Erich Gamma

Design Patterns
Object-Oriented Programming Languages

- provide powerful construction elements for building systems
- but... designing good object-oriented systems is hard – many degrees of freedom
- anybody can make objects that work challenges: flexibility, no duplication, ...
  ⇒ how to reuse design expertise?
Object-Oriented Frameworks

- Lisa Toolkit (84) and MacApp (87) illustrate how to reuse both design and code.
- A framework is a customizable set of cooperating classes that defines a reusable solution for a given problem.
ET++

- a comprehensive C++ class library and framework (87)

⇒ there are idiomatic class an object structures that makes designs more flexible and reusable

⇒ same structures can be observed in other frameworks
Problems with Frameworks - Taligent

• Framework development is challenging
• frameworks often work on the supplier side only!
• frameworks apply for a specific domain only
• *Greyhound principle* “Leave the driving to us”
Conceptual Integrity of a System

- conceptual integrity
  Brooks 75
- architecture example
  - cathedral of Reims
  - architectural unity
  - 8 generation of builders

⇒ micro-architectures that contribute to the overall architecture
Engineer Handbooks

Slider Crank Mechanism of a Centrifugal Governor

Link 1 is designed as a Round plate turning About axis A. The angle $\alpha$ Between the plane and The axis of rotation of shaft 2 depends upon the Centrifugal force. When Angle $\alpha$ is changed, connecting rod 3 slides sleeve 4 along the axis of shaft 2.

From: "Mechanisms of Modern Engineering Design": Ivan Artobolevsky 1947
Alexander’s Pattern Languages

- Kent Beck and Ward Cunnigham got interested in Alexander’s work (87)
- what gives a building its quality?
  - freedom, durability, adaptability
- a pattern is a solution to a problem in a context

“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.”

- pattern language with 253 patterns
A Design Pattern...

• describes a recurring design structure
  - names and abstracts from concrete designs
  - identifies objects and their collaborations
  - discusses trade-offs, implementation variations

⇒ pattern idea isn’t new, the contribution is the codification
Useful, Used, Useable

• useful - patterns enable to repeat a successful design done by someone else
  - non-obvious solution
• used - recurring phenomenon
  - reverse architecting instead of inventing
    ⇒ rule of three
• usable - include coding details, enable many variations
Frameworks and Patterns

- Design patterns fill the gap between class libraries and frameworks
  - smaller architectural elements than frameworks
  - design patterns are more abstract than frameworks

increasing reuse
Towards a Design Patterns Catalog

- first dozen of patterns reverse architected from ET++ (91)
- Jim Coplien’s catalog of C++ idioms (91)
- GOF (John Vlissides, Ralph Johnson, Richard Helm, Erich Gamma) get together between (90-92)
  - Smalltalk, InterViews, ET++
- first version of catalog
  ⇒ only experts understand it:
    “oh, I’ve done that”
Towards a Patterns Catalog (Cont’d)

- **finding** patterns is much easier than **describing** so that others can understand them
  - discussed negative consequences
  - added motivating example
  - added sample code
  ⇒ 3 rewrites
  ⇒ 2000+ emails
Design Pattern Examples

- **Observer**: managing dependencies between objects
- **Strategy**: algorithms as objects
- **Composite**: recursive structures
# GOF Pattern Catalog

<table>
<thead>
<tr>
<th><strong>creational</strong></th>
<th><strong>structural</strong></th>
<th><strong>behavioral</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Factory</td>
<td>Adapter</td>
<td>Chain of Resp.</td>
</tr>
<tr>
<td>Prototype</td>
<td>Bridge</td>
<td>Command</td>
</tr>
<tr>
<td>Singleton</td>
<td>Composite</td>
<td>Iterator</td>
</tr>
<tr>
<td>Factory Method</td>
<td>Decorator</td>
<td>Mediator</td>
</tr>
<tr>
<td>Builder</td>
<td>Flyweight</td>
<td>Memento</td>
</tr>
<tr>
<td></td>
<td>Facade</td>
<td>Observer</td>
</tr>
<tr>
<td></td>
<td>Proxy</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Visitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Template Method</td>
</tr>
</tbody>
</table>

- **creating objects**
- **composing objects**
- **interacting objects**
Growth – Pattern Community

- pattern mining - writing patterns
  - non-anonymous reviewing ⇒ shepherding
  - writer’s workshops
  - pattern discussion groups
- conferences
  - PLoP 94-, EuroPLoP 95-
  - theory and practice met
- more pattern catalogs
Patterns at Work

- patterns must be
  - instantiated and tweaked
  - combined with other patterns
- act as sign posts during design
  - help understand trade-offs
  - help evaluate alternatives
Describing JUnit with Patterns

**Command**

- **TestDecorator**
  - run()

**Template Method**

- **TestCase**
  - run(TestResult result)
  - setUp()
  - tearDown()

- **Test**
  - run(TestResult result)

- **TestSuite**
  - run(TestResult result)

**Decorator**

- **TestDecorator**
  - run()

**Composite: Component**

- **Composite**
  - tests

**Observer**

- **TestListener**
  - testFailed()
Observations

- design patterns provide words for telling a story about a design
- large portion of design covered with patterns
- patterns explain relationships among objects
- objects participate in more than one pattern ⇒ density
Patterns Today

- patterns applied in wide variety of domains
  - analysis, architectural, coding, organizational
- design notations/methods and tools have added support for patterns
- many new published patterns
  - there is a need for a consolidation
- patterns show up in APIs of new class libraries
Java – Swing UI

- **JTable**: setModel(), addActionListener(), getUIClassID()
- **TableModel**: getElementAt(), getSize()
- **JTableUI**: paint(), getPreferredSize(), mousePressed()
- **TableCellRenderer**: getComponent()
- **Observer**: Subject
- **Strategy**: Context
- **Flyweight**: getComponent()
- **Strategy**: getComponent()
Confessions

- we have never told people when to remove a pattern from a design
- patterns over-enthusiasm has resulted in over-engineered designs
  - simplicity is the most important architectural quality
- not all patterns are “goodness”
Looking Back

- design patterns were a success because of their direct applicability
- patterns are tools and not rules
- enabling design communication is the major contribution of design patterns
The Show Trial of the Gang of Four for Crimes against Computer Science (OOPSLA 99)

at OOPSLA John Vlissides, Ralph Johnson, Richard Helm, and Erich Gamma, were brought to justice for crimes against the field. Charges included their cultivation of a cult of personality, training novices to behave like experts, and exhibiting an utter disregard for traditional standards of academic originality.