Design Patterns

On Design Patterns

- A **design pattern** systematically names, explains and evaluates an important and recurring design problem and its solution
- Good designers know not to solve every problem from first principles

  They reuse solutions

- This is very different from code reuse
- Software practitioners have not done a good job of recording experience in software design for others to use

Classification

- Structural

  Decouple interface and implementation of classes and objects

  Adapter, Composite, Decorator, Facade

- Behavioural

  Dynamic interaction among classes and objects

  Command, Iterator, Master-Slave, State, Visitor

- Creational

  Initializing and configuring classes and objects

  Abstract Factory, Builder, Prototype, Singleton

Acknowledgement

Descriptions of many patterns
based on
Design Patterns
by
Erich Gamma, Richard Helm
Ralph Johnson, John Vlissides

Addison-Wesley, 1995.
ISBN 0-201-63361-2

Descriptive Template

- Name
- Intent

  What does the pattern do? What problems does it address?
- Motivation

  A scenario of pattern applicability
- Applicability

  In which situations can this pattern be applied
- Participants

  Describe participating classes/objects

Descriptive Template – 2

- Scenario – Collaborations
  How do the participants carry out their responsibilities?
- Architecture
  Graphical representation of the pattern
- Consequences
  How does the pattern support its objectives?
- Implementation
  Pitfalls, language specific issues
- Examples
  From real systems
- See also
  Pointers to related patterns

Design Patterns – Definition

"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, collaborations and distribution of responsibilities."

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides ,"*Design Patterns*", Addison-Wesley, 1995. ISBN 0-201-63361-2

Others On Design Patterns

- Christopher Alexander

  "Each person describes a problem which occurs over and 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."

- Cunningham

  "Patterns are the recurring solutions to the problem of design. People learn patterns by seeing them and recall them when need be without a lot of effort"

Others On Design Patterns – 2

- Booch

  "A pattern is a solution to a problem in a specific context. A pattern codifies specific knowledge collected from experience in a domain."

Patterns & Frameworks

- Patterns support reuse of software architecture and design

  They capture static and dynamic structures of successful solutions to problems. These problems arise when building applications in a particular domain

- Frameworks support reuse of detailed design and program source text

  A framework is an integrated set of components that collaborate to provide a reusable architecture for a family of related applications

Patterns & Frameworks – 2

- Frameworks tend to be less abstract than patterns
- Together, design patterns and frameworks help to improve key quality factors like reusability, extensibility and modularity

Becoming a Master Designer

- Learn the rules
  - » **algorithms and data structures**
  - » **languages**
  - » **mathematics**
- Learn the principles
  - » **structured and modular programming**
  - » **theory of software engineering**
  - » **OO design and programming**
- Study the designs of masters
  - » **Design patterns must be understood, memorized and applied**
  - » **Thousands of existing patterns**

Are they all memorable?

Design Patterns Solve Design Problems

- Finding appropriate classes
- Determine class granularity

  How abstract, how correct

- Specify interfaces
- Specify implementation
- Put reuse to work

  Client vs inheritance

- Relate run time and compile time structures

  Program text may not reflect design

Design Patterns Solve Design Problems – 2

- Design for change is difficult
- Common problems
  - » **Explicit object creation**
    - **Use name of interface, not name of implementation**
  - » **Dependence of particular operations**
    - **Avoid hard coded operations**
  - » **Dependencies on hardware or software platforms**
  - » **Dependencies of object representation**
  - » **Dependencies on algorithms**
  - » **Tight coupling**

Claims of the Pattern Community

- Well defined design principles have a positive impact on software engineering
  - » **Achievable reusability**
  - » **Provide common vocabulary for designers**

    Communicate, document, explore alternatives
  - » **Patterns are like micro architectures**

    Useful for building small parts of a system
  - » **Reduce the learning time for understanding class libraries**
  - » **Avoid redesign stages by using encapsulated experience**

When to Use Patterns

- Solutions to problems that recur with variations
  - » **No need for pattern if the problem occurs in only one context**
  - » **Can we generalize the problem instance in which we are interested?**
- Solutions that require several steps
  - » **Not all problems need all steps**
  - » **Patterns can be overkill if solution is a simple linear set of interactions**
- Solutions where the solver is more interested in “does there exist a solution?” than in a solution’s complete derivation

  Patterns often leave out lots of detail

Key Principles

- Successful use of patterns and frameworks can be boiled down to a few key principles
  - » **Separate interface from implementation so each can vary independently**
  - » **Determine what is common and what is variable with an interface and an implementation**
  - » **Allow substitution of variable implementation via a common interface. Use deferred classes and effect them**
- Don't use blindly

  Separating commonalties from variabilities should be done on a goal by goal basis not exhaustively

  It isn't always worthwhile to apply them

Pattern Benefits

- Enable large scale reuse of software architectures
- Explicitly capture expert knowledge and design trade-offs
- Help improve developer communication
- Help ease the the transition to OO methods
- High level abstraction that leaves out the details

Pattern Drawbacks

- Patterns do not lead to direct code reuse
- Patterns are often deceptively simple
- You may suffer from pattern overload
- Patterns must be validated by experience and debate rather than automated testing
- Integrating patterns into a process is human intensive rather than a technical activity