Software Design

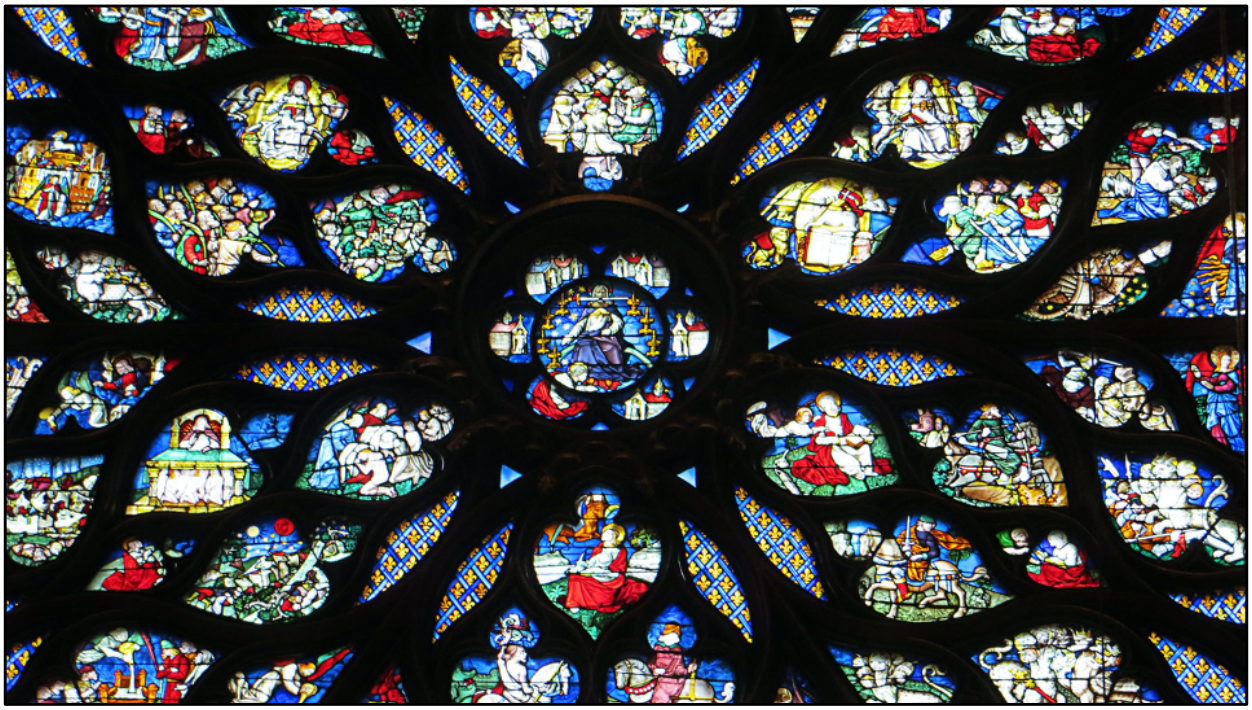
Jeffrey Carver
University of Alabama
carver@cs.ua.edu

Andrew Loftus
University of Illinois
aloftus@illinos.edu

Overview

- (Brief) Introduction to Software Design - Jeff
- Think Like a Programmer – Andrew
- BREAK
- Introduction to Design Patterns - Jeff

(Brief) Introduction to Software Design



Why is it necessary to have good software design?
Does software design have to be heavyweight?
Do you have to do it all up front?

One definition of design:

- Firmness – no bugs that inhibit function
- Commodity – Suitable for intended purpose
- Delight – Experience of using software should be pleasurable

Design Principles

- Traceability
- Minimize intellectual distance
- Don't reinvent the wheel
- Accommodate change
- Degrade gracefully

- Traceability – maps back to analysis model
- Intellectual distance – between the software and the real world
- Degrade gracefully – Even when there is bad data or error conditions

Fundamental Concepts

- Abstraction
- Patterns
- Modularity
- Hiding
- Functional independence

- Functional Independence – single-minded functions
- Briefly discuss each of these and why they are important concepts

Think Like a Programmer

https://docs.google.com/presentation/d/1eNpMYEyS2x92P2r94pWwykfHSS2KaRHxzRsRN85WFh0/edit

Introduction to Design Patterns

- Design Patterns are standard ways to solve common programming problems
- DP are not created – they emerge
- Each pattern describes a problem that occurs frequently, along with a standard solution
- DP include context that explain when they work along with limitations and constraints

Behavioral Patterns

Assignment of responsibility between objects and communication between

Chain of Responsibility

- Problem: Need a common interface by which requests can be sent without knowing which specific method will handle the request or parts of the request.
- Solution: Decouple sender and receiver by providing multiple methods to handle the request (or parts of the request)
- Example?

