CSC532 Term Paper

DESIGN PATTERNS

Shirin A. Lakhani
Louisiana Tech University

Design Patterns

Shirin A. Lakhani
Louisiana Tech University

Abstract

This paper is an introduction to Design Patterns. Patterns are recent software engineering problem solving discipline that emerged from object oriented community. The primary purpose of pattern is communicating design insights and making patterns coherent and easier to understand.

1. Introduction

Design Patterns are rational reconstruction of existing programming practice. Patterns help create a shared language for communicating insight and experience about problems and their solutions. Formally codifying solutions and their relationships lets us successfully capture the body of knowledge which defines our understanding of good architectures that meet the needs of their users. In software engineering, design patterns are ideas that have been useful in one practical context of software design and will probably be useful in others. They provide a firm and stable engineering backbone around which the rest of the system's architecture should be built. The goal of patterns within the software community is to create a body of literature to help software developers resolve recurring problems encountered throughout all of software development.

Gamma et al (1994) describes a design pattern as a proven solution for a general design problem. It consists of communicating classes and objects that are customized to solve the problem in a particular 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. -- *Christopher Alexander*

2. Elements of Design Pattern

The following essential elements should be clearly recognizable upon reading a pattern*:*

a) **Name**: It must have a meaningful name. This allows us to use a single word or short phrase to refer to the pattern, and the knowledge and structure it describes. It would be very unwieldy to have to describe or even summarize the pattern every time we used it in a discussion. Good pattern names form a vocabulary for discussing conceptual abstractions. Sometimes a pattern may have more than one commonly used or recognizable name in the literature. In this case it is common practice to document these nicknames or synonyms under the heading of Aliases or Also Known As. Some pattern forms also provide a classification of the pattern in addition to its name.

b) **Problem**: A statement of the problem which describes its intent: the goals and objectives it wants to reach within the given context and forces. Often the forces oppose these objectives as well as each other.

c) **Context:** The preconditions under which the problem and its solution seem to recur, and for which the solution is desirable. This tells us the pattern's applicability. It can be thought of as the initial configuration of the system before the pattern is applied to it.

d) **Forces**: A description of the relevant forces and constraints and how they interact/conflict with one another and with goals we wish to achieve (perhaps with some indication of their priorities). Forces reveal the intricacies of a problem and define the kinds of trade-offs that must be considered in the presence of the tension or dissonance they create. A good pattern description should fully encapsulate all the forces which have an impact upon it.

e) **Solution**: Static relationships and dynamic rules describing how to realize the desired outcome. This is often equivalent to giving instructions which describe how to construct the necessary work products. The description may encompass pictures, diagrams and prose which identify the pattern's structure, its participants, and their collaborations, to show how the problem is solved. The solution should describe not only static structure but also dynamic behavior. The static structure tells us the form and organization of the pattern, but often it is the behavioral dynamics that make the pattern "come alive". The description of the pattern's solution may indicate guidelines to keep in mind (as well as pitfalls to avoid) when attempting a concrete implementation of the solution. Sometimes possible variants or specializations of the solution are also described.

f) **Examples**: One or more sample applications of the pattern which illustrate: a specific initial context; how the pattern is applied to, and transforms, that context; and the resulting context left in its wake. Examples help the reader understand the pattern's use and applicability. Visual examples and analogies can often be especially illuminating. An example may be supplemented by a sample implementation to show one way the solution might be realized. Easy-to-comprehend examples from known systems are usually preferred.

g) **Resulting Context**: The state or configuration of the system after the pattern has been applied, including the consequences (both good and bad) of applying the pattern, and other problems and patterns that may arise from the new context. It describes the postconditions and side-effects of the pattern. This is sometimes called resolution of forces because it describes which forces have been resolved, which ones remain unresolved, and which patterns may now be applicable. Documenting the resulting context produced by one pattern helps you correlate it with the initial context of other patterns (a single pattern is often just one step towards accomplishing some larger task or project).

h) **Rationale**: A justifying explanation of steps or rules in the pattern, and also of the pattern as a whole in terms of how and why it resolves its forces in a particular way to be in alignment with desired goals, principles, and philosophies. It explains how the forces and constraints are orchestrated in concert to achieve a resonant harmony. This tells us how the pattern actually works, why it works, and why it is "good". The solution component of a pattern may describe the outwardly visible structure and behavior of the pattern, but the rationale is what provides insight into the deep structures and key mechanisms that are going on beneath the surface of the system.

i) **Related Patterns**: The static and dynamic relationships between this pattern and others within the same pattern language or system. Related patterns often share common forces. They also frequently have an initial or resulting context that is compatible with the resulting or initial context of another pattern. Such patterns might be predecessor patterns whose application leads to this pattern; successor patterns whose application follows from this pattern; alternative patterns that describe a different solution to the same problem but under different forces and constraints; and codependent patterns that may (or must) be applied simultaneously with this pattern.

j) **Known Uses**: Describes known occurrences of the pattern and its application within existing systems. This helps validate a pattern by verifying that it is indeed a proven solution to a recurring problem. Known uses of the pattern can often serve as instructional examples.

