[CS3704] Software Engineering

Shawal Khalid
Virginia Tech
3/22/2024

Design Patterns (i.e. Low-Level Design)

Design patterns are descriptions of *communicating objects and classes* that are customized to solve a general design problem in a particular context.

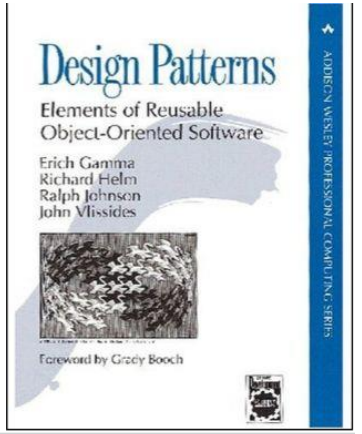
The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities.

Design Patterns (cont.)

Why design patterns?

- Appy working solutions to approaches
- Based on the implementations of many systems
- Capture and pass on the knowledge of experienced designers
  - Useful for inexperienced
  - Communicating about design

But do software engineers actually use them?



Design Pattern Families

Creational

Concerned with the process of object creation

- Increases flexibility and reuse of code

Structural

Deal with the composition of classes or objects

- Organizing different classes and modules to form larger structures or add new functionality

Behavioral

Characterize the ways in which classes or objects interact and distribute responsibility

- Algorithms and assignment of responsibilities between objects

Creation Patterns

- **Abstract Factory**: Creates an instance of several families of classes
- **Builder**: Separates object construction from its representation
- **Factory Method**: Creates an instance of several derived classes
- **Object Pool**: Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- **Prototype**: A fully initialized instance to be copied or cloned
- **Singleton** A class of which only a single instance can exist

More details later…

Structural Patterns

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

Behavioral Patterns

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

Design Disclaimer

- No silver bullet for choosing high-level or low-level design patterns.
- Design will change as requirements and code change.
  - First Law!

High-level design processes, patterns, and issues will differ based on the domain of the product you are implementing!

Design Workshop

- Find a small group (3-4)
- Create a *prototype, storyboard, wireframe, etc.* for the given program (Using any software or paper/pencil)
- Complete as much as possible in class time
- Turn in with group member first and last names, PIDs, and who did which role.

Design Workshop

Design a user interface for new social media platform (web/mobile) called Top Five, that allows users to list their top five favorite things in any topic (i.e. songs, movies, foods, etc.). Turn in by 11:59pm. This is graded on completion/reasonable effort.