Object-Oriented Programming

Design Patterns

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Observations

- There are repeated design structures and roles of classes used in most software

- Not possible to capture these structures and roles into specific classes, i.e., they are part of a class but do not make up the entire class

- Language features may support these if organized correctly

- Need a way to communicate about them

Patterns

Issues

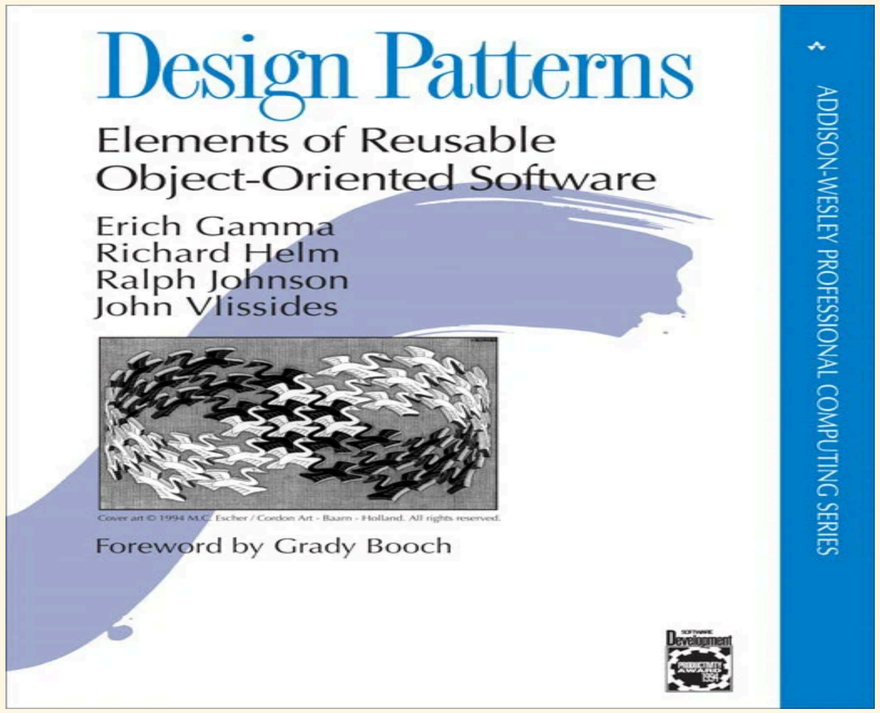
- Must be widely applicable
- Solution must be safe
- The solution should be efficient

Origins

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 the solution a million times over, without ever doing it in the same way twice

Christopher Alexander, "A Pattern Language", 1977

Software Design Patterns



- *Design Patterns: Elements of Reusable Object-Oriented Software* Gamma, Helm, Johnson, Vlissides

- AKA, "Gang of 4 Book" (GOF'95)

Elements of Design Patterns

- *Name*

- *Problem*

- *Solution*

- *Consequences*

Pattern Categories

- ***Creational Patterns*** E.g., *Factory Method*

- ***Structural Patterns*** E.g., *Proxy*

- ***Behavioral Patterns*** E.g., *Template Method*

Creational Patterns

How the program creates objects

- E.g., *Factory Method*

- Are more important when we choose object composition over class inheritance

- Class inheritance provides a fixed set of behaviors, while object composition provides a greater variety of ways to combine functionality

Creational Patterns

Pattern	Description
Factory Method	Instantiation is deferred to subclasses
Builder	Separate object construction from representation
Abstract Factory	Create families of objects using only abstract classes
Prototype	Create new objects by copying an existing object
Singleton	Only allow one object of a class to exist

Structural Patterns

How classes and objects are composed to form larger objects

- ***structural class patterns***

  Use inheritance to compose an interface or implementation

  Static, compile-time only

- ***structural object patterns***

  Compose objects for new objects

  Static and dynamic, compile-time and runtime

Structural Patterns

Pattern	Description
Proxy	A surrogate for an object
Adapter	Convert the interface of an object into one that clients expect
Bridge	Decouple an abstraction from its implementation so both can vary independently
Composite	Allows clients to treat individual objects and compositions of objects uniformly
Decorator	Attach features to an object dynamically instead of through subclassing
Facade	Provides a unified interface to a set of interfaces in a subsystem, simplifying its use
Flyweight	Efficiently handles large numbers of fine-grained objects

Behavioral Patterns

Algorithms and the assignment of responsibilities between objects

- *class behavioral patterns* Use inheritance
- *object behavioral patterns* Use composition

Behavioral Patterns

Pattern	Description
Template Method	A class pattern that defines the skeleton of an algorithm as an abstract class, allowing its subclasses to provide concrete behavior
Interpreter	Implements a specialized language
Mediator	Allows loose coupling between classes by being the only class that has detailed knowledge of their methods
Chain of Responsibility	Delegates commands to a chain of processing objects
Observer	A publish/subscribe pattern that allows many observer objects to see an event
Strategy	Allows one of a family of algorithms to be selected on the fly at runtime
Command	Creates objects that encapsulate actions and parameters
State	Allows an object to alter its behavior when its internal state changes
Iterator	Accesses the elements of an object sequentially without exposing its underlying representation
Memento	Provides the ability to restore an object to its previous state (undo)
Visitor	Separates an algorithm from an object structure by moving the hierarchy of methods into one object