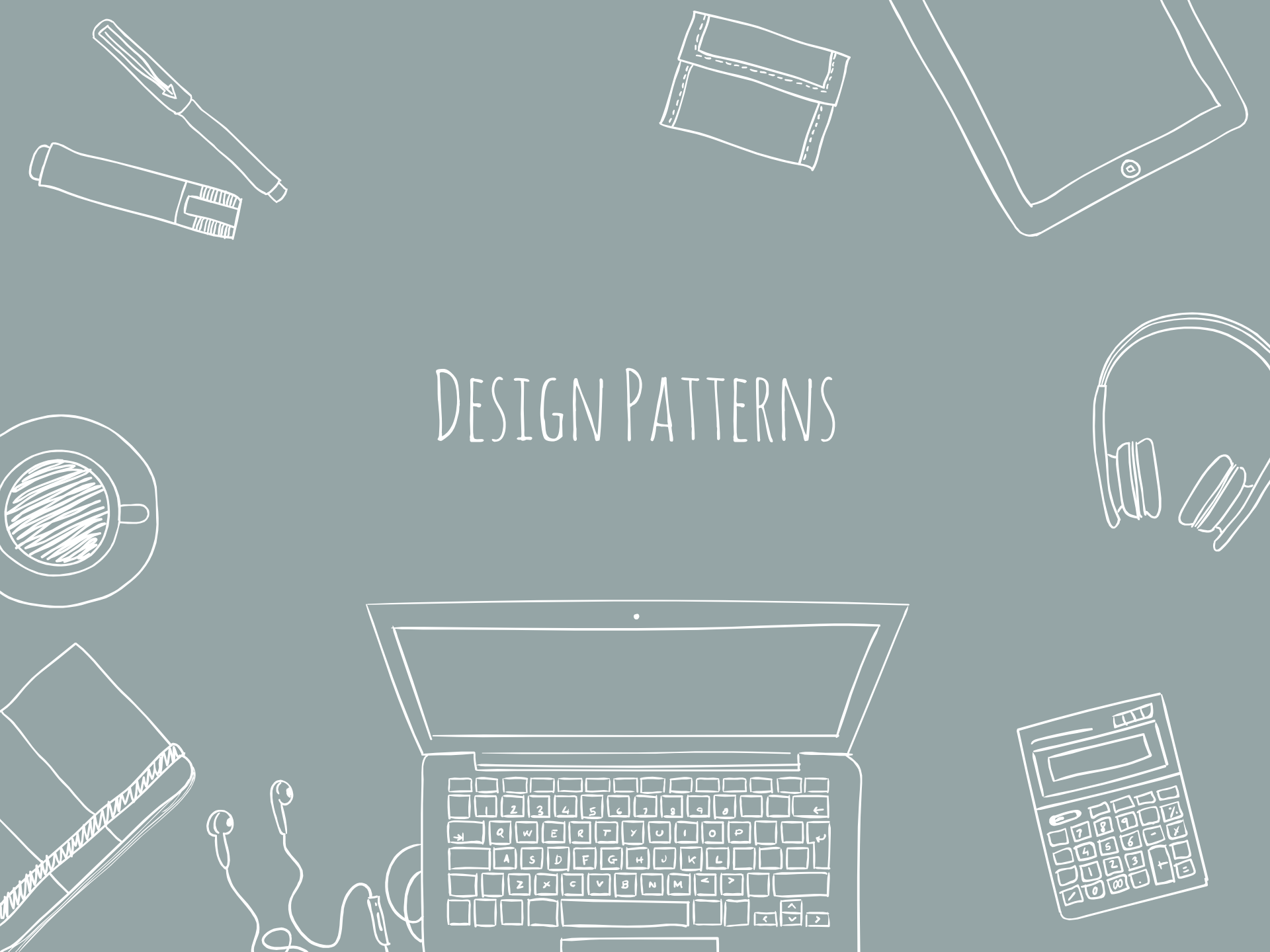
Software Engineering I

CS361

Announcements

- Friday Extra office hour of "Coding Lab" 2-3pm
- CI instructions added to Assignment 3
- travis-ci.ORG
- http://www.umlet.com/umletino/umletino.html

Design Patterns



Attribution

Much of this material inspired by a great slides from Kenneth M. Anderson, available here: https://www.cs.colorado.edu/~kena/classes/5448/f12/lectures/07-designpatterns.pdf

Also, here: https://sourcemaking.com/design_patterns/template_method

Christopher Alexander

- ✖ Worked as in computer science but trained as an architect
- ✖ Wrote a book called A Pattern Language: Towns, Buildings, Construction.
- ✖ Adopted as some cities as a building code

The timeless Way of Building

- Asks the question, “is quality objective?”
- Specifically, “What makes us know when an architectural design is good? Is there an objective basis for such a judgement?”

