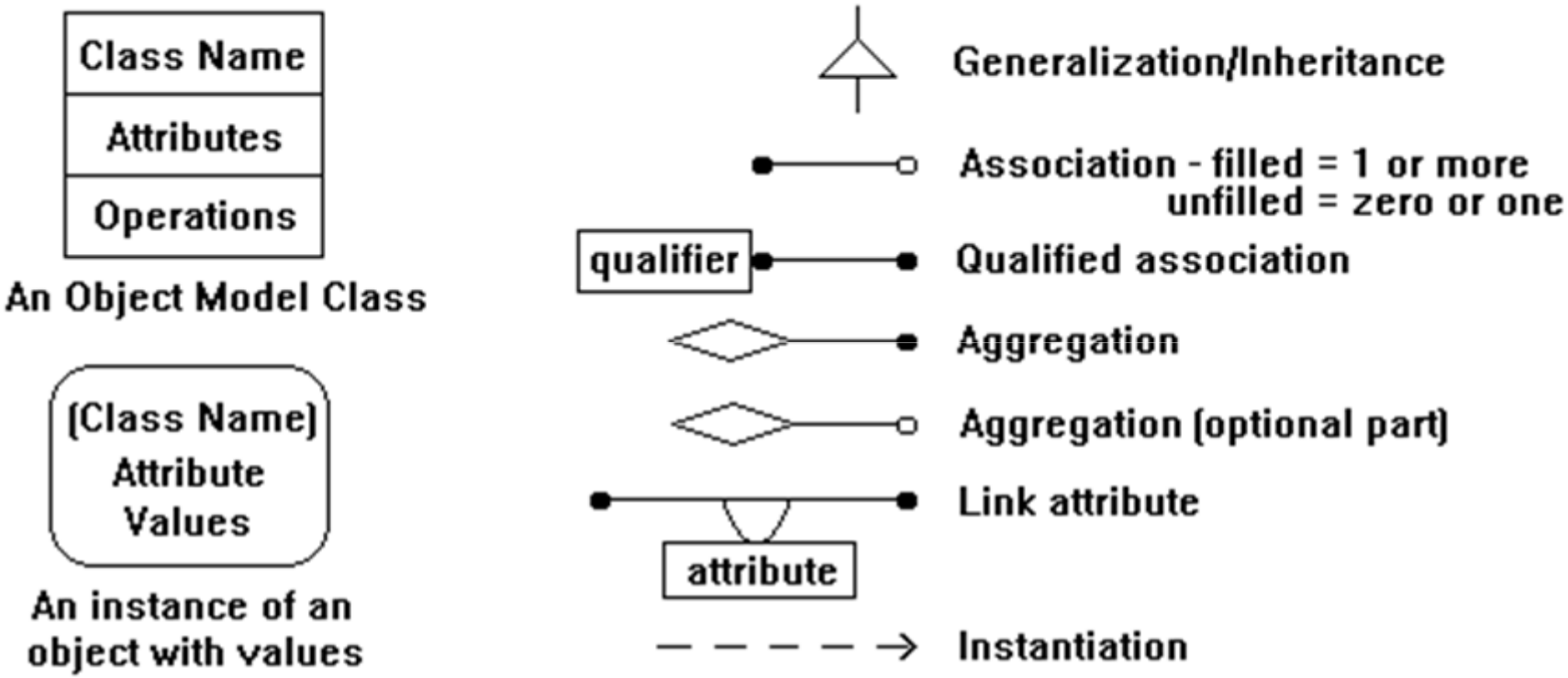
OMT object model (continued)

anchor a note

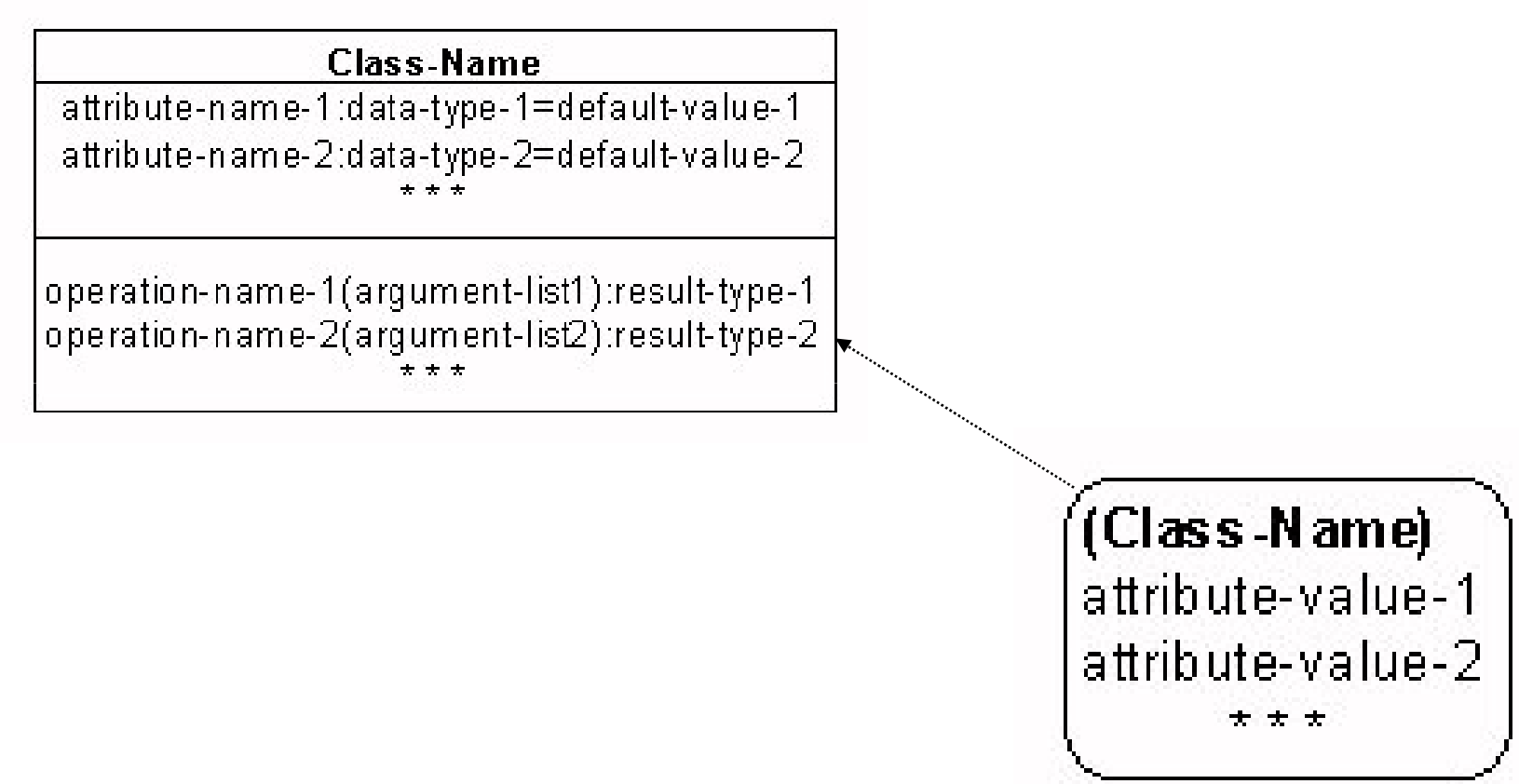


(d) Pseudocode annotation

OMT object model (continued)



Classes & instances (metadata & data)

