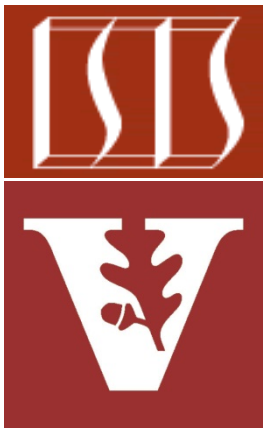
Overview of Patterns: Introduction

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



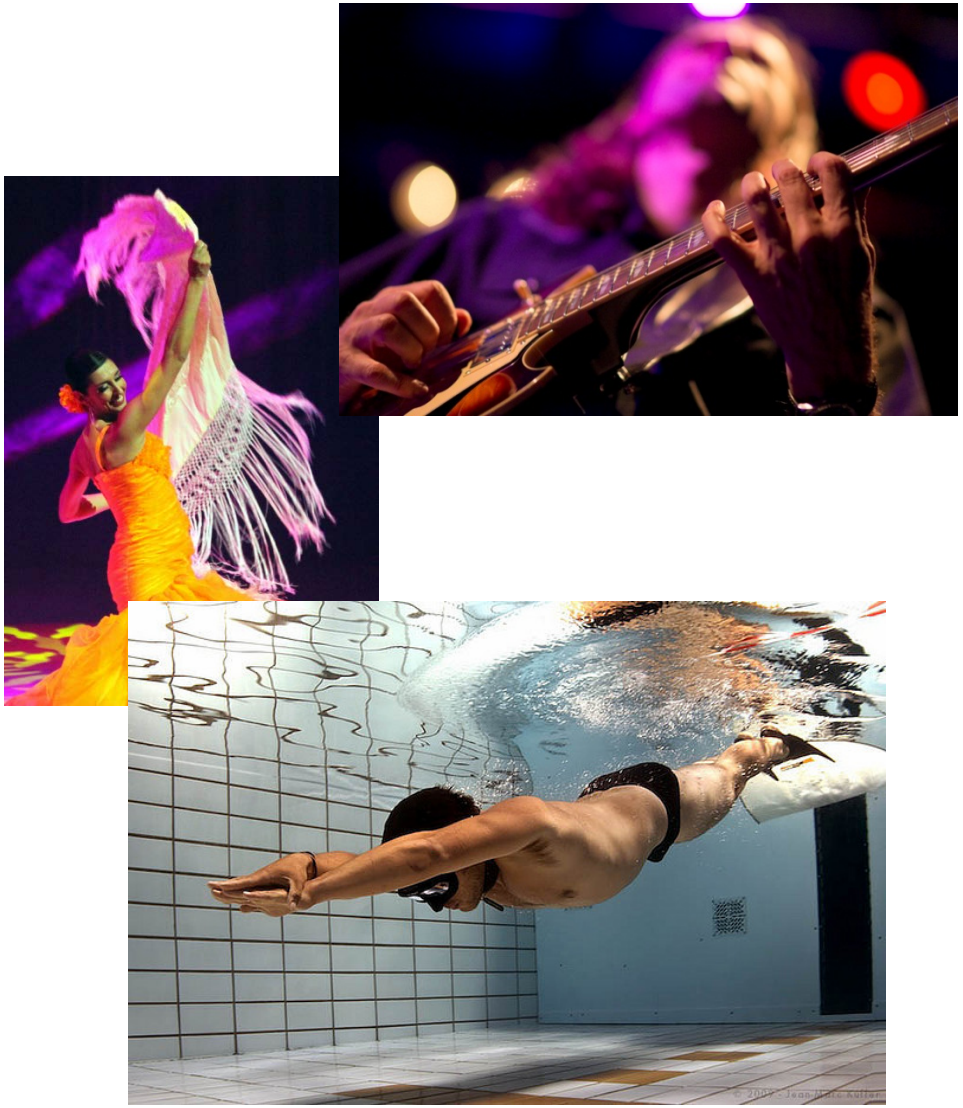
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Introduction

- Experts perform differently than beginners
  - Unlike novices, professional athletes, musicians, & dancers move fluidly & effortlessly, without focusing on each individual movement



Introduction

- Experts perform differently than beginners
- When watching experts perform it's easy to forget how much effort they've put into reaching high levels of achievement

Introduction

- Experts perform differently than beginners
- When watching experts perform it's easy to forget how much effort they've put into reaching high levels of achievement
- Continuous repetition & practice are crucial to their success

Introduction

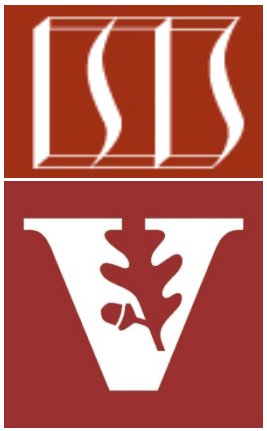
- Experts perform differently than beginners
- When watching experts perform it's easy to forget how much effort they've put into reaching high levels of achievement
- Continuous repetition & practice are crucial to their success
- Mentoring from other experts is also essential to achieving mastery

Introduction

At the heart of all these activities is knowledge & mastery of *patterns*

Overview of Patterns: Part 1

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



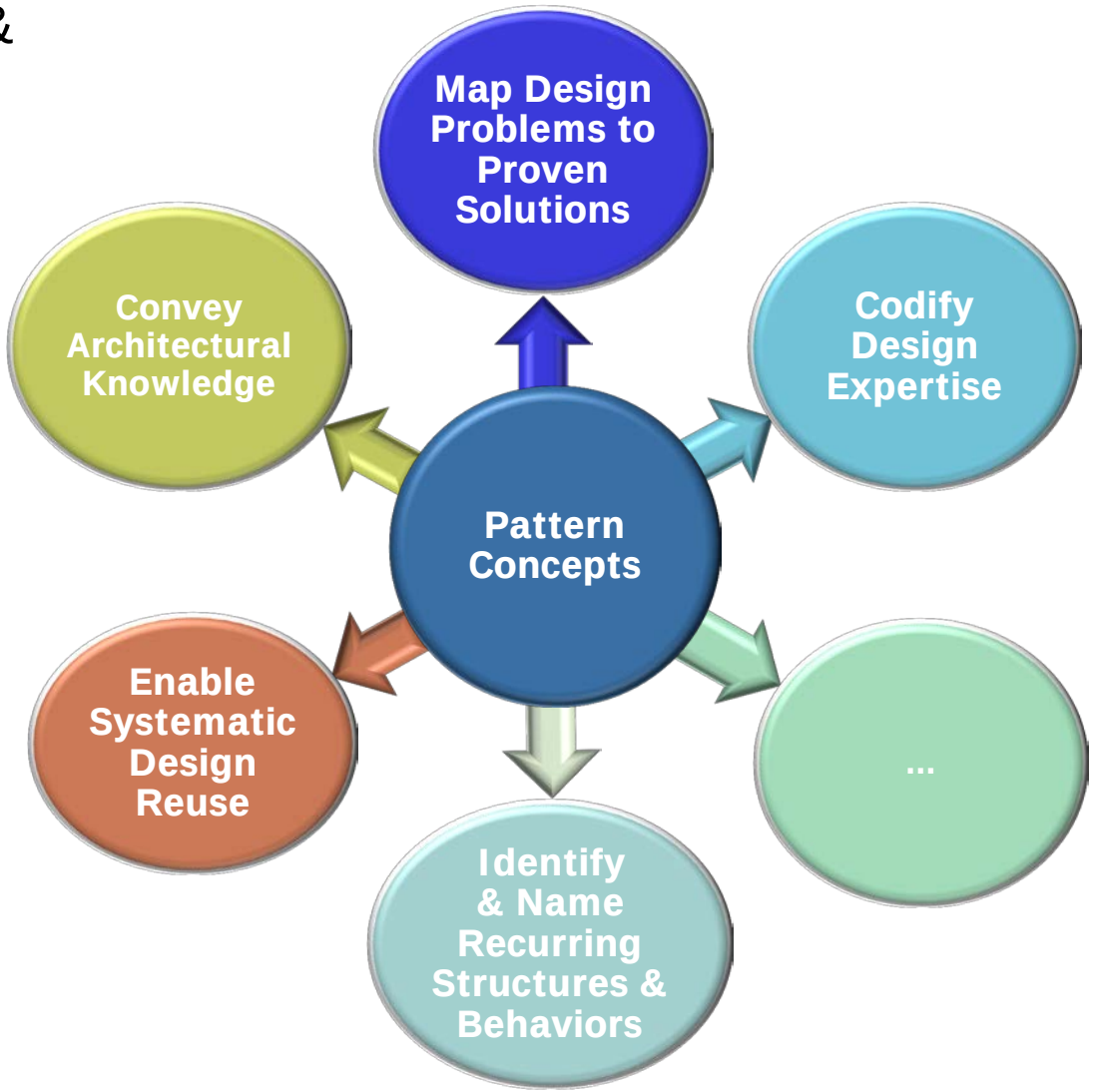
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

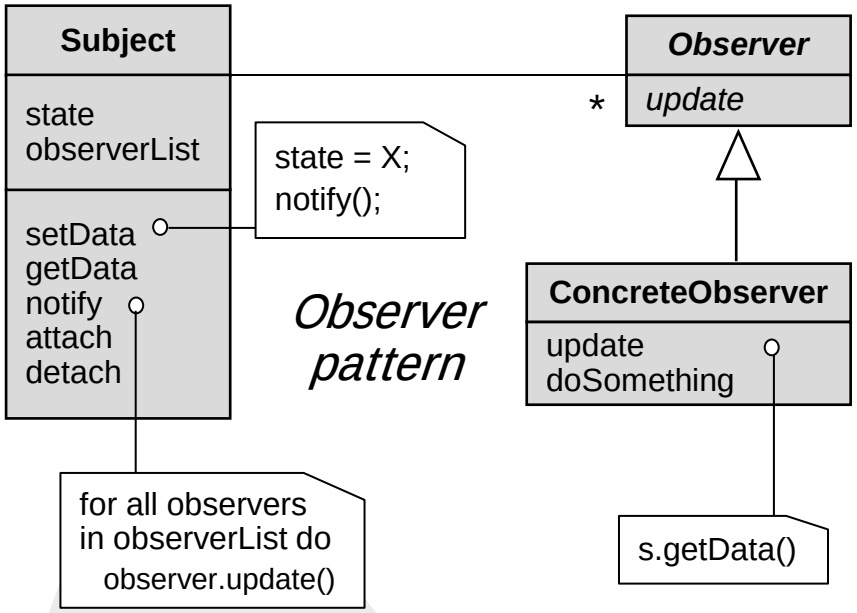
Learning Objectives in this Part of the Module

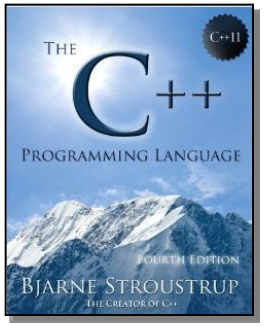
- Understand what patterns are & how they codify design experience to help improve quality & productivity

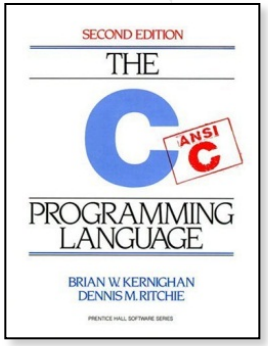


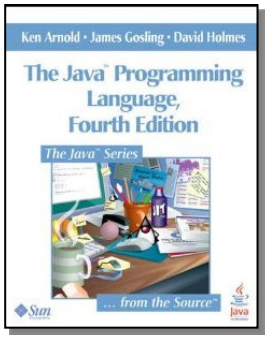
Learning Objectives in this Part of the Module

- Understand what patterns are & how they codify design experience to help improve quality & productivity
- Identify common characteristics of patterns & pattern descriptions









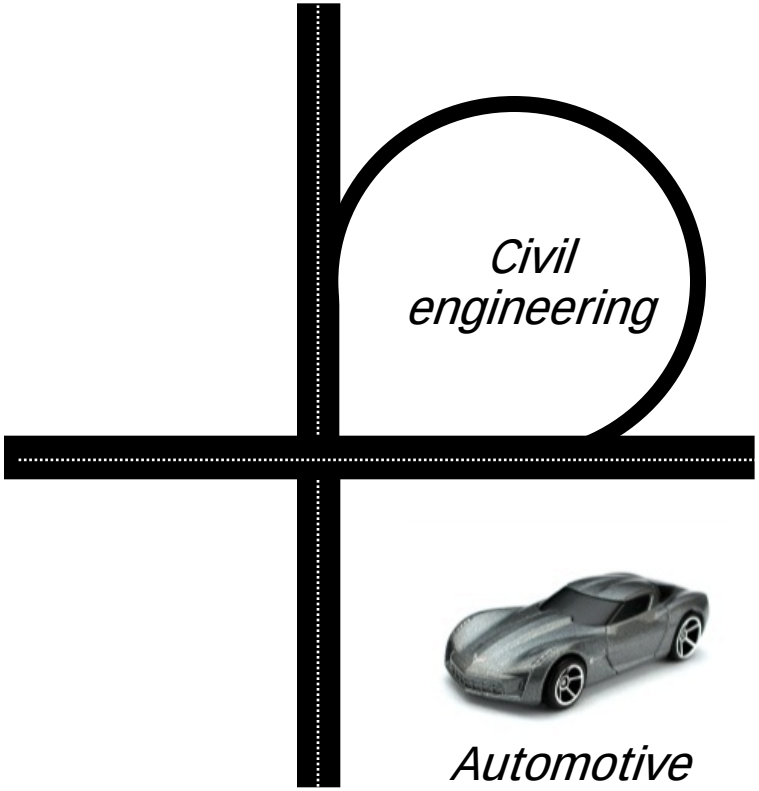
What are Software Patterns?