Approach

He studied the problem of identifying what makes a good architectural design by observing:

- buildings,
- towns,
- streets,
- homes,
- community centers,
- etc.

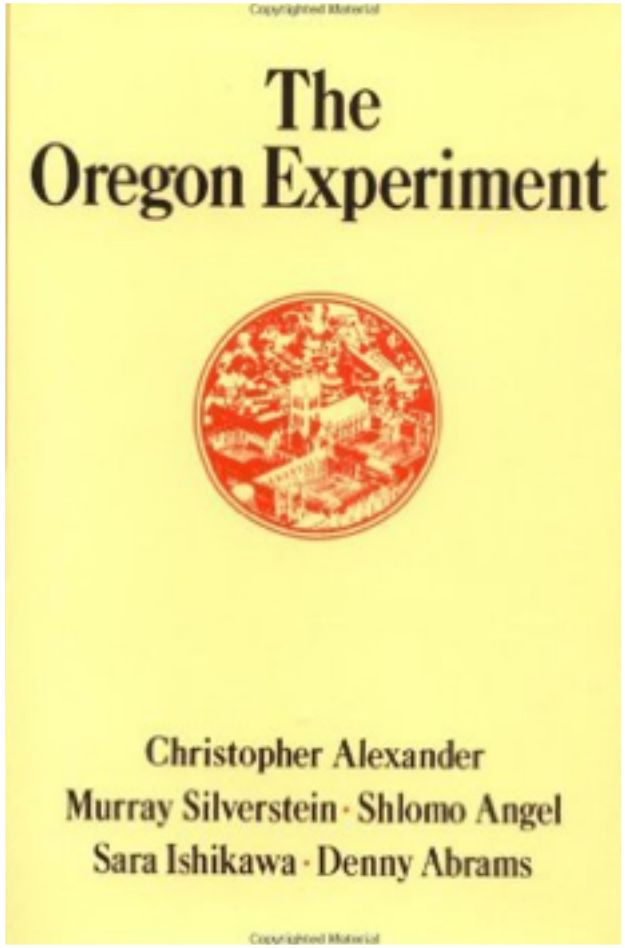
When he found a good example, he would compare with others.

Four Elements of a Pattern

Alexander identified four elements to describe a pattern:

- The name of the pattern
- The purpose of the pattern: what problem it solves
- How to solve the problem
- The constraints we have to consider in our solution