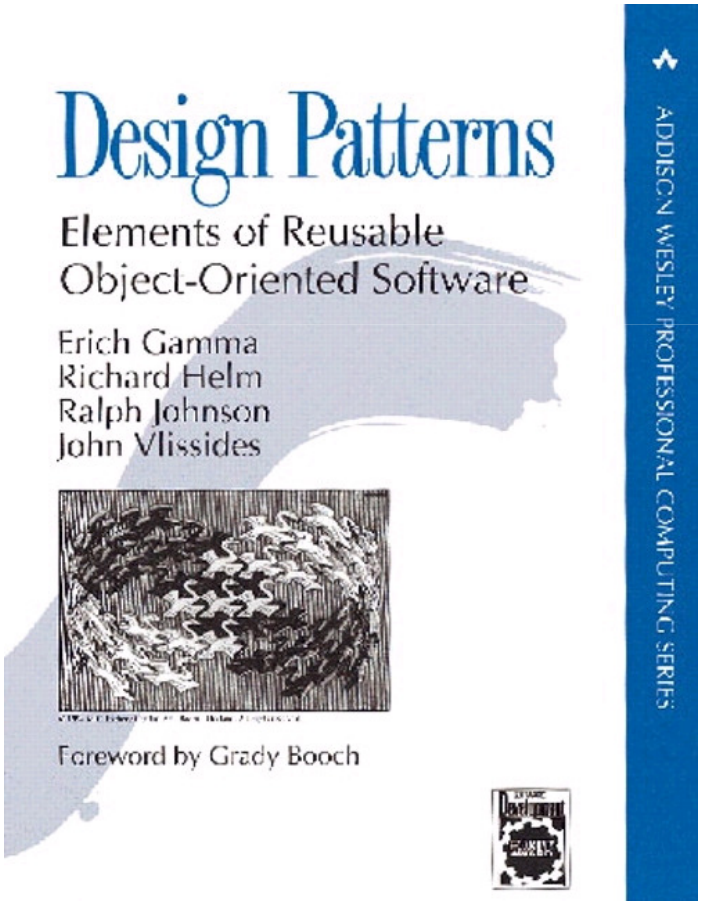


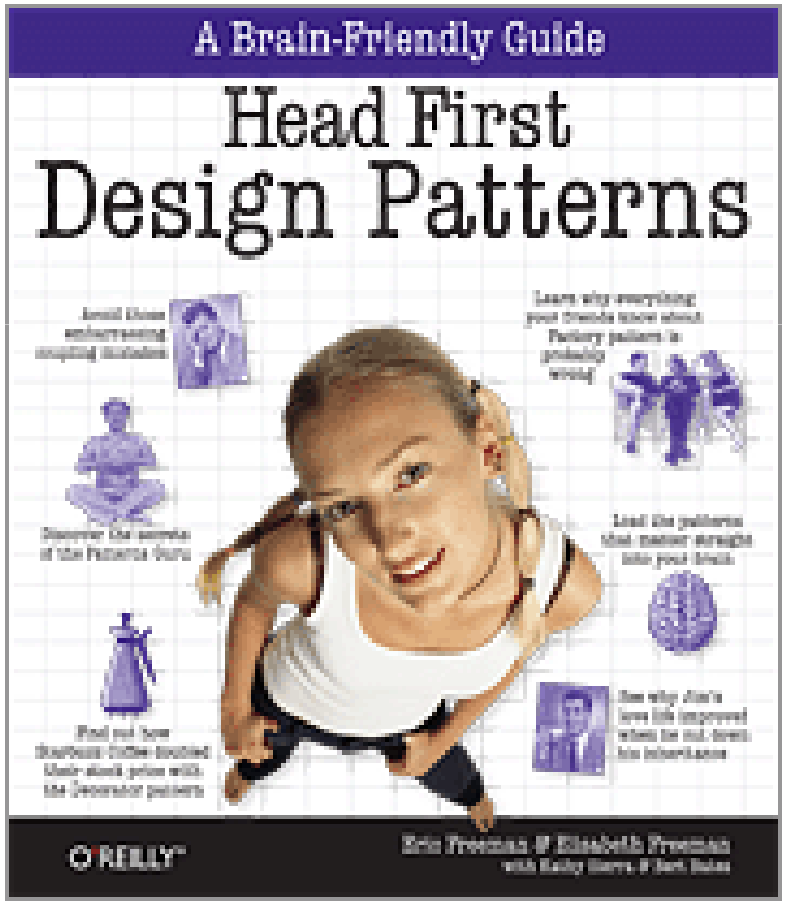
A citation

Good design and programming is not learned by generalities, but by seeing how significant programs can be made clean, easy to read, easy to maintain and modify, human-engineered, efficient, and reliable, by the application of good design and programming practices. Careful study and imitation of good designs and programs significantly improves development skills.

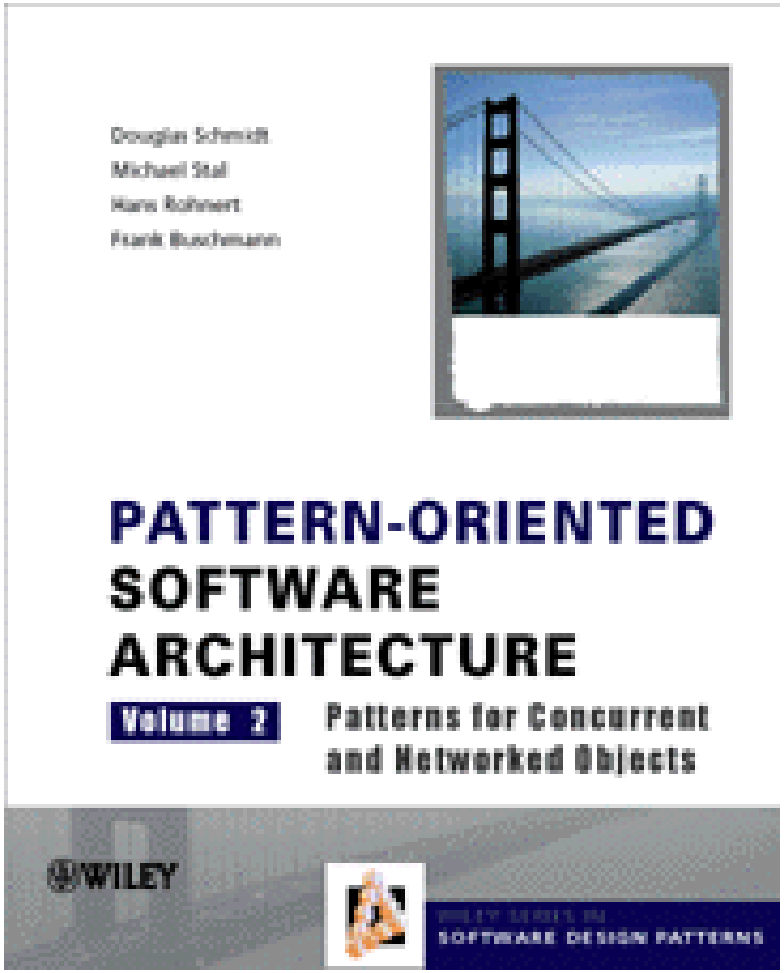
- Kernighan and Plauger

Libri


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON WESLEY PROFESSIONAL COMPUTING SERIES


A Brain-Friendly Guide
Head First
Design Patterns
O'REILLY

An extra book (?)



ORARIO e RICEVIMENTO

- Problemi di orario?
- Ricevimento da spostare