

# Software Engineering

---

CS / COE 1530  
Object-oriented Design Patterns

CS 1530 Software Engineering Fall  
2004

---

---

---


---

---

---

---

---



# Patterns & Frameworks

- Motivation
- What is a pattern / framework
- Pattern categories
- Pattern examples

CS 1530 Software Engineering Fall  
2004

---

---

---


---

---

---

---

---



# Motivation

- Developing software is hard
- Developing reusable software is even harder
- Proven solutions include patterns and frameworks

CS 1530 Software Engineering Fall  
2004

---

---

---


---

---

---

---

---



## Overview

- Patterns support reuse of *software architecture and design*
  - patterns capture the static and dynamic structures and collaborations of successful solutions to problems that arise when building applications in a particular domain
- Frameworks support reuse of *detailed design and code*
  - a framework is an integrated set of components that collaborate to provide a reusable architecture for a family of related applications
- Both improve development time and reuse

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Patterns of Learning

- Successful solutions are often rooted in patterns
- Eg. in chess
- Learning to develop good software is similar to learning to play good chess

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## What are design patterns? - Analogy

- If I only know how the pieces move in chess
  - can I claim to know how to play chess?
  - can I win a game against a Master by only using my good judgment?
  - can I remember all my good experiences to repeat them in the future?
  - can I remember all my bad experiences to avoid them in the future?

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Chess Theory vs. Patterns

- Chess opening theory
  - what pieces should I move first, what are the consequences
- Middle game theory
  - what constitutes a general advantageous position, attack and defense, sacrifices
- Endgame theory
  - wins in 3 moves, drawn positions etc.
- Analysis Patterns
  - what architecture to choose, consequences
- Design Patterns
  - design that is flexible to change
- Idioms
  - implementations for a specific language

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Becoming a Design Master

- First, learn the rules
  - algorithms, data structures, languages
- Then, learn the principles
  - e.g.,. structures programming, modular programming, object-oriented programming
- However, to become a master of design, one must study the design of other masters
  - they contain patterns that must be understood, memorized and applied repeatedly

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Design Patterns

- Represent solutions to problems that arise when developing software in a particular context
  - i.e., pattern = problem / solution pair in context
- capture static & dynamic structure and collaboration among key participants in software design
  - particular useful in articulating how and why to resolve non-functional forces
- Facilitate reuse of successful software architectures and design

CS 1530 Software Engineering Fall 2004

---

---

---

---

---

---

---

---

## Design Patterns - Reuse Perspective

- Advantageous position that supports change within a context (reusability)
- Types of change:
  - new objects: how are they created, by whom
  - structural: what parts can change, what parts can grow
  - behavioral: what functionality could change, what functionality can be added
- What kind of changes do we want to tolerate and at what cost?

CS 1530 Software Engineering Fall 2004

---

---

---

---

---

---

---

---

## Example: Stock Quote Service

- There may be many observers
- Each may react differently to same notification
- The subject should be decoupled as much as possible from the observers
  - i.e., allow observers to change independently of the subject