Inspired by Alexanders Work



Inspired by Alexanders Work

Inspired by Alexanders Work

https://archive.org/details/msdos_SimCity_1989

Inspired by Alexanders Work

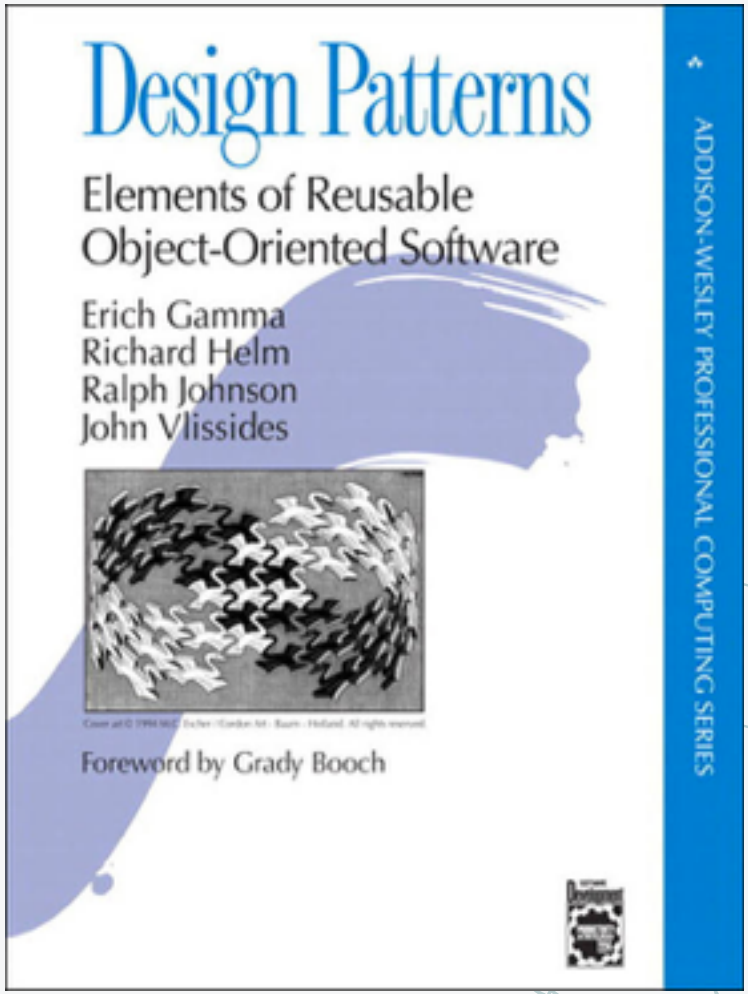
Software design patterns

- Are there problems in software that occur all the time that can be solved in somewhat the same manner?

- Is it possible to design software in terms of patterns?

Design Patterns

- 1995 book first introduced Design Patterns
- 23 Patterns in first
- Since then, many more design patterns have been published
- Authors knows as “Gang of Four”



Key Features of a Pattern

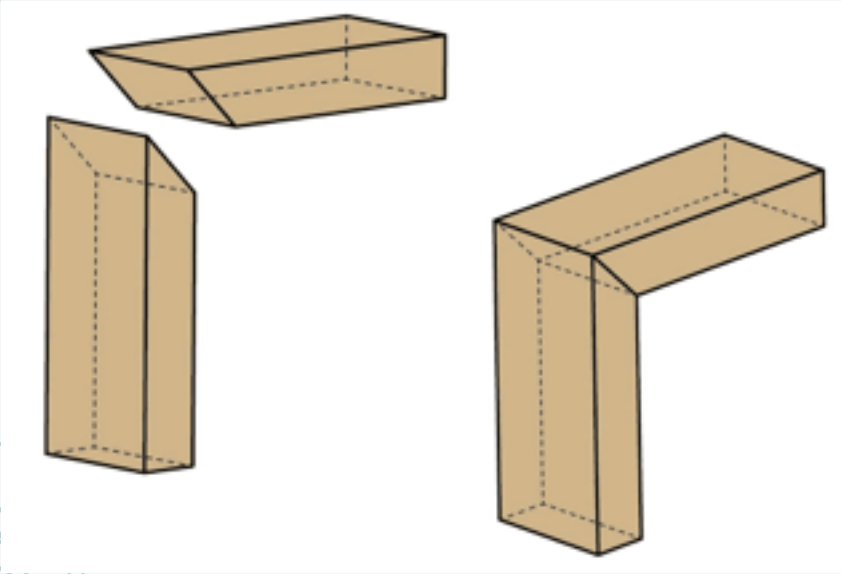
- **Name**
- **Intent**: The purpose of the pattern
- **Problem**: What problem does it solve?
- **Solution**: The approach to take to solve the pattern
- **Participants**: The entities involved in the pattern
- **Consequences**: The effect the pattern has on your software
- **Implementation**: Example ways to implement the pattern
- **Structure**: Class Diagram

Why Study Design Patterns?

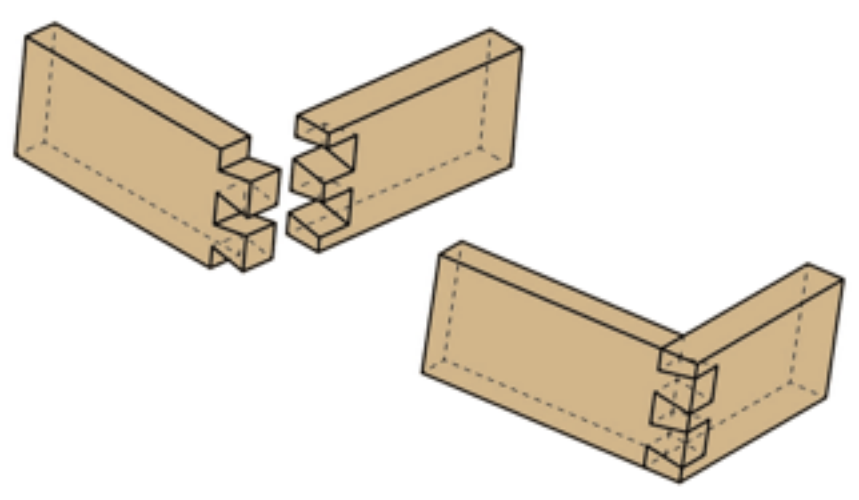
Patterns let us

- reuse solutions that have worked in the past; why waste time reinventing the wheel?
- have a shared vocabulary around software design.
  - e.g., “What if we used a facade here?”

