

Design Practices

17-654/17-754: Analysis of Software Artifacts

Jonathan Aldrich



Specification: The Starting Point for Design



- **Functionality**
 - Usually a set of use cases
 - Detailed scenarios of system use
 - Includes normal and exceptional cases
 - Less often: mathematical specifications
- **Quality attributes**
 - **Expected areas of extension**
 - Robustness, Security
 - Performance, Fault-tolerance
- We'll talk more about specifications and requirements gathering later

8 April 2008

Identifying Classes: Noun Extraction



- Start with short problem description
- Identify the nouns and analyze
 - External entities: leave out
 - unless system needs to model them
 - example: "The User"
 - Tangible entities: classes
 - Abstract nouns: classes or attributes (fields)
 - weight, brightness, size
 - Complex abstract nouns might end up as a class
 - e.g. Color, Message, Event
- Add
 - Boundary classes: interaction with world
 - Typically one per screen/dialog
 - Control classes: encapsulate non-trivial computations
 - Data structures that support the entities
 - Classes for abstract implementation concepts
 - Controller, Router, Manager, ...

8 April 2008

What should be a Class?



- Retained information
 - Need to remember data about the object
- Needed services
 - Operations that change attribute values or compute information
- Multiple attributes
 - Class groups data related by a concept
 - No class usually needed for a scalar
- Common attributes & operations
 - A set of attributes/operations is common to many objects
- Essential requirements
 - Entities in the problem space

8 April 2008

Source: [Coad and Yourdon 91]

Example: Noun Extraction



To-Do List Application

- The to-do list application is designed to keep track of tasks for the user. Tasks can be sorted by due date and priority. New tasks can be added, and existing tasks can be canceled or completed. The initial application is web-based, but we would like to support disconnected operation on a laptop or PDA in the future.

To-Do Application Use Cases

- Sign up for an account, with username and password
- Log in to system with username and password
- Add a to-do item with name, priority, and due date
- Show to-do items by date
- Show to-do items by date
- Cancel a task
- Complete a task

8 April 2008

Abstract Design: CRC Cards




- Class-Responsibility-Collaboration
 - Name of class
 - Responsibilities/functionality of the class
 - Other classes it invokes to achieve that functionality
- Responsibility guidelines
 - Spread out functionality
 - No "god" classes – make maintenance difficult
 - State responsibilities generally
 - More reusable, more abstract
 - Group behavior with related information
 - Enhances cohesion, reduces coupling
 - Promotes information hiding of data structures
 - Information about one thing goes in one place
 - Spreading it out makes it hard to track

8 April 2008

CRC Validation



- Validation
 - Ensure all functionality in specification is covered by some class
 - Reason through how functionality could be achieved
 - Abstractly executing the program
 - What other classes are needed?
 - Are their responsibilities enough for this class to do what it needs to do?
- Refine as needed 

8 April 2008

Attributes, Associations, and Operations



- Go through use cases
 - Attribute: something that belongs to a class
 - Needed for computation in the use case
 - Association: one class stores another
 - Usually implemented by a field or collection—but keep abstract early in design
 - Operation: verbs in use cases
 - OO: usually goes in the object on which the verb operates
 - Categories
 - accessors: access data
 - mutators: manipulate data
 - computational methods

8 April 2008

Quality Attributes



- So far, we've focused on capturing functionality in a design
- But *good* design is primarily about *quality attributes*, e.g.
 - *Extensibility* – ability to easily add and change capabilities
 - *Robustness* – operate under stress or invalid input
 - *Usability* – ability for users to easily accomplish tasks
 - *Security* – withstand attacks
 - *Fault-tolerance* – recover from component failure
 - *Performance* – yields results at a high rate or with low latency

8 April 2008

Refining a Design



- Step through Use Cases
 - Verify completeness of diagram by asking:
 - Which methods execute?
 - What methods are called?
 - What does each method or object have to know?
- Consider quality attributes
 - Make concrete with a test
 - e.g. modification scenario, performance target
 - Generate multiple designs – **NOT JUST ONE!**
 - What design patterns achieve this attribute?
 - May be helpful to have different people develop designs independently
 - Evaluate designs
 - How well does this design achieve the entire set of quality attributes?
 - May require prioritizing attributes

8 April 2008

Design Patterns

17-654/17-754: Analysis of Software Artifacts

Jonathan Aldrich



Design Patterns



- "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

8 April 2008

History



- Christopher Alexander, *The Timeless Way of Building* (and other books)
 - Proposes patterns as a way of capturing design knowledge in architecture
 - Each pattern represents a tried-and-true solution to a design problem
 - Typically an engineering compromise that resolves conflicting forces in an advantageous way

8 April 2008

Patterns in Physical Architecture



- When a room has a window with a view, the window becomes a focal point: people are attracted to the window and want to look through it. The furniture in the room creates a second focal point: everyone is attracted toward whatever point the furniture aims them at (usually the center of the room or a TV). This makes people feel uncomfortable. They want to look out the window, and toward the other focus at the same time. If you rearrange the furniture, so that its focal point becomes the window, then everyone will suddenly notice that the room is much more "comfortable".
 - Leonard Budney, Amazon.com review of *The Timeless Way of Building*

8 April 2008

Benefits of Patterns



- Shared language of design
 - Increases communication bandwidth
 - Decreases misunderstandings
- Learn from experience
 - Becoming a good designer is hard
 - Understanding good designs is a first step
 - Tested solutions to common problems
 - Where is the solution applicable?
 - What are the tradeoffs?

8 April 2008

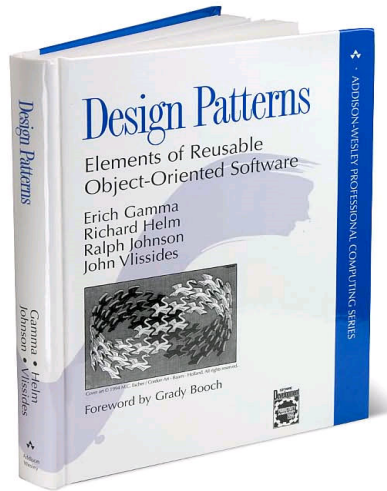
Elements of a Pattern



- Name
 - Important because it becomes part of a design vocabulary
 - Raises level of communication
- Problem
 - When the pattern is applicable
- Solution
 - Design elements and their relationships
 - Abstract: must be specialized
- Consequences
 - Tradeoffs of applying the pattern
 - Each pattern has costs as well as benefits
 - Issues include flexibility, extensibility, etc.
 - There may be variations in the pattern with different consequences

8 April 2008

History: Design Patterns Book



- Brought Design Patterns into the mainstream
- Authors known as the Gang of Four (GoF)
- Focuses on *descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context*
- Great as a reference text
- Uses C++, Smalltalk

8 April 2008

A More Recent Patterns Text



- Uses Java
 - The GoF text was written before Java went mainstream
- Good pedagogically
 - Lots of examples and explanation
 - GoF is really more a reference text

8 April 2008

Patterns to Know



- Strategy, Observer, Decorator, Factory, Singleton, Command, Adapter, Facade, Template Method, Iterator, Composite, State, Proxy, and Model-View-Controller
- Know pattern name, problem, solution, and consequences

8 April 2008