An Introduction to Patterns and Pattern Languages

CSC591O
April 7-9, 1997
Raleigh, NC

Patterns -- Why?

- Learning software development is hard
  - Lots of new concepts
  - Hard to distinguish good ideas from bad ones
- Languages and frameworks are very complex
  - Too much to explain
  - Much of their structure is incidental to our problem

Overview

- Patterns
- Software Patterns
- Design Patterns
- Architectural Patterns
- Pattern Catalogs
- Pattern Languages

Patterns -- Why?

- Must be some way to communicate better
  - Allow us to concentrate on the problem
- Patterns can provide the answer
Patterns -- What?

- What is a pattern?
  - A solution to a problem in a context
  - A structured way of representing design information in prose and diagrams
  - A way of communicating design information from an expert to a novice
  - Generative: show when and how to apply solutions
  - One of the hottest topics in OO Design

Patterns -- Parts

- Patterns are made up of four main parts
  - Title -- the name of the pattern
  - Problem -- a statement of what the pattern solves
  - Context -- a discussion of the constraints and forces on the problem
  - Solution -- a description of how to solve the problem
    - This may include a list of the participants in the pattern
  - May have other sections

Patterns -- Origin (1)

- Pattern concept came from architecture
  - A Pattern Language [Alexander77]
  - A Timeless Way of Building [Alexander79]
- Alexander used patterns to
  - Express the interaction of forces in a problem
  - Arrive at an elegant solution

Patterns -- Origin (2)

- A Pattern Language describes
  - Common architectural motifs
  - How they can come together to form a cohesive, liveable environment

Patterns -- Doors Example (1)

- Corner Doors
  - [96 in Alexander77]
- Problem: How do you place the doors in a room?
- “If the doors create a pattern of movement which destroys the places in a room, the room will never allow people to be comfortable.”

Patterns -- Doors Example (2)

- Solution: “…in most rooms, especially small ones, put the doors as near the corners of the room as possible.”
- “If a room has two doors, and people move through it, keep both doors at one end of the room.”
Software Patterns -- What?

- Ward Cunningham and Kent Beck from Tektronix
- In 1987, they applied Alexander’s ideas to designing user interfaces
  - “Nouns in Lists, Verbs in Menus”
  - “Window per Task”
- The idea was independently discovered
  - Jim Coplien of AT&T [Coplien92]
  - Others

Software Patterns -- Why?

- Abstractors and Elaborators [Beck94]
  - Abstractors create reusable pieces (frameworks)
  - Elaborators use pieces to build applications
  - Abstractors must explain pieces to elaborators
  - Patterns are an efficient explanation of intent

Software Pattern -- GUIs

- Nouns in Lists, Verbs in Menus
- Problem:
  - Designing a GUI: What goes in lists vs. menus?

Nouns in lists...

- Context:
  - Principle of least astonishment
  - People expect to perform actions on subjects
  - List of actions without subjects looks incomplete
  - Select a subject first, then an action
Patterns have exploded!

Architectural Patterns

- Deal with the highest levels of software development
- Often discovered in very large projects
- A few published examples
  - Coad, Object Models: Strategies Patterns and Applications
  - Buschmann, Pattern Oriented Software Architectures: A System of Patterns

Architectural Pattern - Pipes and Filters

- From [Buschmann96]
- Problem: Building Systems to Transform and Process Streams of Data
  - Different steps of the process may be represented differently
  - Many intermediate data forms

Pipes and Filters (2)

- Solution: Divide the task into sequential processing steps (filters).
- Steps are connected by data flow components (pipes)
- Example: Compiler

Architectural Pattern -- Four Layer Architecture (1)

- Problem:
  - How do you structure an application with significant presentation and model components?
- Forces:
  - Existing structures
  - Code reuse
  - Distribution of work among team members
  - Portability to new environments and language vendors

Four Layer Architecture (2)

- Solution:
  - Divide classes into layers
  - The MVC architecture of Smalltalk-80 separated the "model" and "view" components
  - More layers are needed to model all necessary abstractions
### Smalltalk Application Layers

<table>
<thead>
<tr>
<th>View layer</th>
<th>Application layer</th>
<th>Domain layer</th>
<th>Infrastructure layer</th>
</tr>
</thead>
</table>

### Layer purposes
- **View layer** - provides the graphical view of an application. Handles user interaction
- **Application layer** - Mediates between different views of an application. Adapts the protocol of the View layer to the protocol of the domain model.
- **Domain layer** - Represents the business or “real-world” model of the application
- **Infrastructure layer** - supports the domain layer with connections to external interfaces

### Analysis Patterns
- Patterns about Analysis
- Are they patterns about the Analysis process, or patterns found in systems analysis?
- Both have been published...

### Analysis Patterns
- Written by Martin Fowler
- Deals with common business domains
  - Accounting
  - Finance
  - Inventory
- Includes general purpose analysis patterns

### An Analysis Pattern
- **Account**
- Problem: In many fields you must keep records of not just the current value of a thing, but the changes that have affected that value.

### Account - Solution
- Basic Double-Entry Accounting!
Design Patterns

- Design Patterns are patterns about OO design
- Discuss tradeoffs inherent in the process
- Evaluate alternative designs

Design Patterns -- Gang of Four

- Gang of Four is Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides
- Collaborated on finding patterns in software over a series of years
- Finally decided to write a book

Design Patterns

- Their book is *Design Patterns: Elements of Reusable Object-Oriented Software* [GHJV95]
- 23 basic design patterns found in many large OO systems
- A pattern catalog
  - Patterns are not dependent on each other

A Design Pattern -- Strategy (315)

- Often you have the "Choose 1 of N algorithms" problem.
- Example: Converting different file formats — GIF, JPEG, EPS, etc.
- You could write one "FileReader" with a LOT of case statements.