Example of Higher-Level Perspective



Miter Joint



Dovetail Joint

Example of Higher-Level Perspective

When two carpenters are deciding how to make a joint, They could say:

"Should we use a dovetail or miter joint?"

"Should I make the joint by cutting down into the wood and then going back up 45 degrees and…"

Example of Higher-Level Perspective Cont...

The former avoids getting bogged down in details

The former relies on the carpenter's shared knowledge

- They both know that dovetail joints are higher quality then miter joints but with higher costs
- Knowing that, they can debate whether the higher quality is needed in the situation they are in

Design Pattern Categories

Creational Design Patterns

Design patterns about class instantiation

Structural Design Patterns

All about Class and Object composition

Behavioral Design Patterns

All about Object Communication

Creational 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

Singleton

Singleton - Problem

✖ Application needs one, and only one, instance of an object. Additionally, it must have lazy initialization and global access.

Singleton - Intent

- ✖ Ensure a class has only one instance, and provide a global point of access to it.

- ✖Encapsulated “just-in-time initialization” or “initialize on first use”

Singleton - Discussion

The class of the single instance object is should be responsible for:

- creation
- initialization
- access
- enforcement

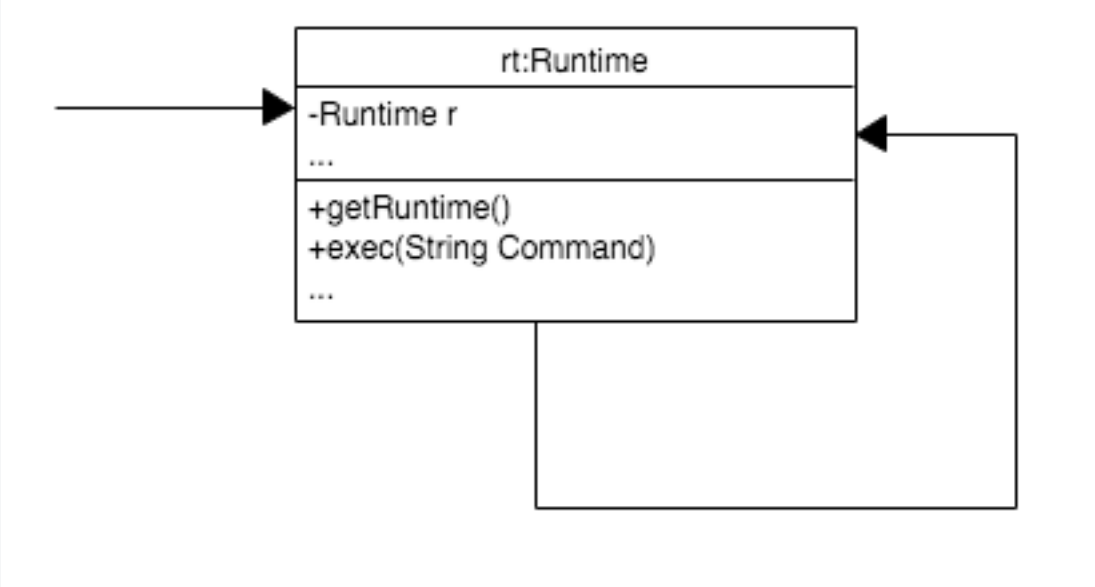
Singleton - Discussion

Singleton should be used when:

- Ownership of a single instance cannot be reasonably assigned
- Lazy initialization is desirable
- Global access is not otherwise provided for

Example Code

Singleton - UML



Singletons - Pros and Cons

Structural design 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

Facade

https://en.wikipedia.org/wiki/Florence_Cathedral

Facade - problem

✖ Complexity is the biggest problem that developers face.

✖ Clients want functionality without having to understand/ master functionality of entire system

Facade - Intent

- Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

- Wrap a complicated subsystem with a simpler interface

Facade Discussion

- ✖ Encapsulates the a complex system within a single interface object
- ✖ Reduces the learning curve necessary to leverage the subsystem

Example Code

Behavioral design 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

Template Method - problem

- Two different components have significant similarities, but demonstrate no reuse of common interface or implementation. If a change common to both components becomes necessary, duplicate effort must be expended.

Template Method - intent

- Define the skeleton of the operation, but differ some steps to client subclasses.

- Base class declares algorithm placeholders and derived classes implement the placeholders

Template Method - Discussion

- The overall algorithm is the same, but certain steps vary.
- The abstract class defines the overall algorithm, as well as the invariant steps
- Each subclass defines the variant steps

Example Code

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash