Refactoring to Patterns

Jim Fawcett

CSE776 – Design Patterns

Summer 2005

Reference

- Refactoring to Patterns, Joshua Kerievsky, Addison-Wesley, 2005

Agenda

- This presentation provides a summary of the content from this well received book.

Background

- Evolutionary Programming
  - A development process that focuses on incremental development.
  - Usually set up as a sequence of development cycles, each culminating in a software release.
  - Each item from the sequence provides some planning, design, implementation, and test.
  - Sometimes the elements are inverted, as is the case for extreme programming:

Extreme Programming

- Process Model Phases:
  - A sequence of releases as per Evol. Prog.
  - Steps to achieve a release:
    - Specify the functionality for next release
    - Design Test(s)
    - Write code to make tests pass
    - Refactor code to improve the design
    - Constantly run regression tests on the entire build
      - Goal is to stay in state where all tests pass
    - Release when the implemented functionality is incorporated in the build.

Refactoring

- Refactoring is a major part of the Evolutionary style of programming, especially for Extreme Programming.
- But how do you successfully refactor?
  - What are the goals?
  - When should you refactor?
  - How do you do it?
  - How do you know when to stop?

A Selection of Chapters

- Chap 3: Patterns
- Chap 4: Code Smells
- Chap 5: Catalog of Refactorings to Patterns
- Chap 6: Creation
- Chap 7: Simplification
- Chap 8: Generalization
- Chap 9: Protection
- Chap 10: Accumulation
- Chap 11: Utilities

Patterns

- **What is a pattern?**
  - "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution"
    - Christopher Alexander, Architect
- **There are many ways to implement a pattern**
  - "Every 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 samy way twice." – Christopher Alexander

Refactoring to, towards, and away from Patterns

- "Good designers refactor in many directions, always with the goal of reaching a better design. While many of the refactorings I apply don't involve patters (i.e., they're small, simple transformations, like Extract Method …), when my refactorings do involve patterns, I refactor to, towards, and even away from patterns." – Joshua Kerievsky

Code Smells

- "It's … necessaryto learn common design problems so you can recognize them in your code."
- Robert Martin, Martin Fowler, and Kent Beck have all written about specific "Code Smells"

Catalog of Code Smells

- Duplicated Code
- Long Method
- Conditional Complexity
- Primitive Obsession
- Indecent Exposure
- Solution Sprawl
- Alternative Classes with Different Interfaces
- Lazy Class
- Large Class
- Switch Statements
- Combinational Explosion
- Oddball Solution

Catalog of Patterns

- Replace Ctors with Creation methods, chain Ctors
- Encapsulate Classes with Factory
- Introduce Polymorphic Creation with Factory Method
- Replace Conditional Logic with Strategy
- Form Template Method
- Compose Method
- Replace Implicit Tree with Composite
- Encapsulate Composite with Builder
- Move Accumulation to Collecting Parameter
- Extract Composite, Replace one/many with Composite.
- Replace Conditional Dispatcher with Command
- Extract Adapter, Unify Interfaces with Adapter
- Replace Type Code with Class

Catalog of Patterns

- Replace State-Altering Conditionals with State
- Introduce Null Object
- Inline Singleton, Limit Instantiation with Singleton
- Replace Hard-Coded Notifications with Observer
- Move Embellishment to Decorator, Unify Interfaces, Extract Parameter
- Move Creation Knowledge to Factory
- Move Accumulation to Visitor
- Replace Implicit Language with Interpreter

A Typical Catalog Item

- States Pattern Name and Intent
- Gives an application example
- Discusses motivation
- Benefits and Liabilities
- Mechanics
  - Specific things to do
- Presents detailed example

Some non-GoF Patterns

- Compose Method
  - Extract blocks of code as methods with intent revealing names
- Chain Constructors
  - Remove duplicate code in constructors by calling, in a constructor, other constructor(s).
- Extract Parameter
  - Assign, to a field, a caller provided parameter instead of a locally instantiated object.
- Move Accumulation to Collecting Parameter
  - Replace a complex function that accumulates results with a set of Composed Methods which are passed a Collecting Parameter.

Summary

- Interesting ideas, presented clearly
- Book is on restricted hold in Sci-Tech library.