Problem
- Requesting objects do not always know who should handle the request (or each part of the request)
- Request handlers may need to be changed dynamically

Solution
- A “chain” of methods – each handles what it is supposed to and then passes the remaining needs along the chain

Example
- In a situation where there are multiple machines that could potentially handle a processing request, the client may not know which one is available. You don’t want to make the client figure this out. So, you can give an interface that passes the request to the most appropriate node.

Command

- Problem: Need to queue commands, need a history of commands, need an "undo"
- Solution:
  - Encapsulate commands as objects
  - Client instantiates the command object with the appropriate information (needed to be called later)
  - Invoker decides when method is called
- Examples?

Creational Patterns

Creation, composition, and representation of objects

Builder

- Problem: Need to create various representations of the same object
- Solution:
  - Abstract construction steps
  - Different implementations can occur for different objects
- Examples?

Example
- Need to build various types of automobiles. May have a builder class that includes the body, engine, tires, electronics, etc… Then you would instantiate those differently for different types of cars. But, the interface at the top is the same

Structural Patterns

How classes and objects are organized and integrated to build a larger structure

Decorator

- Problem:
  - Need to add new functionality to an object dynamically
  - Want different instances of a class to behave differently
- Solution
  - Enclose object inside of another object
  - Attach new responsibilities at runtime
  - Keep the same interface
- Examples?

Example: Adding functionality to a windowing system (e.g. scroll bars)

Adapter Pattern

- Problem:
  - Two classes with incompatible interfaces need to work together
  - An existing component may have needed functionality, but an incompatible interface
- Solution
  - Convert the interface into what is expected
- Examples?

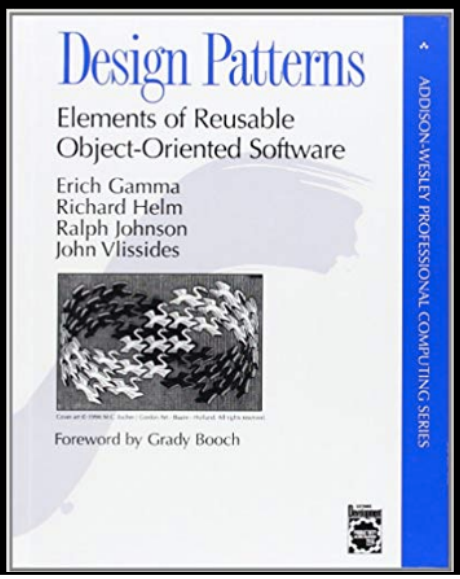
Example: Real world example – your device uses a plug for a US based outlet. When you travel to Europe, you need an adapter to make that interface still work (note that you aren't converting from 110 to 220) just adapting the interface.

Proxy

- Problem:
  - Need to temporarily provide a placeholder or surrogate
  - Something takes a long time to load – may not want to wait
  - Something is expensive to create – delay creation until needed
- Solution
  - Allow other objects to access through a placeholder
  - Proxy takes care of creating object (when time is right)
  - Proxy forwards requests to object
- Examples?

Example: Real-world example – large photo on a webpage. First a proxy is loaded then, if needed, the full image is loaded.

Design Patterns



More Python Examples

https://www.toptal.com/python/python-design-patterns

Software Design

Jeffrey Carver
University of Alabama
carver@cs.ua.edu

Andrew Loftus
University of Illinois
aloftus@illinos.edu

Photo Credits

- Slide 4 - Photo by lisaleo at Morguefile.com
- Slide 9 - https://flic.kr/p/brTebt