- Describes a **solution** to a common **problem** arising within a **context**

Mobile devices

Aerospace

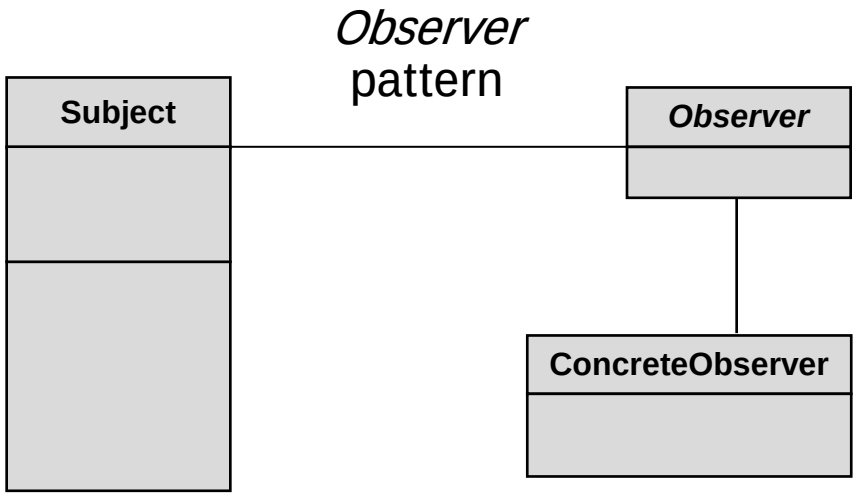
Electronic Trading



Automotive

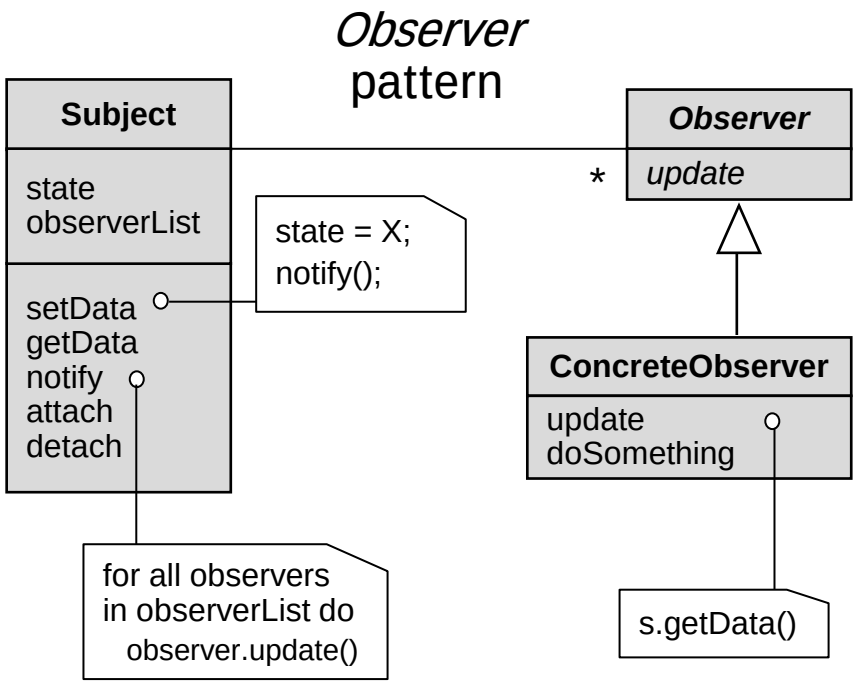
What are Software Patterns?

- Patterns help improve software quality and developer productivity by
  - **Naming** recurring design structures
    - e.g., the *Observer* pattern "defines a one-to-many dependency between objects so that when one object changes state, all dependents are notified & updated"



What are Software Patterns?

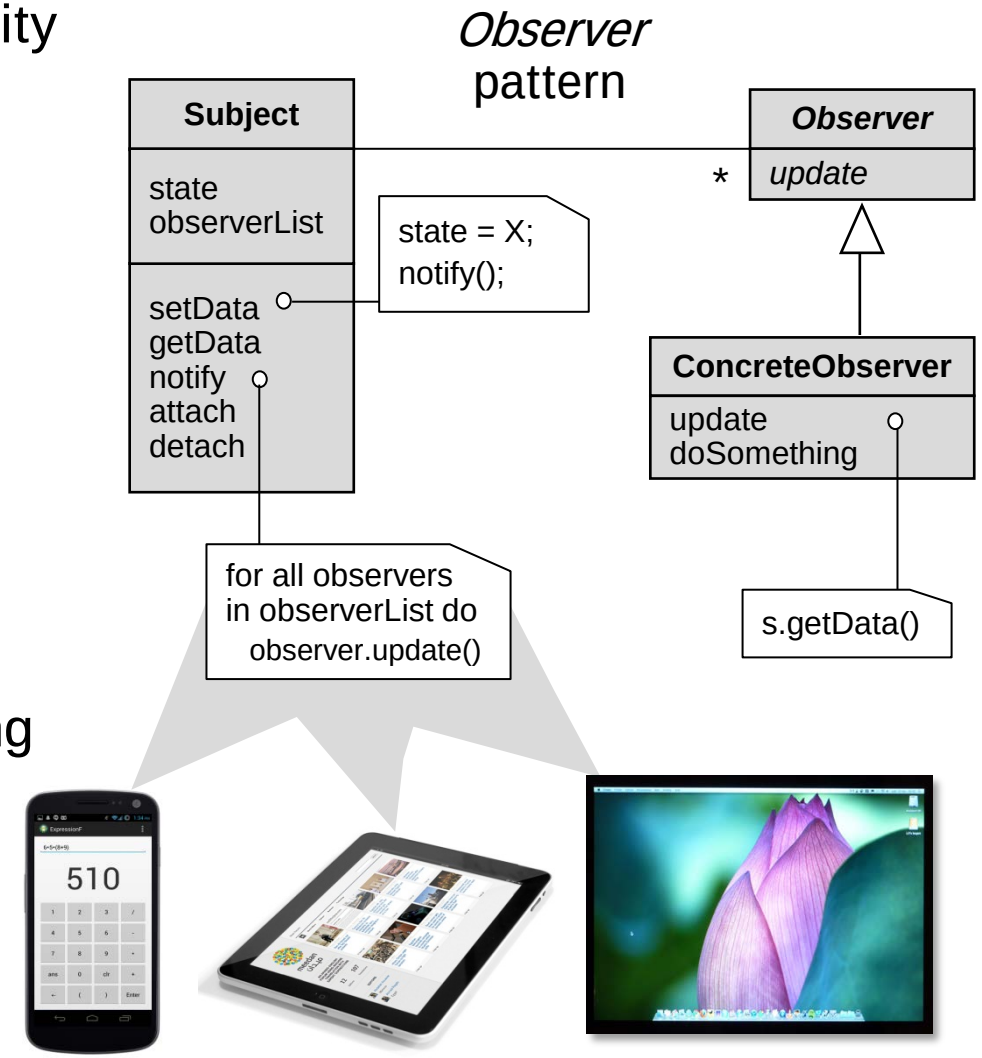
- Patterns help improve software quality and developer productivity by
  - **Naming** recurring design structures
  - **Specifying** design structure explicitly by identifying key properties of classes/objects, e.g.:
    - roles & relationships
    - dependencies
    - interactions
    - conventions



Interpret *class* & *object* loosely: patterns are for more than OO languages!

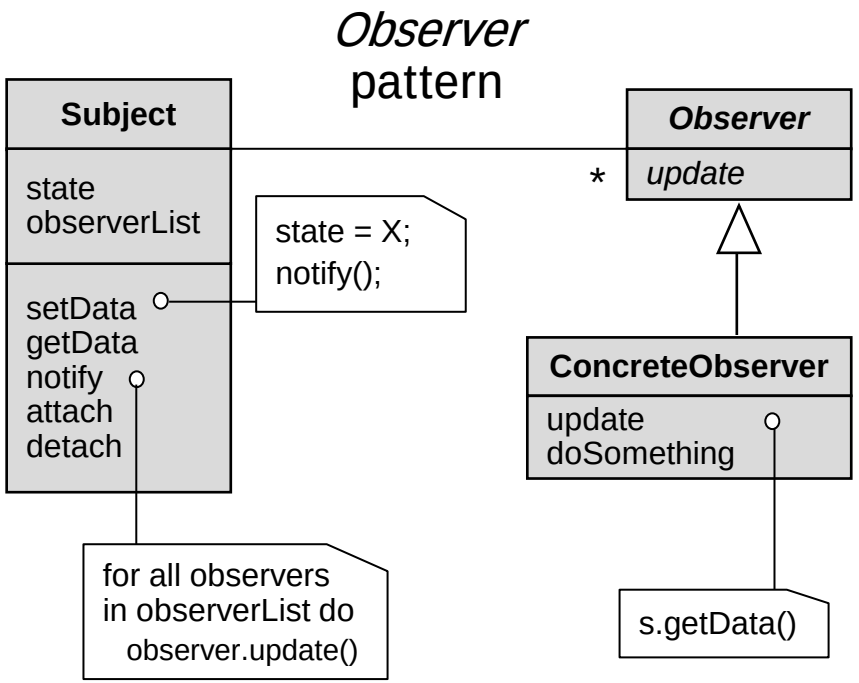
What are Software Patterns?

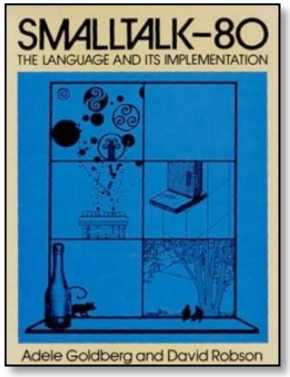
- Patterns help improve software quality and developer productivity by
  - **Naming** recurring design structures
  - **Specifying** design structure explicitly by identifying key properties of classes/objects
  - **Abstracting** from concrete design elements
    - For example, problem domain, form factor, vendor, programming language, etc.

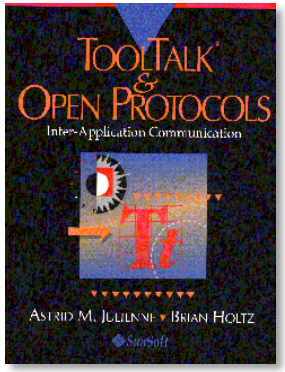


What are Software Patterns?

- Patterns help improve software quality and developer productivity by
  - **Naming** recurring design structures
  - **Specifying** design structure explicitly by identifying key properties of classes/objects
  - **Abstracting** from concrete design elements
  - **Distilling & codifying knowledge** gleaned by experts from their successful design experience







Patterns help avoid “reinventing the wheel” for common software problems

Common Characteristics of Patterns

- They describe both a *thing* & a *process*:
  - The "thing" (the "what") typically means a particular high-level design outline or description of implementation details

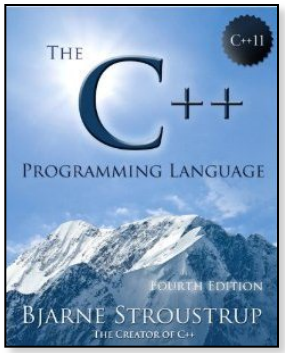
Common Characteristics of Patterns

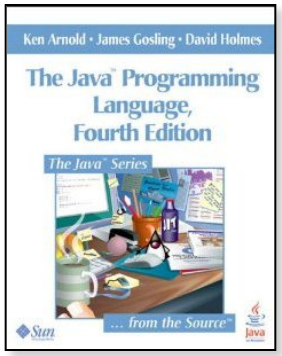
- They describe both a *thing* & a *process*:
  - The "process" (the "how") typically describes the steps to perform to create the "thing"

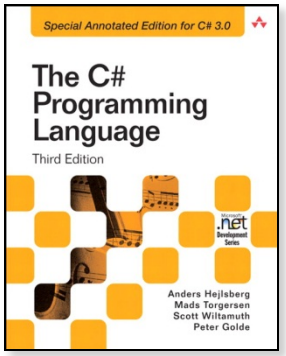
csis.pace.edu/~bergin/dcs/SoftwarePatterns_Coplien.pdf has more info

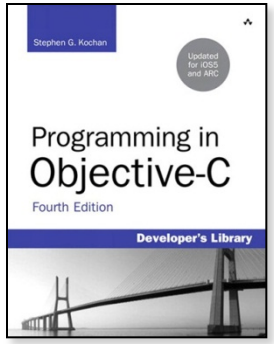
Common Characteristics of Patterns

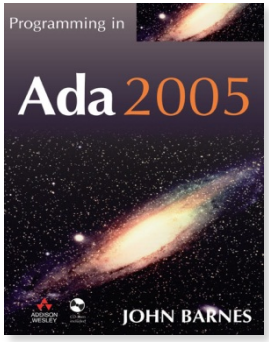
- They describe both a *thing* & a *process*
- They can be independent of programming languages & implementation techniques

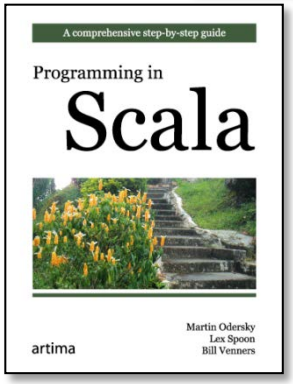


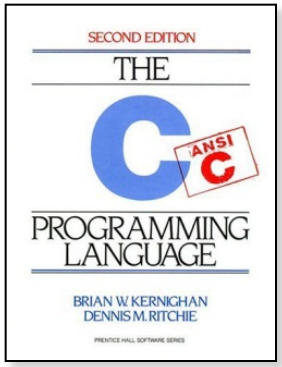


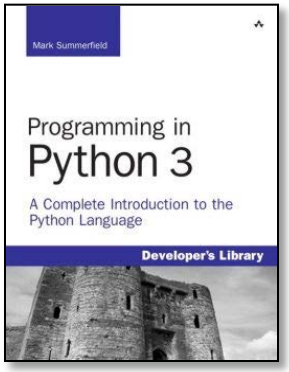










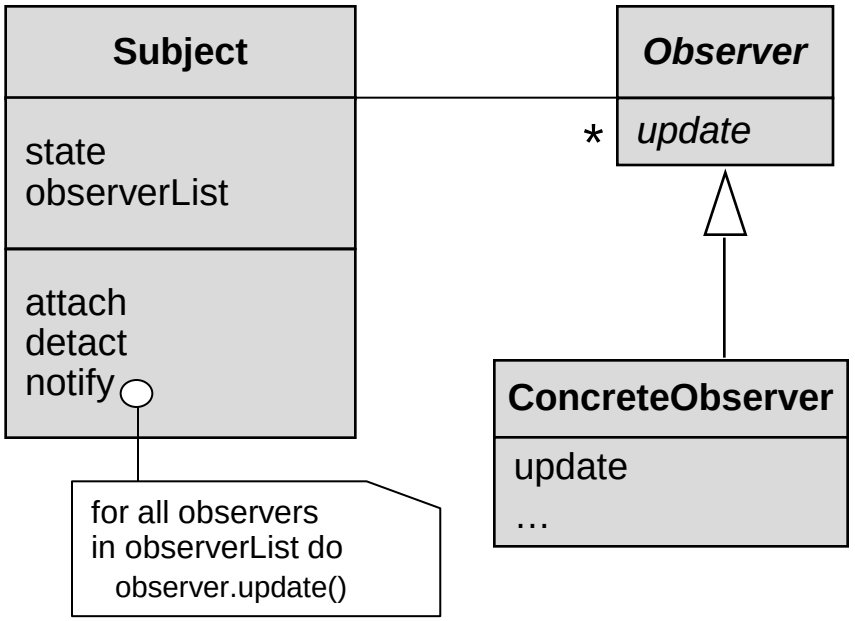


Naturally, different patterns apply to different programming languages

Common Characteristics of Patterns

- They describe both a *thing* & a *process*
- They can be independent of programming languages & implementation techniques
- They define "micro-architectures"
  - In other words, recurring design structure

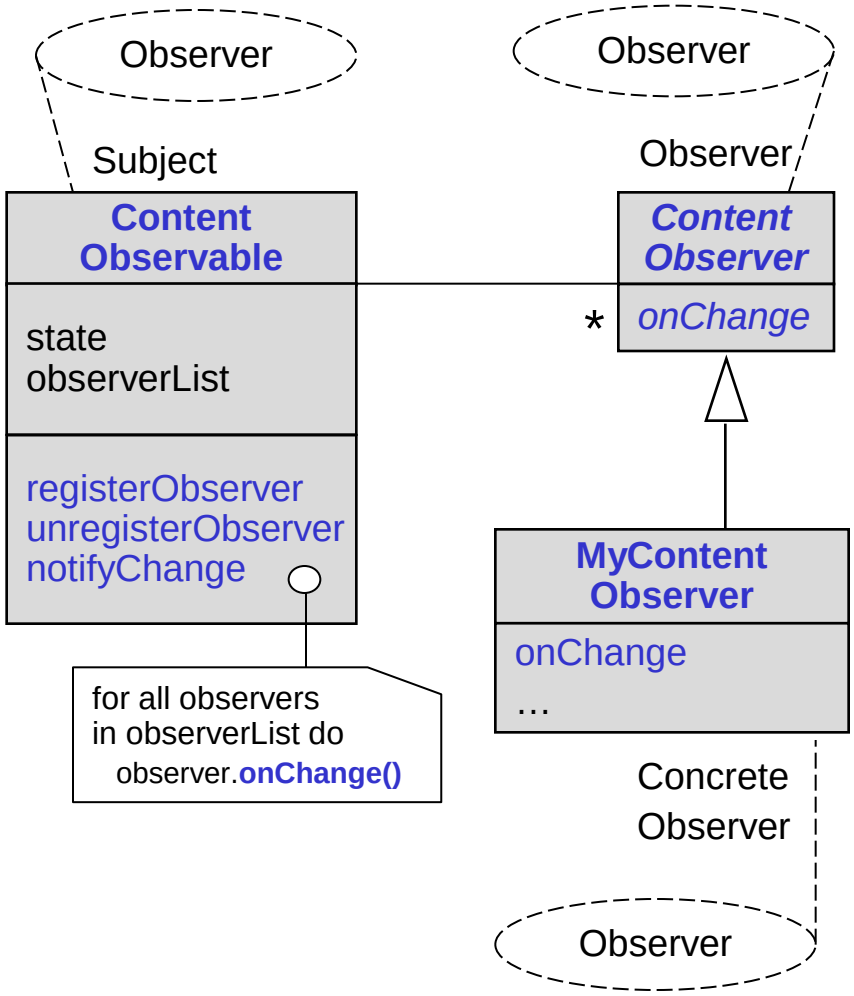
The *Observer* pattern



Common Characteristics of Patterns

- They describe both a *thing* & a *process*
- They can be independent of programming languages & implementation techniques
- They define "micro-architectures"
  - Certain properties may be modified for particular contexts

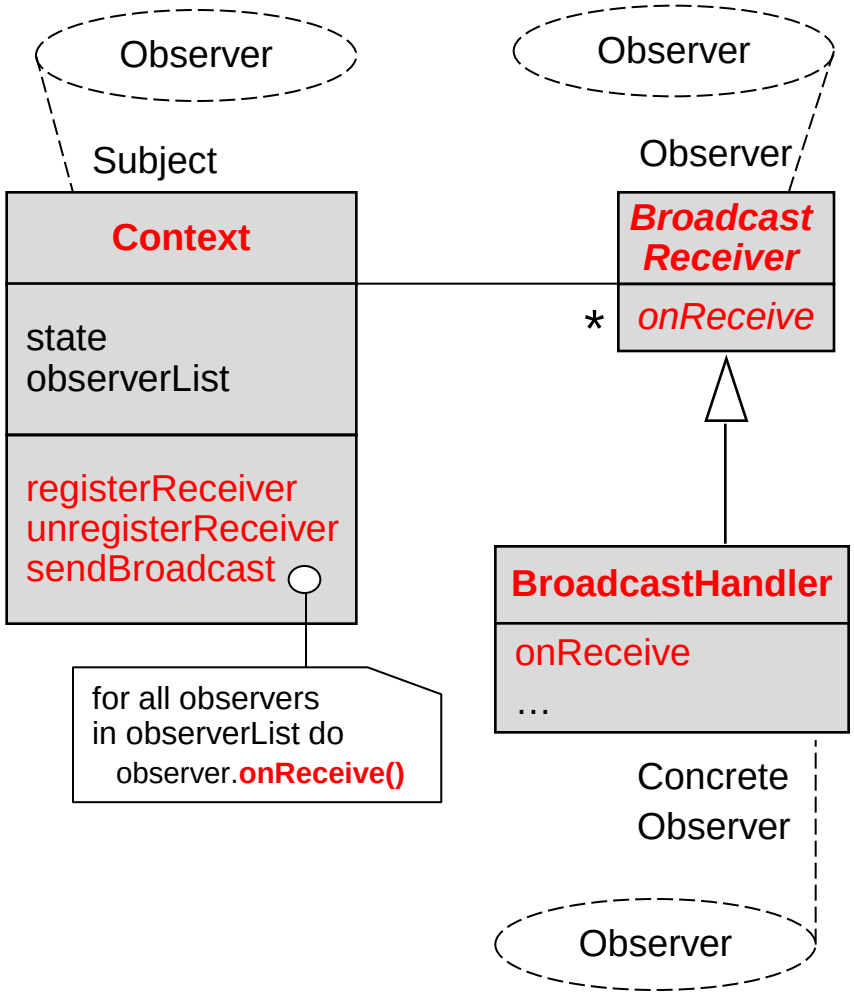
One use of the *Observer* pattern in Android



Common Characteristics of Patterns

- They describe both a *thing* & a *process*
- They can be independent of programming languages & implementation techniques
- They define "micro-architectures"
  - Certain properties may be modified for particular contexts

A different use of the *Observer* pattern in Android



Common Characteristics of Patterns

- They describe both a *thing* & a *process*
- They can be independent of programming languages & implementation techniques
- They define "micro-architectures"
- They aren't code or (concrete) designs, so they must be reified and applied in particular languages

The *Observer* pattern in Java

```
public class EventHandler
    extends Observer {
  public void update(Observable o,
                     Object arg)
  { /*…*/ }
    …

public class EventSource
    extends Observable,
    implements Runnable {
  public void run()
  { /*…*/ notifyObservers(/*…*/); }
    …

EventSource eventSource =
        new EventSource();
EventHandler eventHandler =
        new EventHandler();
eventSource.addObserver(eventHandler);
Thread thread =
   new Thread(eventSource);
thread.start();
…
```

en.wikipedia.org/wiki/Java_(programming_language) has more info on Java

Common Characteristics of Patterns

- They describe both a *thing* & a *process*
- They can be independent of programming languages & implementation techniques
- They define "micro-architectures"
- They aren't code or (concrete) designs, so they must be reified and applied in particular languages

The *Observer* pattern in C++ & ACE

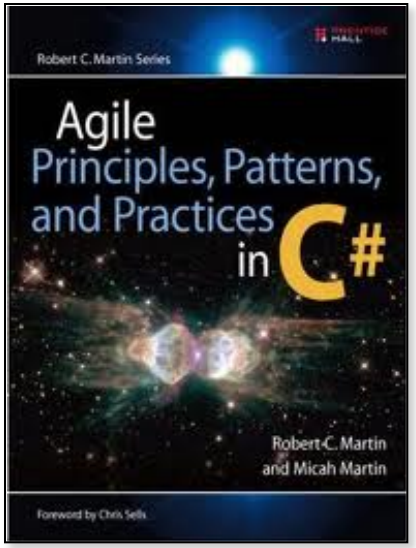
(uses the GoF *Bridge* pattern with reference counting to simplify memory management & ensure exception-safe semantics)

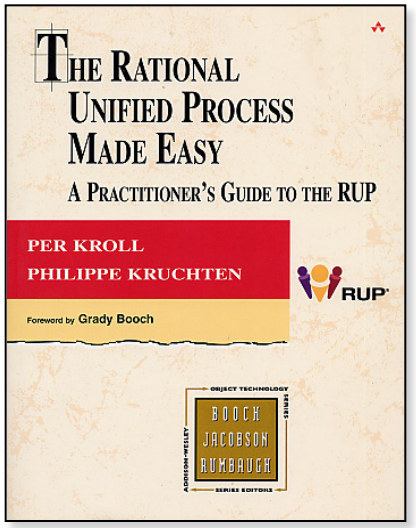
```
class Event_Handler
    : public Observer {
public:
  virtual void update(Observable o,
                      Object arg)
  { /* … */ }
    …
class Event_Source
    : public Observable,
      public ACE_Task_Base {
public:
  virtual void svc()
  { /*…*/ notify_observers(/*…*/); }
    …

Event_Source event_source;
Event_Handler event_handler;
event_source->add_observer
                      (event_handler);
Event_Task task (event_source);
task->activate();
…
```

Common Characteristics of Patterns

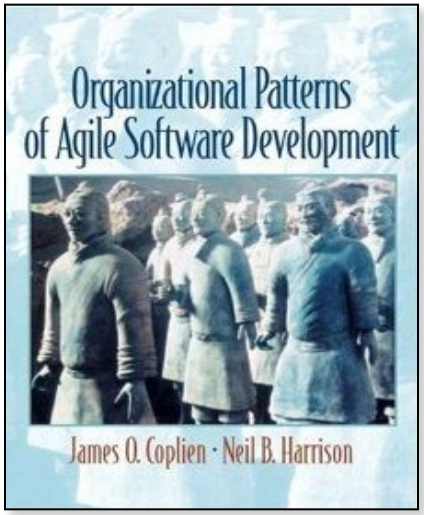
- They describe both a *thing* & a *process*
- They can be independent of programming languages & implementation techniques
- They define "micro-architectures"
- They aren't code or (concrete) designs, so they must be reified and applied in particular languages
- They are not methods but can be used as an adjunct to methods, e.g.:
  - Rational Unified Process
  - Agile
  - Others

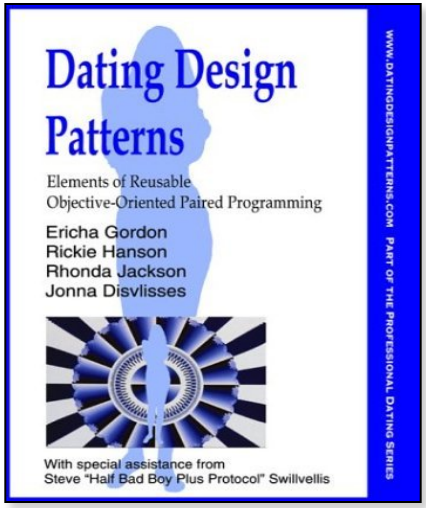




Common Characteristics of Patterns

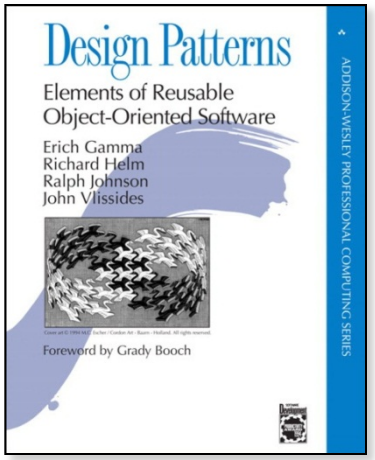
- They describe both a *thing* & a *process*
- They can be independent of programming languages & implementation techniques
- They define "micro-architectures"
- They aren't code or (concrete) designs, so they must be reified and applied in particular languages
- They are not methods but can be used as an adjunct to methods
- There are also patterns for organizing effective software development teams and navigating other complex settings

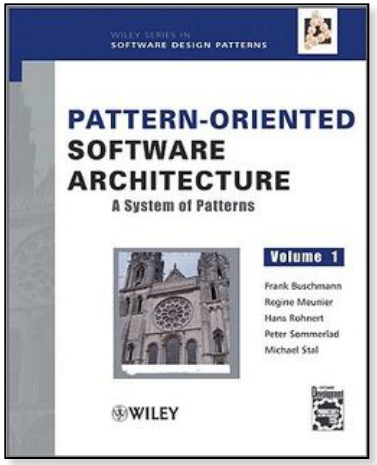


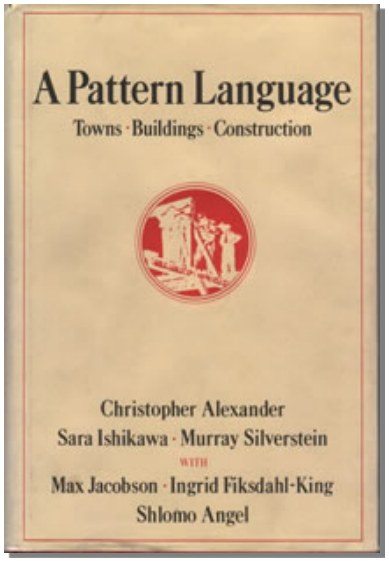


Common Parts of a Pattern Description

- **Name**
  - Should be pithy & memorable

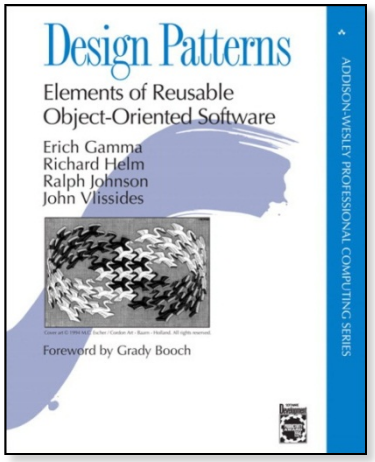


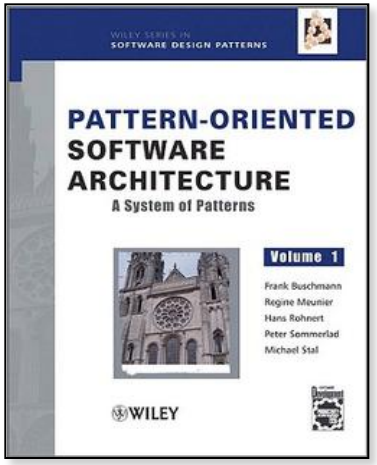


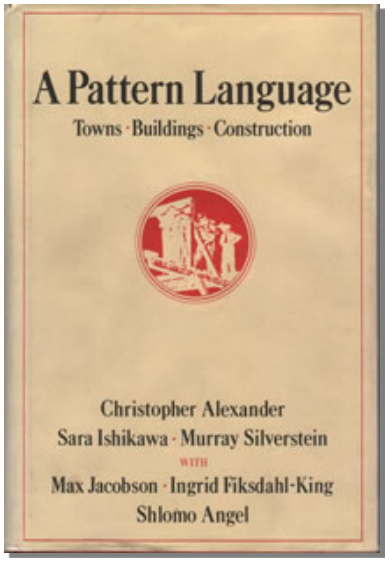


Common Parts of a Pattern Description

- **Intent**
  - Goal behind the pattern & the reason(s) for using it

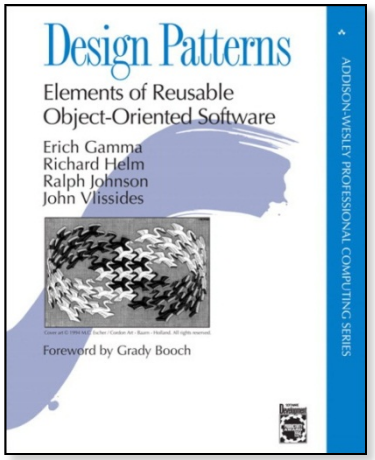


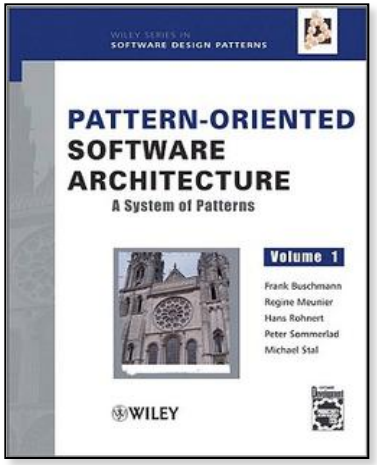


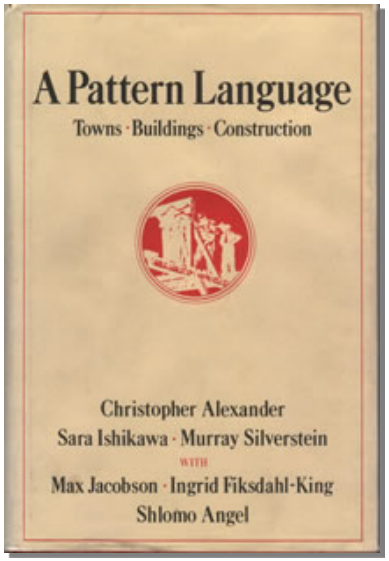


Common Parts of a Pattern Description

- **Problem** addressed by pattern
  - Motivate the “forces” & situations in which pattern is applicable



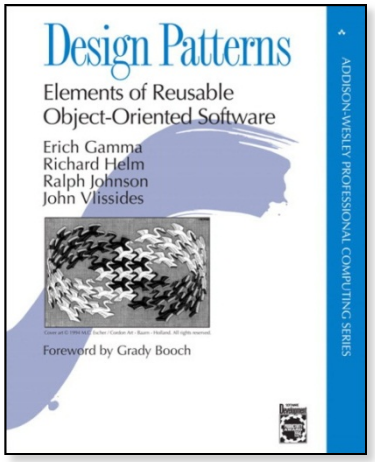


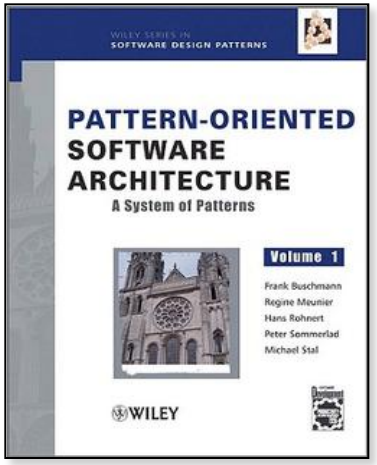


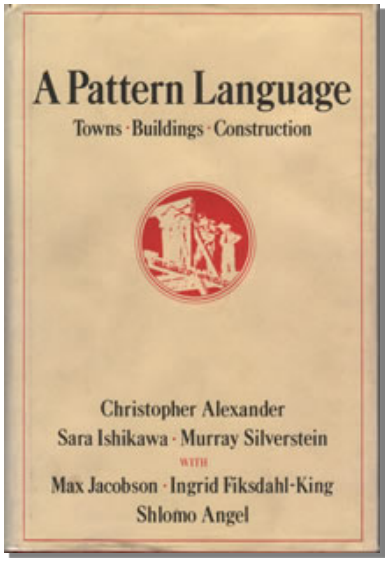
See c2.com/cgi/wiki?PatternForms for more info on pattern forms

Common Parts of a Pattern Description

- **Solution**
  - Visual & textual descriptions of pattern static structure, participants, and collaboration dynamics

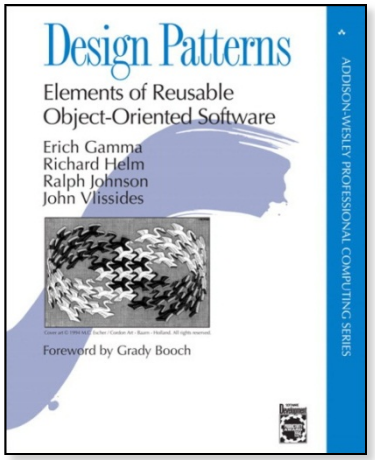


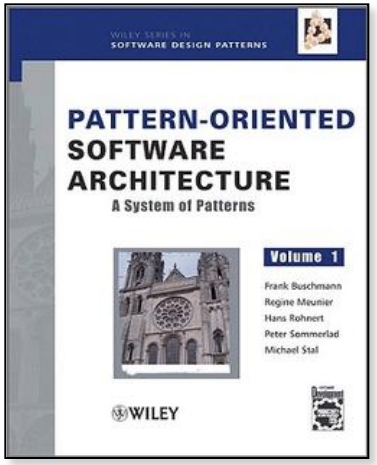


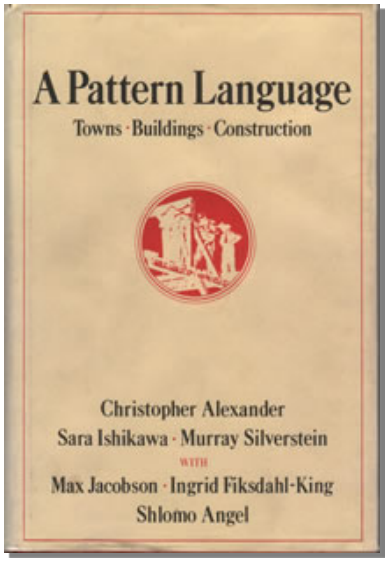


Common Parts of a Pattern Description

- **Examples** & **Implementation guidance**
  - May include source code snippets in one or more programming languages

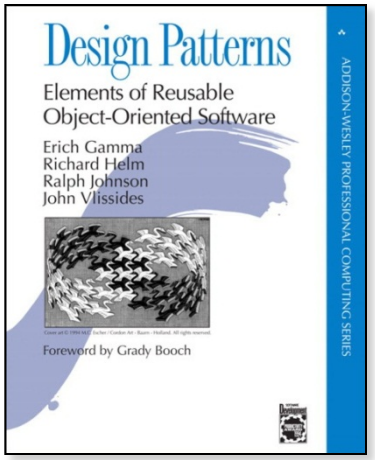


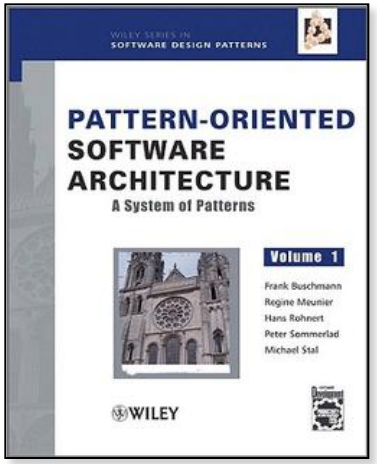


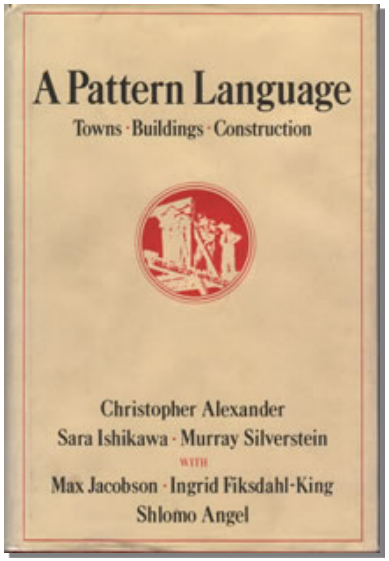


Common Parts of a Pattern Description

- **Consequences**
  - Benefits & liabilities of applying the pattern

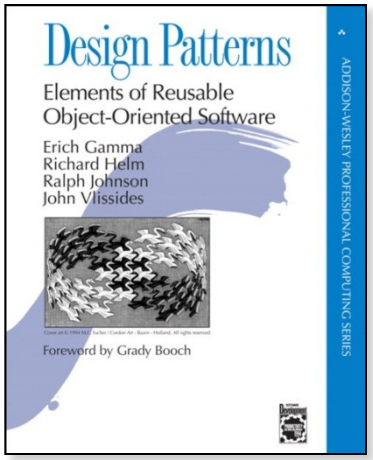


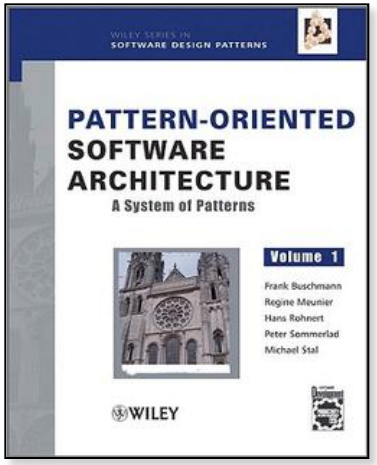


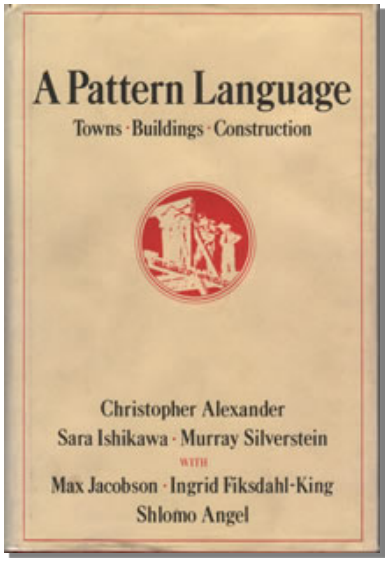


Common Parts of a Pattern Description

- **Known uses**
  - Examples of real uses of the pattern
  - Should follow the “rule of three”

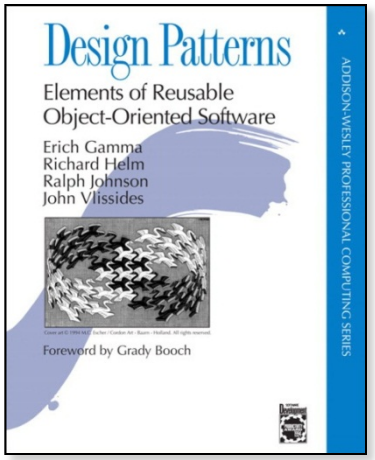


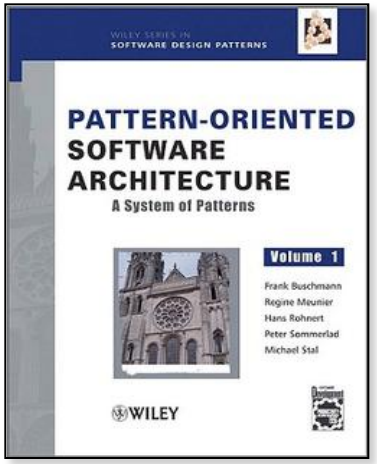


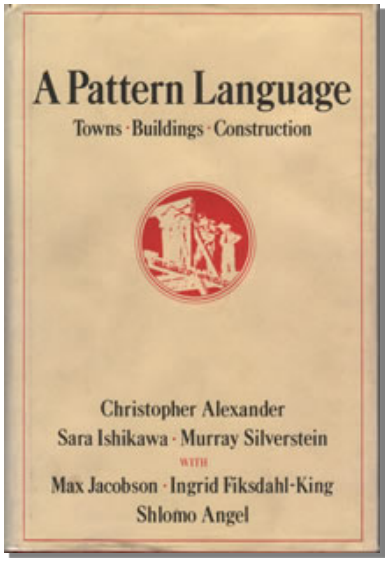


Common Parts of a Pattern Description

- **Related patterns**
  - Summarize relationships & tradeoffs between alternative patterns for similar problems

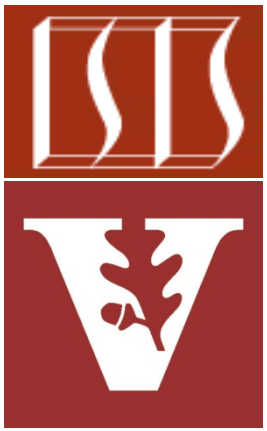






Overview of Patterns: Part 2

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



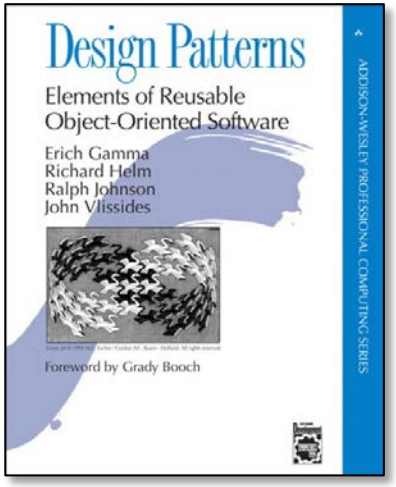
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

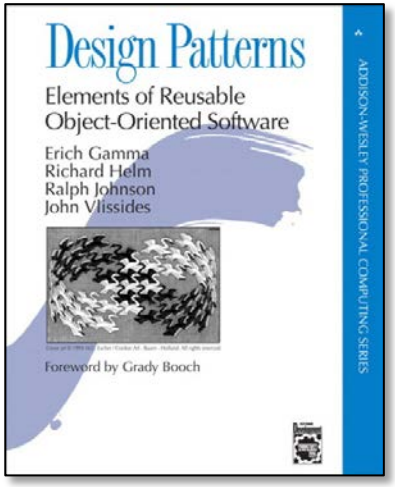
Learning Objectives in this Part of the Module

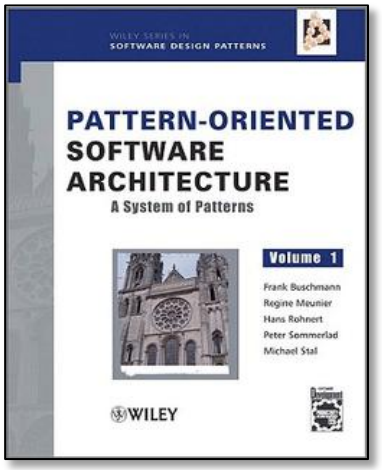
- Understand the history & contents of
  - *Design Patterns: Elements of Reusable Object-Oriented Software*
    - Commonly known as the "Gang of Four" (GoF) book



Learning Objectives in this Part of the Module

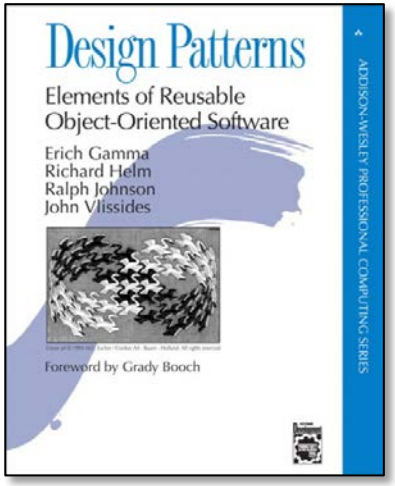
- Understand the history & contents of
  - *Design Patterns: Elements of Reusable Object-Oriented Software*
    - Commonly known as the "Gang of Four" (GoF) book
  - *Pattern-Oriented Software Architecture: A System of Patterns*
    - Commonly known as the "POSA1" book

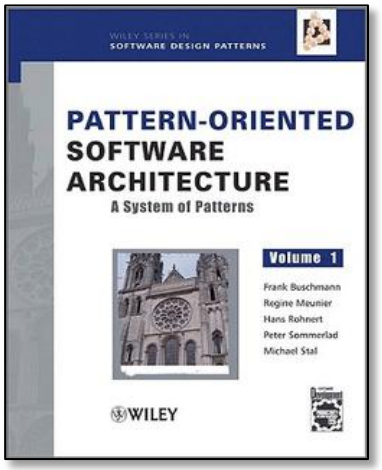




Learning Objectives in this Part of the Module

- Understand the history & contents of
  - *Design Patterns: Elements of Reusable Object-Oriented Software*
    - Commonly known as the "Gang of Four" (GoF) book
  - *Pattern-Oriented Software Architecture: A System of Patterns*
    - Commonly known as the "POSA1" book
  - *Pattern-Oriented Software Architecture: Patterns for Concurrent & Networked Objects*
    - Commonly known as the "POSA2" book





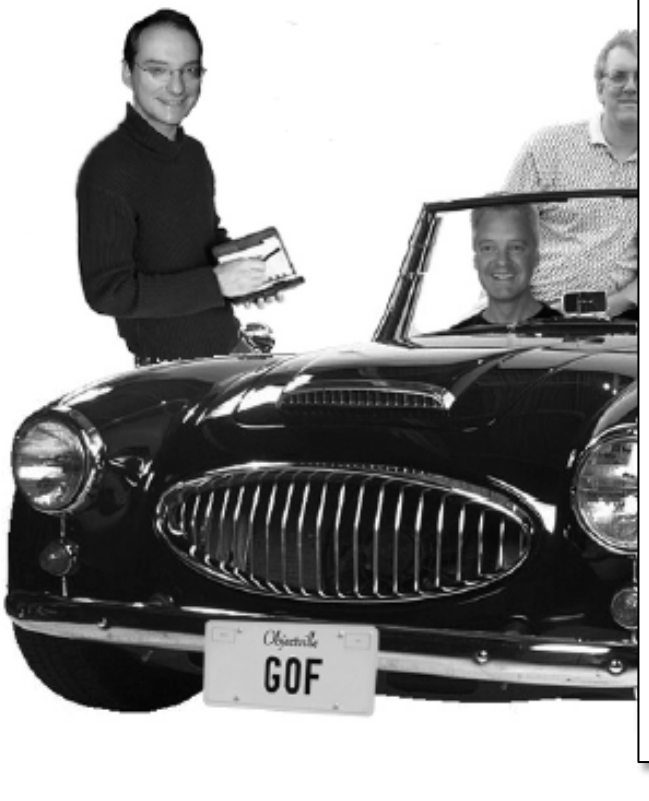
History of the GoF & POSA Pattern Books

- 1991 – Erich Gamma completes his PhD dissertation on patterns for GUIs

www.informatik.uni-trier.de/~ley/pers/hd/g/Gamma:Erich.html has more info

History of the GoF & POSA Pattern Books

- 1991 – Erich Gamma completes his PhD dissertation on patterns for GUIs
- 1992-1993 – Gang-of-Four participate in "Towards an Architecture Handbook" workshop at OOPSLA



OOPSLA'92
Vancouver, British Columbia, Canada
5 – 10 October 1992

Addendum to the Proceedings

Workshop Report—
Towards an Architecture Handbook

Report by:
Bruce Anderson,
University of Essex

Introduction

This report describes the workshop, but also describes the context in which it took place. We did not use the workshop (nor its precursor at OOPSLA'91) to present our work (which we did by reading the position papers), but to work together. Such an environment does not automatically produce outputs useful to those who did not attend, but is a source of emerging ideas, new approaches, and new relationships. These ideas, approaches and relationships generate the context described here.

Some Context

Architecture

We take architecture to mean the structuring paradigms, styles and patterns that make up our software systems. They are overarching or pervasive structures.

They are important in many ways: they allow us to talk usefully about systems without talking about their detail; a knowledge of them gives us design choices; attention to this level can make systems have the properties we want, especially reusability and extensibility

Of course every system has an architecture, but if it has not been consciously created then the system may well be more muddled, both conceptually and in its code, than is necessary.

We do not define architecture. Like Health or Quality, we can reach a productive shared understanding for the purpose at hand without making a definition. Different groups will have different understandings, and these will be reflected in different practices: processes, checklists, libraries, diagrams. And different handbooks.

Note also that a system may contain several architectures, and that it may have a different architecture when viewed from a different perspective. Many of our powerful yet ambiguous words may be brought into play: "we are using an object-oriented paradigm to build a generic-application framework via a delegation approach."

Handbooks

Once some level of consensus is reached in a community, it is possible, and worthwhile, to produce a handbook. So a company (for internal use) or a publisher (for sale) can find someone (an editor) to put together such a book. The source materials for such a book would be found in the community; there would be technical issues of choosing versions and values, but these would come mainly from the already-existing consensus of practitioners.

There is currently no handbook of software architecture available, because at the moment there is no universal, or even large, community of software architects, so any architecture handbook would have to be based instead on the theories of its author.

5–10 October 1992 - 109 - Addendum to the Proceedings

c2.com/cgi/wiki?ArchitectureHandbookWorkshop has more info

History of the GoF & POSA Pattern Books

- 1991 – Erich Gamma completes his PhD dissertation on patterns for GUIs
- 1992-1993 – Gang-of-Four participate in "Towards an Architecture Handbook" workshop at OOPSLA
- 1993 – GoF publish their first paper at ECOOP

Design Patterns: Abstraction and Reuse of Object-Oriented Design

Erich Gamma[1*], Richard Helm[2], Ralph Johnson[3], John Vlissides[2]

[1] Taligent, Inc.
10725 N. De Anza Blvd., Cupertino, CA 95014-2000 USA

[2] I.B.M. Thomas J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598 USA

[3] Department of Computer Science
University of Illinois at Urbana-Champaign
1034 W. Springfield Ave., Urbana, IL 61801 USA

Abstract. We propose **design patterns** as a new mechanism for expressing object-oriented design experience. Design patterns identify, name, and abstract common themes in object-oriented design. They capture the intent behind a design by identifying objects, their collaborations, and the distribution of responsibilities. Design patterns play many roles in the object-oriented development process: they provide a common vocabulary for design, they reduce system complexity by naming and defining abstractions, they constitute a base of experience for building reusable software, and they act as building blocks from which more complex designs can be built. Design patterns can be considered reusable micro-architectures that contribute to an overall system architecture. We describe how to express and organize design patterns and introduce a catalog of design patterns. We also describe our experience in applying design patterns to the design of object-oriented systems.

1 Introduction

Design methods are supposed to promote good design, to teach new designers how to design well, and to standardize the way designs are developed. Typically, a design method comprises a set of syntactic notations (usually graphical) and a set of rules that govern how and when to use each notation. It will also describe problems that occur in a design, how to fix them, and how to evaluate a design. Studies of expert programmers for conventional languages, however, have shown that knowledge is not organized simply around syntax, but in larger conceptual structures such as algorithms, data structures and idioms [1, 7, 9, 27], and plans that indicate steps necessary to fulfill a particular goal [26]. It is likely that designers do not think about the notation they are using for recording the design. Rather, they look for patterns to match against plans, algorithms, data structures, and idioms they have learned in the past. Good designers, it appears, rely

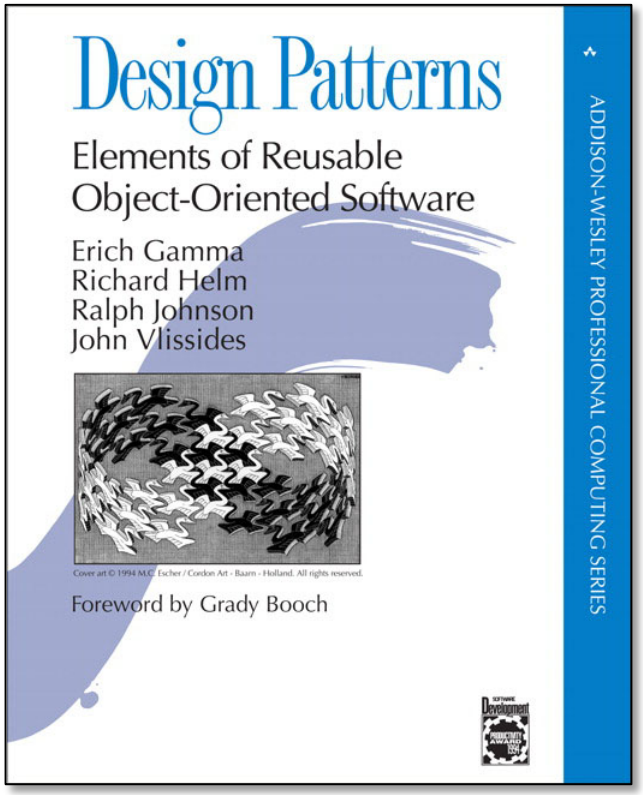
* Work performed while at *UBILAB*, Union Bank of Switzerland, Zurich, Switzerland.

O.M. Nierstrasz (Ed.): ECOOP '93, LNCS 707, pp. 406-431, 1993.

www.dre.vanderbilt.edu/~schmidt/original-GOF-patterns-paper.pdf

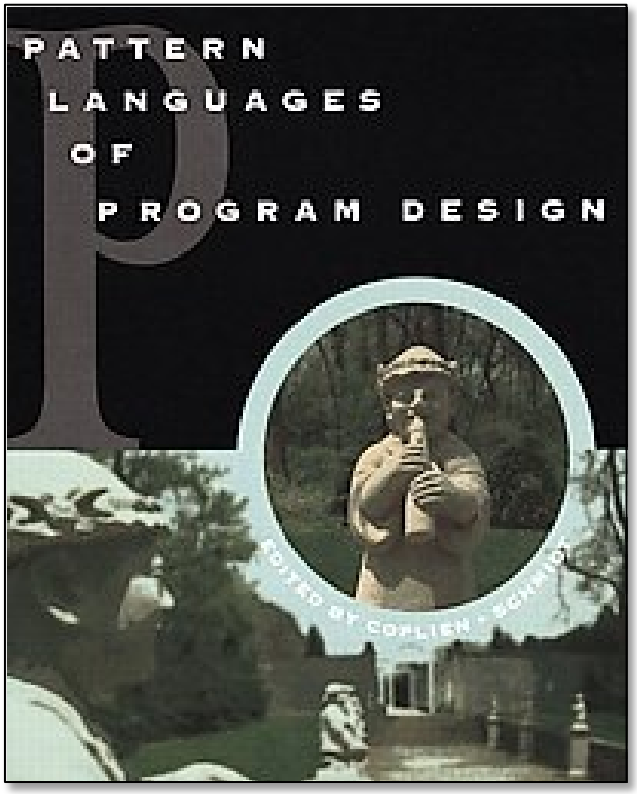
History of the GoF & POSA Pattern Books

- 1991 – Erich Gamma completes his PhD dissertation on patterns for GUIs
- 1992-1993 – Gang-of-Four participate in "Towards an Architecture Handbook" workshop at OOPSLA
- 1993 – GoF publish their first paper at ECOOP
- 1994 – *Design Patterns: Elements of Reusable Object-Oriented Software* (GoF book) published



History of the GoF & POSA Pattern Books

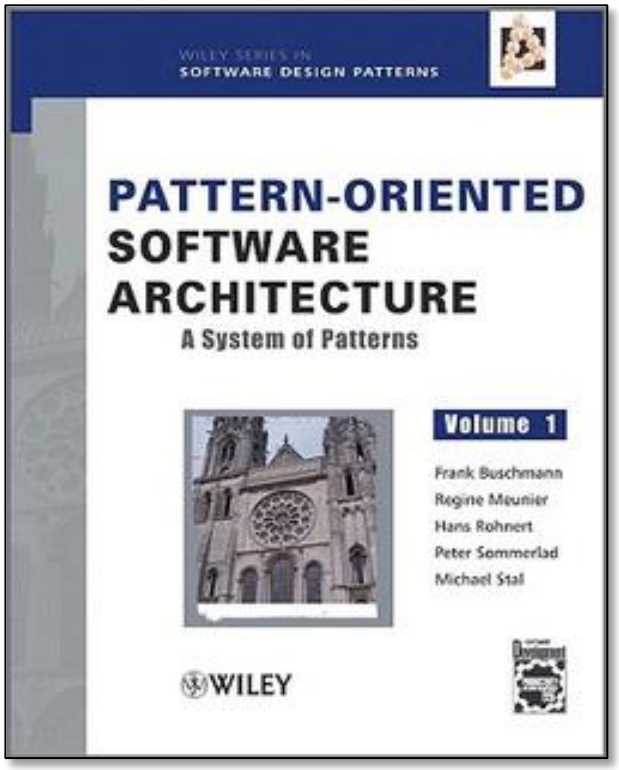
- 1991 – Erich Gamma completes his PhD dissertation on patterns for GUIs
- 1992-1993 – Gang-of-Four participate in "Towards an Architecture Handbook" workshop at OOPSLA
- 1993 – GoF publish their first paper at ECOOP
- 1994 – *Design Patterns: Elements of Reusable Object-Oriented Software* (GoF book) published
- 1994 – First PLoP conference



en.wikipedia.org/wiki/Pattern_Languages_of_Programs has more info on PLoPs

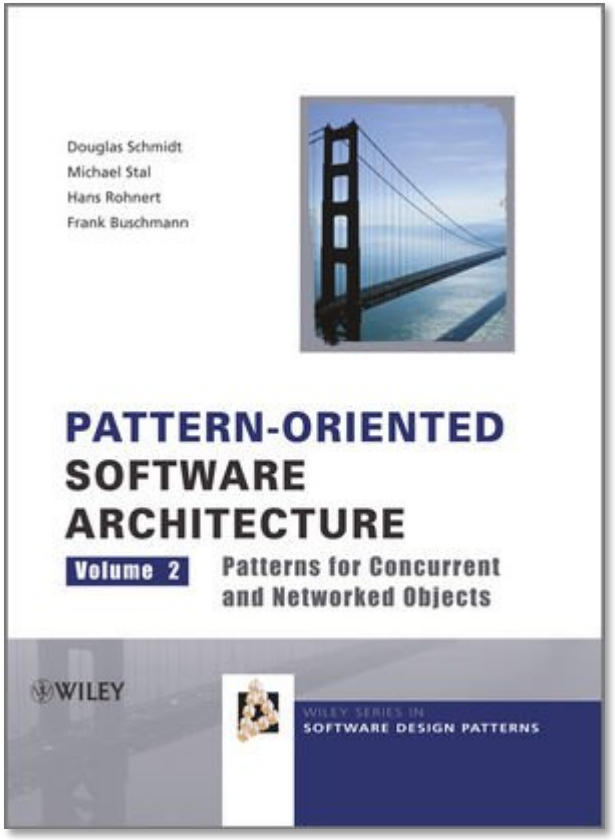
History of the GoF & POSA Pattern Books

- 1991 – Erich Gamma completes his PhD dissertation on patterns for GUIs
- 1992-1993 – Gang-of-Four participate in "Towards an Architecture Handbook" workshop at OOPSLA
- 1993 – GoF publish their first paper at ECOOP
- 1994 – *Design Patterns: Elements of Reusable Object-Oriented Software* (GoF book) published
- 1994 – First PLoP conference
- 1996 – volume 1 of the *Pattern-Oriented Software Architecture* (POSA1 book) published



History of the GoF & POSA Pattern Books

- 1991 – Erich Gamma completes his PhD dissertation on patterns for GUIs
- 1992-1993 – Gang-of-Four participate in "Towards an Architecture Handbook" workshop at OOPSLA
- 1993 – GoF publish their first paper at ECOOP
- 1994 – *Design Patterns: Elements of Reusable Object-Oriented Software* (GoF book) published
- 1994 – First PLoP conference
- 1996 – volume 1 of the *Pattern-Oriented Software Architecture* (POSA1 book) published
- 2000 – volume 2 of the *Pattern-Oriented Software Architecture* (POSA2 book) published



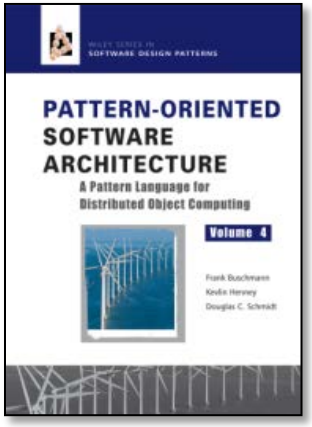
History of the GoF & POSA Pattern Books

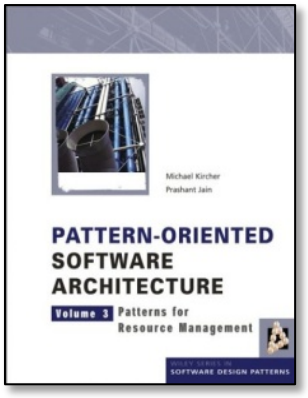
- 1991 – Erich Gamma completes his PhD dissertation on patterns for GUIs
- 1992-1993 – Gang-of-Four participate in "Towards an Architecture Handbook" workshop at OOPSLA
- 1993 – GoF publish their first paper at ECOOP
- 1994 – *Design Patterns: Elements of Reusable Object-Oriented Software* (GoF book) published
- 1994 – First PLoP conference
- 1996 – volume 1 of the *Pattern-Oriented Software Architecture* (POSA1 book) published
- 2000 – volume 2 of the *Pattern-Oriented Software Architecture* (POSA2 book) published
- GoF & POSA authors worked on their books for years
  - They selected design practices that could be recast as patterns, distilled the presentations, culled those deemed immature, etc.

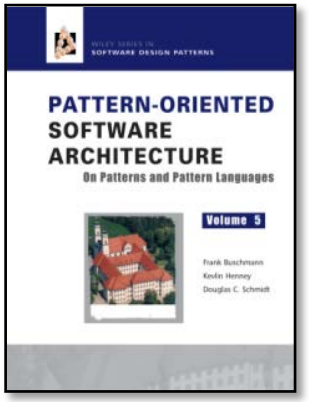
See c2.com/cgi/wiki?HistoryOfPatterns for brief history of patterns

History of the GoF & POSA Pattern Books

- 1991 – Erich Gamma completes his PhD dissertation on patterns for GUIs
- 1992-1993 – Gang-of-Four participate in "Towards an Architecture Handbook" workshop at OOPSLA
- 1993 – GoF publish their first paper at ECOOP
- 1994 – *Design Patterns: Elements of Reusable Object-Oriented Software* (GoF book) published
- 1994 – First PLoP conference
- 1996 – volume 1 of the *Pattern-Oriented Software Architecture* (POSA1 book) published
- 2000 – volume 2 of the *Pattern-Oriented Software Architecture* (POSA2 book) published
- GoF & POSA authors worked on their books for years
- There are several other POSA books, as well





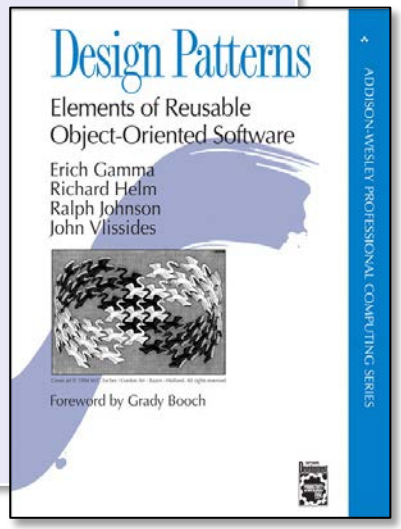


www.dre.vanderbilt.edu/~schmidt/POSA has more info on POSA books

Overview of the Gang of Four Patterns

- The Gof book presents recurring solutions to common problems in software design in the form of 23 patterns

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



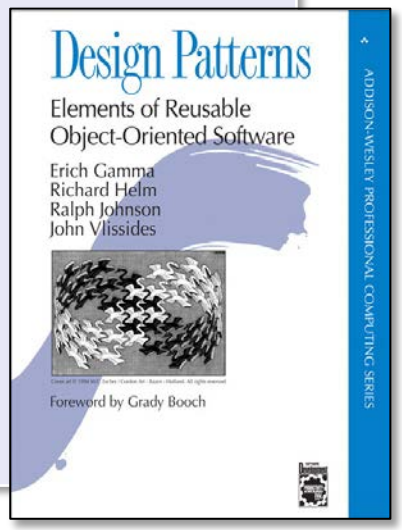
See en.wikipedia.org/wiki/Design_Patterns for more about the GoF book

Overview of the Gang of Four Patterns

Purpose: Reflects What the Pattern Does

Scope: Domain Where Pattern Applies

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



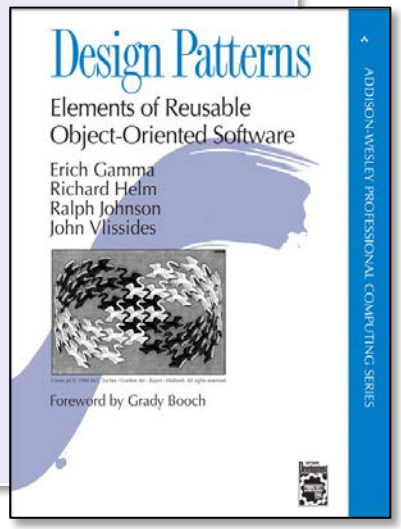
Overview of the Gang of Four Patterns

Abstract the process of instantiating objects

Purpose: Reflects What the Pattern Does

Scope: Domain Where Pattern Applies

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



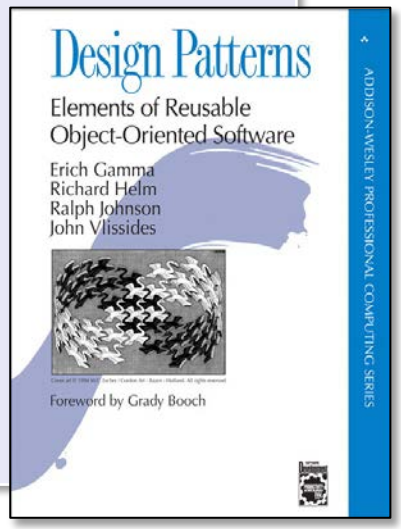
Overview of the Gang of Four Patterns

Describe how classes & objects can be combined to form larger structures

Purpose: Reflects What the Pattern Does

Scope: Domain Where Pattern Applies

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



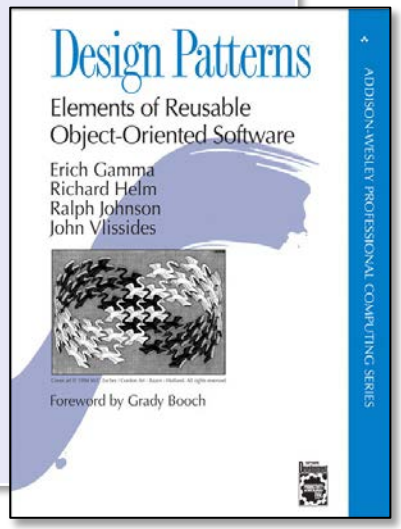
Overview of the Gang of Four Patterns

Concerned with interactions between objects & distribution of responsibility

Purpose: Reflects What the Pattern Does

Scope: Domain Where Pattern Applies

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

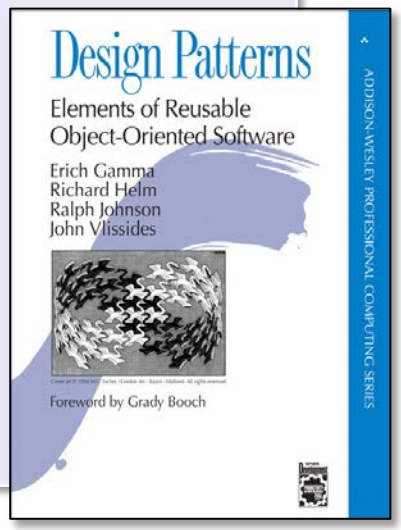


Overview of the Gang of Four Patterns

Purpose: Reflects What the Pattern Does

Scope: Domain Where Pattern Applies

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



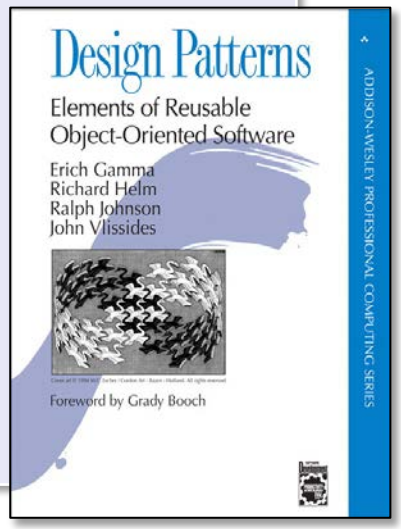
Overview of the Gang of Four Patterns

Class patterns deal with relationships between classes & their subclasses

Purpose: Reflects What the Pattern Does

Scope: Domain Where Pattern Applies

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



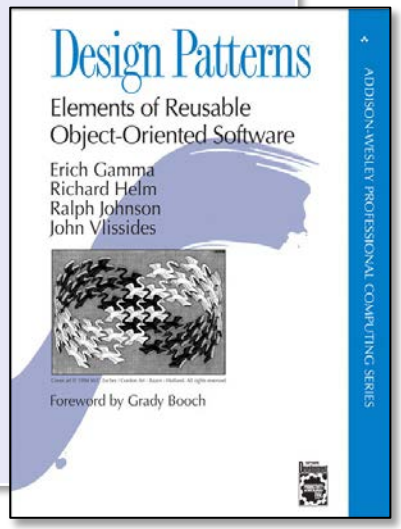
Overview of the Gang of Four Patterns

Object patterns deal with object relationships that can be changed at run-time

Purpose: Reflects What the Pattern Does

Scope: Domain Where Pattern Applies

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



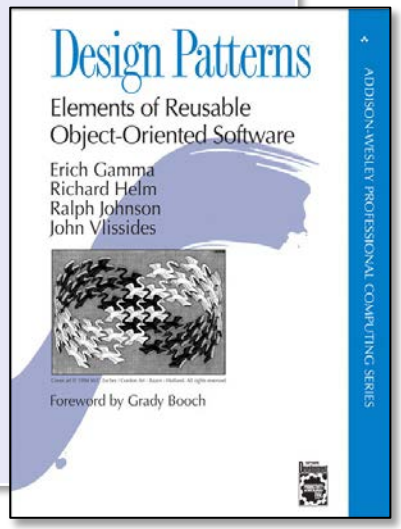
Overview of the Gang of Four Patterns

Although GoF patterns don't focus much on concurrency & networking they are used extensively in Java & Android for a wide range of purposes

Purpose: Reflects What the Pattern Does

Scope: Domain Where Pattern Applies

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



stackoverflow.com/questions/1673841/examples-of-gof-design-patterns

Overview of the POSA1 Patterns

Expresses a fundamental structural organization schema for a software system

Provides a scheme for refining components of a software system or the relationships between them

Architecture Patterns	Design Patterns
Layers	Whole-Part
Pipes & Filters	Master-Slave
Blackboard	Proxy
Broker	Command Processor
Model-View-Controller	View handler
Presentation-Abstraction-Control	Forwarded-Receiver
Microkernel	Client-Dispatcher-Server
Reflection	Publisher-Subscriber

See www.dre.vanderbilt.edu/POSA/POSA1 for more about POSA1

Overview of the POSA1 Patterns

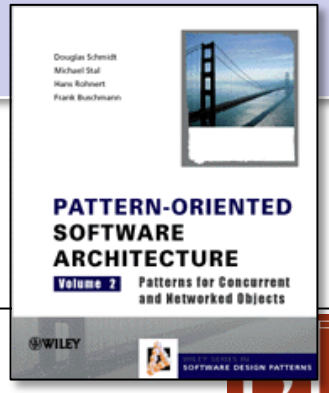
Many POSA1 patterns are relevant for concurrent & networked software in Java & Android

Architecture Patterns	Design Patterns
Layers √	Whole-Part √
Pipes & Filters √	Master-Slave √
Blackboard	Proxy √
Broker √	Command Processor √
Model-View-Controller √	View handler
Presentation-Abstraction-Control	Forwarded-Receiver √
Microkernel	Client-Dispatcher-Server √
Reflection √	Publisher-Subscriber √

Overview of the POSA2 Patterns

Effectively design & configure app access to interfaces & implementations of evolving services & components

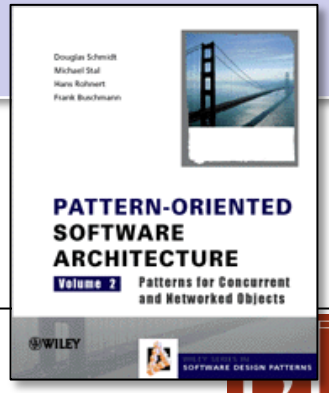
Service Access & Configuration Patterns	Event Handling Patterns	Concurrency Patterns	Synchronization Patterns
Wrapper Facade	Reactor	Active Object	Strategized Locking
Component Configurator	Proactor	Half-Sync/Half-Async	Scoped Locking
Interceptor	Acceptor-Connector	Leader/Followers	Thread-Safe Interface
Extension Interface	Asynchronous Completion Token	Monitor Object	Double-Checked Locking Optimization
		Thread-Specific Storage	