Although it is not strictly required, good patterns often begin with an **Abstract** that provides a short summary or overview. This gives readers a clear picture of the pattern and quickly informs them of its relevance to any problems they may wish to solve (sometimes such a description is called a thumbnail sketch of the pattern, or a **pattern thumbnail**). A pattern should identify its target audience and make clear what it assumes of the reader.

3. Criteria for Design Patterns

The PLoP conferences have several criteria which they feel submitted pattern papers should meet. These are as follows:

a) **Focus on practicability:** Patterns should describe proven solutions to recurring problems rather than the latest scientific results.
b) **Aggressive disregard of originality:** Pattern writers do not need to be the original inventor or discoverer of the solutions that they document.
c) **Non-anonymous review:** Pattern submissions are shepherded rather than reviewed. The shepherd contacts the pattern author(s) and discusses with them how the patterns might be clarified or improved upon.
d) **Writer's workshops instead of presentations:** Rather than being presented by the individual authors, the patterns are discussed in **writer's workshops**: an open forum where all attending seek to improve the patterns presented by discussing what they like about the patterns as well as other areas in which they are lacking.
e) **Careful editing:** The pattern authors have the opportunity to incorporate all the comments and insights during the shepherding and writer's workshops before presenting the patterns in their finished form.

4. Qualities of Pattern

a) **Encapsulation and Abstraction:** Each pattern encapsulates a well-defined problem and its solution in a particular domain. Patterns should provide crisp, clear boundaries that help crystallize the problem space and the solution space by parceling them into a lattice of distinct, interconnected fragments. They also serve as abstractions which embody domain knowledge and experience, and may occur at varying hierarchical levels of conceptual granularity within the domain.

b) **Openness and Variability:** Each pattern should be open for extension or parameterization by other patterns so that they may work together to solve a larger problem. A pattern solution should be also capable of being realized by an infinite variety of implementations (in isolation, as well as in conjunction with other patterns).

c) **Generativity and Composability:** Each pattern, once applied, generates a resulting context which matches the initial context of one or more other patterns in a pattern language. These subsequent patterns may then be applied to progress further toward the final goal of generating a "whole" or complete overall solution. Applying one pattern provides a context for the application of the next pattern." (from the PLoP'97 Call for Papers) But patterns are not simply linear in nature, more like fractals in that patterns at a particular level of abstraction and

granularity may each lead to or be composed with other patterns at varying levels of scale.

d) Equilibrium: Each pattern must realize some kind of balance among its forces and constraints. This may be due to one or more invariants or heuristics that are used to minimize conflict within the solution space. The invariants often typify an underlying problem solving principle or philosophy for the particular domain, and provide a rationale for each step/rule in the pattern.

The aim is that, if well written, each pattern describes a whole that is greater than the sum of its parts, due to skillful choreography of its elements working together to satisfy all its varying demands.

5. Why Design Patterns?

Software Patterns help us because they:

a) Solve "real world" problems
b) Capture domain expertise
c) Document design decisions and rationale
d) Reuse wisdom and experience of master practitioners
e) Convey expert insight to novices
f) Form a shared vocabulary for problem-solving discussion
g) Record experience in designing object oriented software as design patterns.
h) Capture experience in the form of a catalog.
i) Help software developers
   - Choose design alternatives
   - Improve documentation and maintenance of systems by furnishing an explicit specification of intent
   - Help designers get a design “right” faster.
   - Standardize terminology
j) Show more than just the solution:
   - context (when and where)
   - forces (trade-off alternatives, misfits, goals and constraints)
   - resolution (how and why the solution balances the forces)

6. Benefits of using Design Patters

a) A Common Design Vocabulary:

Design Patterns provide a common Vocabulary for designers to use to communicate, document, and explore design alternatives. Design Patterns make a system seem less

complex by letting one talk about it at a higher level of abstraction than that of a design notation or programming language. Design patterns raise the level at which one design and discuss design with your colleagues.

b) A Documentation and Learning Aid:

Most large object-oriented systems use these design patterns. People learning object-oriented programming often complain that the systems they're working with use inheritance in convoluted ways and that it's difficult to follow the flow of control.. in large part this is because they do not understand the design patters in the system. Learning these design patterns will help one understand existing object-oriented systems.

These design patterns can also make one better designer. They provide solutions to common problems. Learning these patterns will help a novice act more like an expert. Describing a system in terms of design patterns that it uses will make it a lot easier to understand.

c) An Adjunct to Existing Methods:

The design patterns show how to use primitive techniques such as objects, inheritance and polymorphism. They show how to parameterize system with an algorithm, behavior, a state, or the kind of objects it's supposed to create. Design patterns provide a way to describe more of the "why" of a design and not just record the results of your decisions. The Applicability, Consequences, implementation sections of the design patterns help guide one in the decisions one have to make.

The programming language and class libraries you use affect the design. Analysis models often must be redesigned to make them reusable. Many of the design patterns in the catalogue address these issues, which is why we call them design patterns.