Strategy Solution

- Define a family of algorithms
  - Various approaches to solving a problem
  - Encapsulate them as objects
  - Make them polymorphic
- Client requests problem be solved
- Strategy directs how problem is solved

Strategy Structure

```
Client --> strategy --> AbstractStrategy
          |                |
          v                v
StrategyA  StrategyB
```
Example Strategy

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Conversion Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIFReader</td>
<td>readFileNamed</td>
</tr>
<tr>
<td>JPGReader</td>
<td>readFileNamed</td>
</tr>
<tr>
<td>AbstractReader</td>
<td>readFileNamed</td>
</tr>
</tbody>
</table>

Strategy Benefits

- Avoids case statements
- Easily extensible
  - just add new Strategy subclasses
- Strategy classes may be reusable elsewhere

A Design Pattern -- State (305)

- Sometimes you have an object that acts differently over time.
- Example: A TCPConnection
  - You can't open an opened one or close a closed one.
- You could use methods like become: or changeClassToThatOf:
  - leads to messy, difficult code

State Solution

- Define a set of State objects.
- Defers the implementation of the different methods to the state object.
- The client changes its state from time to time.

TCP Example STD

TCP Example Hierarchy
### State Benefits

- Makes state transitions explicit
- Localizes behavior; better allocation of responsibility
- Eliminates the need for case statements
- Since the States are objects, the State Machine can be reconfigured on the fly

### Why use Design Patterns

- Proven solutions to common OO problems
- Confer expertise to beginners
- Common vocabulary for designers
- Documentation for existing frameworks
- Once internalized, good design techniques become second nature
- Often combined to create more sophisticated designs

### How to use Design Patterns

- Usually the pattern describes a role for an object -- not a class!
  - Don’t blindly implement all 23 as classes!
- Scout around the intents to find a close match
- Page through the diagrams

### Languages-Specific Patterns

- Often called idioms
- Can be found in any language (not just OO)
- They are the way experts do things
  - take advantage of language features
  - exploit knowledge of language and library internals

### Smalltalk Idioms

- Many found in *Smalltalk Best Practice Patterns*
- Written by Kent Beck
- Contains Smalltalk idioms, programming practice patterns, and many others

### C++ Idioms

- Many found in *Advanced C++ Programming Styles and Idioms*
- Written by Jim Coplien
- Contains a lot of C++ idioms (including some very good Memory Mgt. ones)
An example Smalltalk Idiom

- Problem: How do you implement a Stack?
- Solution: Simulate a Stack using OrderedCollection

<table>
<thead>
<tr>
<th>Stack Operation</th>
<th>OrderedCollection message</th>
</tr>
</thead>
<tbody>
<tr>
<td>push</td>
<td>addLast:</td>
</tr>
<tr>
<td>pop</td>
<td>removeLast:</td>
</tr>
<tr>
<td>top</td>
<td>last</td>
</tr>
<tr>
<td>empty</td>
<td>isEmpty</td>
</tr>
</tbody>
</table>

Organizational Patterns

- Patterns about forming, managing and understanding organizations and their behavior
- Often tied to software more closely than you think...

Organizational Patterns

- Generative Development-Process [Coplien95]
  - 42 patterns discuss team members, team organization, and project management
  - Pattern 14: Conway’s Law
    - organization determines architecture and vice versa
  - Pattern 15: Architect also implements
    - keep the roles synchronized
  - Pattern 32: Divide and Conquer
    - cluster roles that collaborate strongly, form separate organizations

Pattern Languages -- What?

- What is a pattern language?
- A collection of related patterns
- Each pattern leads to others
- Combine to solve an entire domain of problems
- Multiple solutions from multiple paths

Pattern Languages -- Origin

- Christopher Alexander (again)
  - A Pattern Language [Alexander77]
  - A Timeless Way of Building [Alexander79]
- The Hillside Group [Beck94]
  - Promotes patterns and pattern languages for software
  - Started the PLoP (Pattern Languages of Programming) conferences
  - Edited and published the PLoP books [PLoP95 and PLoP96]
Pattern Language -- Self-Encapsulation

- Self-Encapsulation [Auer95]
  - Define Classes by Behavior, Not State
  - Implement Behavior with Abstract State
  - Identify Message Layers
  - Defer Identification of State Variables
  - Encapsulate Concrete State
  - Use Lazy Initialization
  - Define Default Values via Explicit Protocol
- Explains how to develop an extensible hierarchy

Pattern Language -- Crossing Chasms [Brown96]

Review (1)

- Patterns
  - Solutions to common problems
  - Explain the assumptions behind the solution
  - Define names and scope for common techniques
  - Show when and how to apply techniques
  - Document expertise

Review (2)

- Software Patterns
  - Patterns for software development
  - Architecture, design, implementation
  - Reuse of expertise
- Architectural Patterns
  - Recurring architectural structures
- Analysis Patterns
  - Recurring object models

Review (3)

- Design Patterns
  - Recurring OO design structures and techniques
  - Commonly found in successful frameworks
- Idioms
  - Recurring code structures

Review (4)

- Pattern Languages
  - Series of closely related patterns
  - Each pattern builds on others
  - Represents a series of decisions
- Pattern Catalogs
  - A collection of loosely related patterns
  - Each pattern stands on its own
Further Reading (1)


Further Reading (2)


Further Reading (3)


Further Reading (4)


Further Reading (5)


Further Reading (6)