Overview of the POSA2 Patterns

Simplify development of flexible & efficient event-driven apps

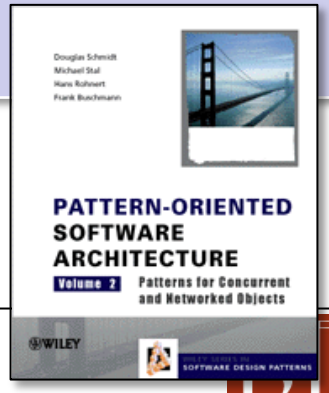
Service Access & Configuration Patterns	Event Handling Patterns	Concurrency Patterns	Synchronization Patterns
Wrapper Facade	Reactor	Active Object	Strategized Locking
Component Configurator	Proactor	Half-Sync/Half-Async	Scoped Locking
Interceptor	Acceptor-Connector	Leader/Followers	Thread-Safe Interface
Extension Interface	Asynchronous Completion Token	Monitor Object	Double-Checked Locking Optimization
		Thread-Specific Storage	



Overview of the POSA2 Patterns

Enhance design & performance of multi-threaded concurrent & networked software

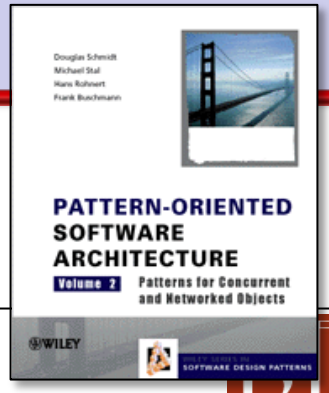
Service Access & Configuration Patterns	Event Handling Patterns	Concurrency Patterns	Synchronization Patterns
Wrapper Facade	Reactor	Active Object	Strategized Locking
Component Configurator	Proactor	Half-Sync/Half-Async	Scoped Locking
Interceptor	Acceptor-Connector	Leader/Followers	Thread-Safe Interface
Extension Interface	Asynchronous Completion Token	Monitor Object	Double-Checked Locking Optimization
		Thread-Specific Storage	



Overview of the POSA2 Patterns

Provide flexible solutions to common problems related to synchronizing concurrent objects

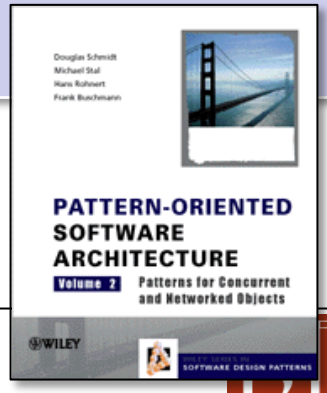
Service Access & Configuration Patterns	Event Handling Patterns	Concurrency Patterns	Synchronization Patterns
Wrapper Facade	Reactor	Active Object	Strategized Locking
Component Configurator	Proactor	Half-Sync/Half-Async	Scoped Locking
Interceptor	Acceptor-Connector	Leader/Followers	Thread-Safe Interface
Extension Interface	Asynchronous Completion Token	Monitor Object	Double-Checked Locking Optimization
		Thread-Specific Storage	



Overview of the POSA2 Patterns

Many POSA2 patterns are relevant for concurrent & networked software in Java & Android

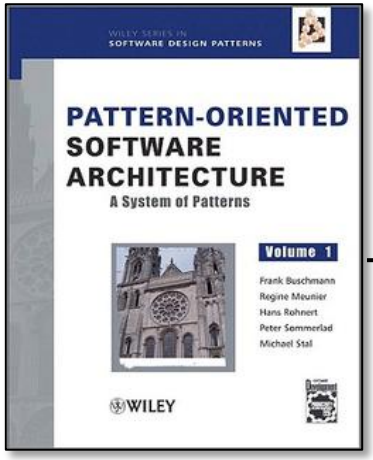
Service Access & Configuration Patterns	Event Handling Patterns	Concurrency Patterns	Synchronization Patterns
Wrapper Facade √	Reactor √	Active Object √	Strategized Locking √
Component Configurator √	Proactor √	Half-Sync/Half-Async √	Scoped Locking √
Interceptor	Acceptor-Connector √	Leader/Followers √	Thread-Safe Interface √
Extension Interface	Asynchronous Completion Token √	Monitor Object √	Double-Checked Locking Optimization √
		Thread-Specific Storage √	

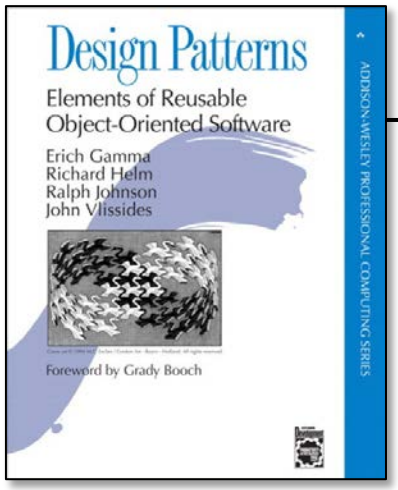


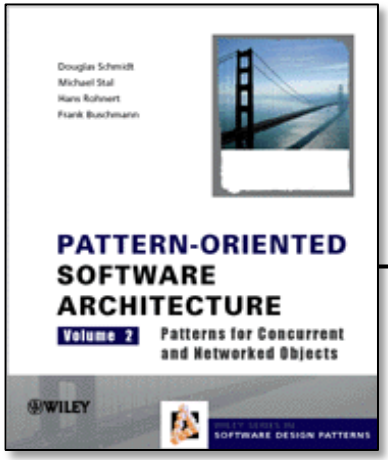
www.dre.vanderbilt.edu/POSA/POSA2 has info on POSA2

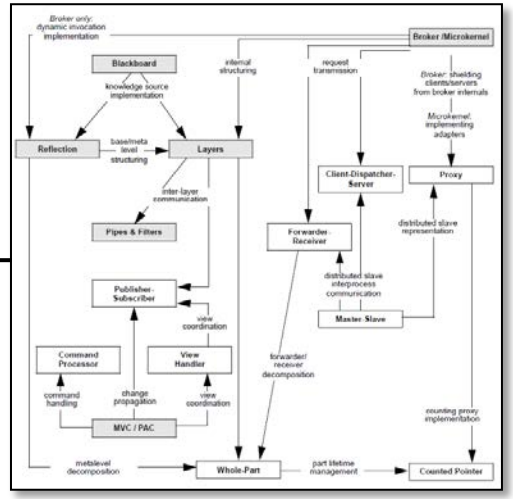
Summary

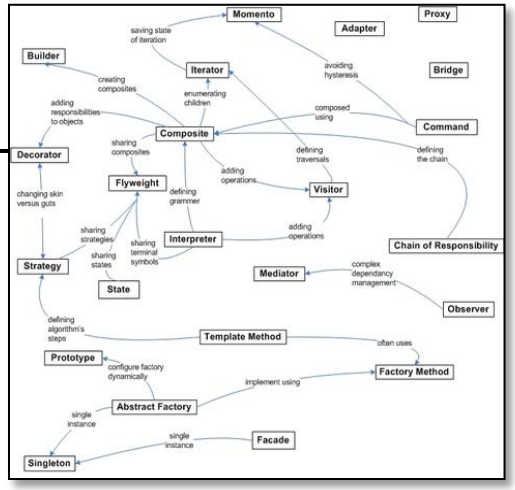
- GoF, POSA1, & POSA2 present patterns in the form of a *pattern collection*
  - Focus is on stand-alone patterns, rather than on pattern relationships

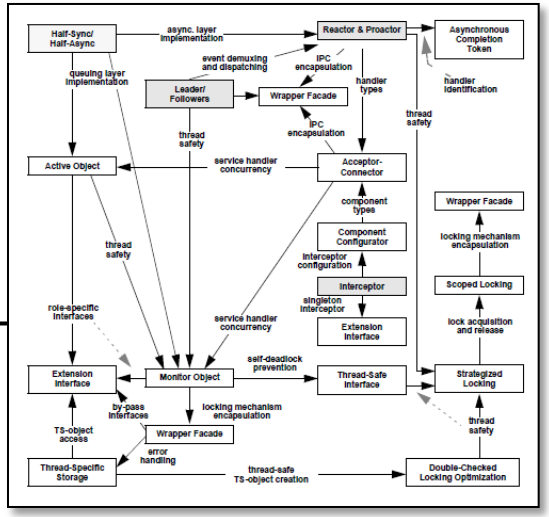










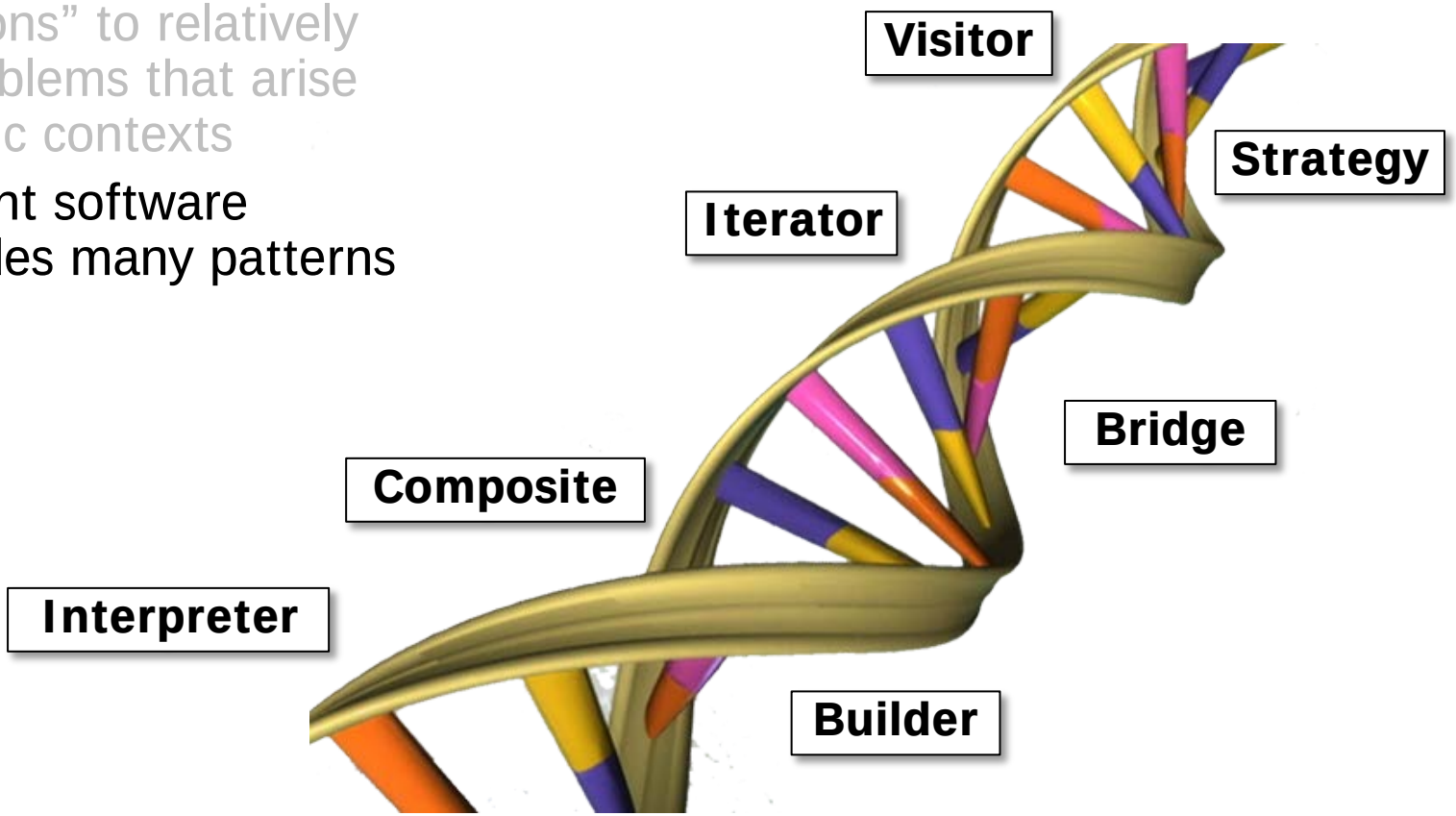


Summary

- GoF, POSA1, & POSA2 present patterns in the form of a *pattern collection*
- Stand-alone patterns provide "point solutions" to relatively bounded problems that arise within specific contexts
  - They play a role similar to vocabulary words in a human language

Summary

- GoF, POSA1, & POSA2 present patterns in the form of a *pattern collection*
- Stand-alone patterns provide "point solutions" to relatively bounded problems that arise within specific contexts
- Any significant software design includes many patterns



en.wikipedia.org/wiki/Pattern_language discusses pattern languages

Summary

- GoF, POSA1, & POSA2 present patterns in the form of a *pattern collection*
- Stand-alone patterns provide "point solutions" to relatively bounded problems that arise within specific contexts
- Any significant software design includes many patterns
- In practice, a stand-alone pattern is unusual

en.wikipedia.org/wiki/Pattern_language discusses pattern languages