CS 1530 Software Engineering Fall 2004

---

---

---

---

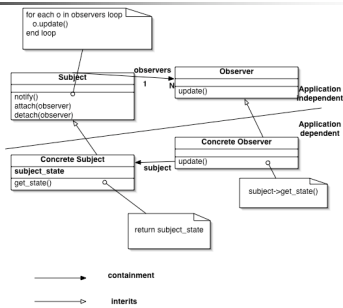
---

---

---

---

## Structure




---

---

---

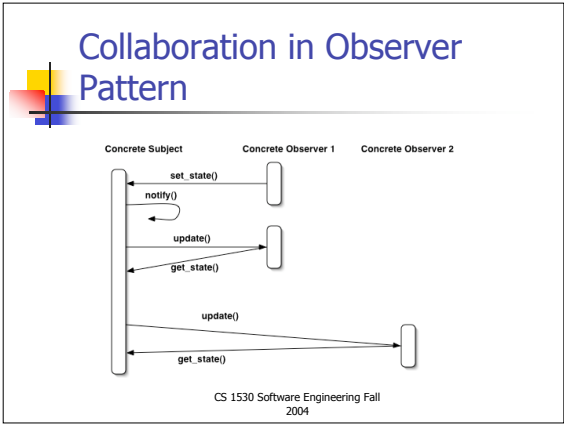
---

---

---

---

---




---

---

---

---

---

---

---

---

- ## Design Pattern Descriptions
- Name and intent
  - Problem and context
  - Force(s) addressed
  - Abstract description of structure and collaborations
  - Positive and negative consequences of use
  - Implementation guidelines and sample code
  - Known uses and related patterns
  - **Pattern descriptions are often independent of programming language or implementation details**
    - contrast with frameworks
- CS 1530 Software Engineering Fall 2004

---

---

---

---

---

---

---

---

- ## Frameworks
- are semi-complete applications
    - complete applications are developed by inheriting from and instantiating parameterized frameworks
  - provide domain-specific functionality
    - e.g., business or telecommunication applications, window systems etc.
  - exhibit inversion of control at run time
    - = framework determines which objects and methods to invoke in response to events
- CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Class Libraries / Frameworks & Patterns

- Class libraries
  - self-contained, pluggable ADTs
- Frameworks
  - reusable, semi-complete applications
- Patterns
  - problem, solution, context

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Patterns & Frameworks

- Highly synergistic
  - neither is subordinate
- Patterns can be characterized as more abstract descriptions of frameworks, which are implemented in a particular language
- Sophisticated frameworks embody dozens of patterns and patterns are often used to document frameworks

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Design Pattern Space

- Creational Patterns
  - initializing and configuring classes and objects
- Structural patterns
  - decoupling interface and implementation of classes and objects
- Behavioral patterns
  - dynamic interaction among groups of classes and objects

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Creational Patterns

- Factory method
  - method in a derived class creates associates
- Abstract factory
  - factory for building related objects
- Builder
  - factory for building complex objects incrementally
- Prototype
  - factory for cloning new instances from prototype
- Singleton
  - factory for a singleton (sole) instance

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Structural Patterns

- Adapter
  - translator adapts a server interface for a client
- Bridge
  - abstraction for binding one of many implementations
- Composite
  - structure for building recursive aggregations
- Decorator
  - extends an object transparently
- Facade
  - simplifies the interface for a subsystem
- Flyweight
  - many, fine-grained objects shared efficiently
- Proxy
  - one object approximates another (intermediary)

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Behavioral Patterns (1)

- Chain of responsibility
  - request delegated to designated service provider
- Command
  - request as first class object
- Interpreter
  - language interpreter for small grammar
- Iterator
  - aggregate elements are accessed sequentially

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Behavioral Patterns (2)

- Mediator
  - mediator coordinates actions between its associates
- Memento
  - snapshot captures and restores object states privately
- Observer
  - dependents update automatically when subject changes
- State
  - object whose behavior depends on its state

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## Behavioral Patterns (3)

- Strategy
  - abstraction for selecting one of many algorithms
- Template method
  - algorithm with some step supplied by a derived class
- Visitor
  - operations adapted to elements of a heterogeneous object structure

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## When to Use Patterns

- Solutions to problems that recur with variations
  - no need to reuse if problem arises only in one context
- Solutions that require several steps
  - patterns can be overkill if solution is simple linear sequence of steps
- Solutions where the solver is more interested in the existence of the solution than in the complete derivation
  - patterns leave out a lot of detail -- only a bridge to understanding

CS 1530 Software Engineering Fall 2004

---

---

---


---

---

---

---

---



## What Makes a Pattern a Pattern

- Solves a problem
  - must be useful
- Has a context
  - must describe where solution can be used
- Recurs
  - must be relevant in other situations
- Teaches
  - must provide sufficient understanding to tailor the solution
- Has a name
  - must be referred to consistently

CS 1530 Software Engineering Fall 2004

---

---

---

---

---

---

---

---