7. Drawbacks of using Design Patterns

a) Patterns do not lead to direct code reuse.

b) Patterns are deceptively simple.

c) Teams may suffer from pattern overload.

d) Patterns are validated by experience and disturbance rather than by automated testing.

e) Integrating patterns into a software development process is a human-intensive activity.

8. Software Patterns are not...

a) Restricted to software design or Object-Oriented design
b) Untested ideas/theories or new inventions
c) Solutions that have worked only once
d) Any old thing written-up in pattern format
e) Abstract principles or heuristics
f) Universally applicable for all contexts
g) A "silver bullet" or panacea

9. Software Patterns are...

a) Recurring solutions to common problems of design
b) Practical/concrete solutions to real world problems
c) Context specific
d) "Best-fits" for the given set of concerns/trade-offs
e) "Old hat" to seasoned professionals and domain experts
f) A literary form for documenting best practices
g) A shared vocabulary for problem-solving discussions
h) An effective means of (re)using, sharing, and building upon existing wisdom/experience/expertise
i) Massively hyped!

10. **Different Types of Design Patterns**

a) Creational Patterns:

"Creational design patterns abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented. A class creational pattern uses inheritance to vary the class that's instantiated, whereas an object creational pattern will delegate instantiation to another object." We separate the object from how it's *directly* created. As we delve into the patterns the reasons for doing so should become clear.

Different types of creational patterns are as follows.

- **Abstract Factory Pattern:** Creates an instance of several families of classes.
- **Builder Pattern:** Separates object construction from its representation.
- **Factory Method Pattern Test:** Creates an instance of several derived classes.
- **Prototype Pattern:** A fully initialized instance to be copied or cloned.
- **Singleton Pattern:** A class of which only a single instance can exist.

b) Structural Patterns:

"Structural patterns are concerned with how classes and objects are composed to form larger structures. Structural class patterns use inheritance to compose interfaces or implementations. As a simple example, consider how multiple inheritances mix two or more classes into one. The result is a class that combines the properties of its parent classes. This pattern is particularly useful for making independently developed class libraries work together."

Different types of structural patterns are as follows.

- **Adapter Pattern:** Match interfaces of different classes.
- **Bridge Pattern:** Separates an object's interface from its implementation.
- **Composite Pattern:** A tree structure of simple and composite objects
- **Decorator Pattern:** Add responsibilities to objects dynamically.
- **Façade Pattern:** A single class that represents an entire subsystem.
- **Flyweight Pattern:** A fine-grained instance used for efficient sharing.
- **Proxy Pattern:** An object representing another object.

c) Behavioral Patterns:

Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

- **Chain of Resp. Pattern:** A way of passing a request between a chain of objects.
- **Command Pattern:** Encapsulate a command request as an object
- **Interpreter Pattern:** A way to include language elements in a program
- **Iterator Pattern:** Sequentially access the elements of a collection
- **Mediator Pattern:** Defines simplified communication between classes
- **Memento Pattern:** Capture and restore an object's internal state
- **Observer Pattern:** A way of notifying change to a number of classes
- **State Pattern:** Alter an object's behavior when its state changes
- **Strategy Pattern:** Encapsulates an algorithm inside a class
- **Template Method Pattern:** Defer the exact steps of an algorithm to a subclass
- **Visitor Pattern:** Defines a new operation to a class without change

11. Conclusion

Patterns represent distilled experiences which, through their assimilation, convey expert insight and knowledge to inexpert developers. They help forge the foundation of a shared architectural vision, and collective of styles. If we want software development to evolve into a mature engineering discipline, then these proven "best practices" and "lessons

learned" must be aggressively and formally documented, compiled, scrutinized, and widely disseminated as patterns (and anti-patterns). Once a solution has been expressed in pattern form, it may then be applied and reapplied to other contexts, and facilitate widespread reuse across the entire spectrum of software engineering artifacts such as: analyses, architectures, designs, implementations, algorithms and data structures, tests, plans, and organization structures.

Perhaps the final remarks from **Pattern-Oriented Software Architecture** best describe the significance of patterns for software:

Patterns expose knowledge about software construction that has been gained by many experts over many years. All work on patterns should therefore focus on making this precious resource widely available. Every software developer should be able to use patterns effectively when building software systems. When this is achieved, we will be able to celebrate the human intelligence that patterns reflect, both in each individual pattern and in all patterns in their entirety.

12. References

a) http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html

b) http://hillside.net/patterns/definition.html

c) http://www.gamedev.net/reference/articles/article1416.asp

d) http://www.cs.wustl.edu/~schmidt/CACM-editorial.html

e) http://www.research.ibm.com/designpatterns/pubs/top10misc.html

f) http://www.research.ibm.com/designpatterns/pubs/dwp-tutorial.pdf

g) http://www.research.ibm.com/designpatterns/pubs/dp-tutorial.pdf

h) http://sern.ucalgary.ca/courses/SENG/609.04/W98/notes/index.html

i) http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/patterns

j) Design Patterns by Erich Gamma (Author), Richard Helm (Author), Ralph Johnson (Author), John Vlissides (Author)