CS 4530
Software Engineering

Lecture 3 - Design Patterns

Jonathan Bell, John Boyland, Mitch Wand
Khoury College of Computer Sciences

Zoom Mechanics

- Recording: This meeting is being recorded
- If you feel comfortable having your camera on, please do so! If not: a photo?
- I can see the zoom chat while lecturing, slack while you're in breakout rooms
- If you have a question or comment, please either:
  - "Raise hand" - I will call on you
  - Write "Q: <my question>" in chat - I will answer your question, and might mention your name and ask you a follow-up to make sure your question is addressed
  - Write "SQ: <my question>" in chat - I will answer your question, and not mention your name or expect you to respond verbally

Today’s Agenda

Administrative:

- HW1 Discussion, due next Friday

Design patterns discussion

Activity: observer pattern

Reflection: What's a software design problem you've solved from an idea you learned from someone else?

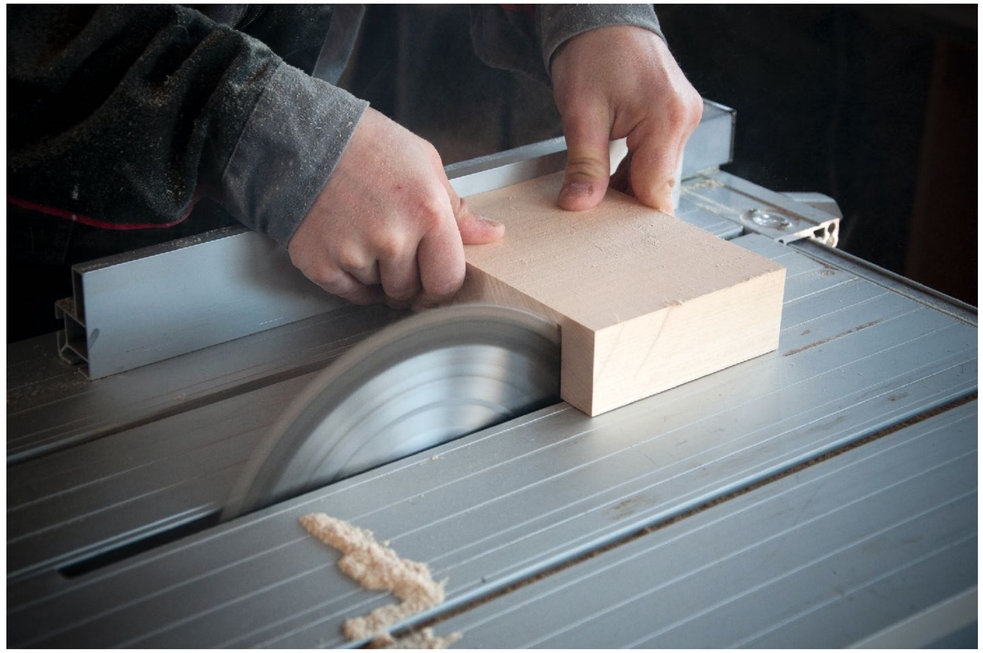
Discuss in small groups for 15 minutes, then at least 3-5 people will share their experiences with the whole class

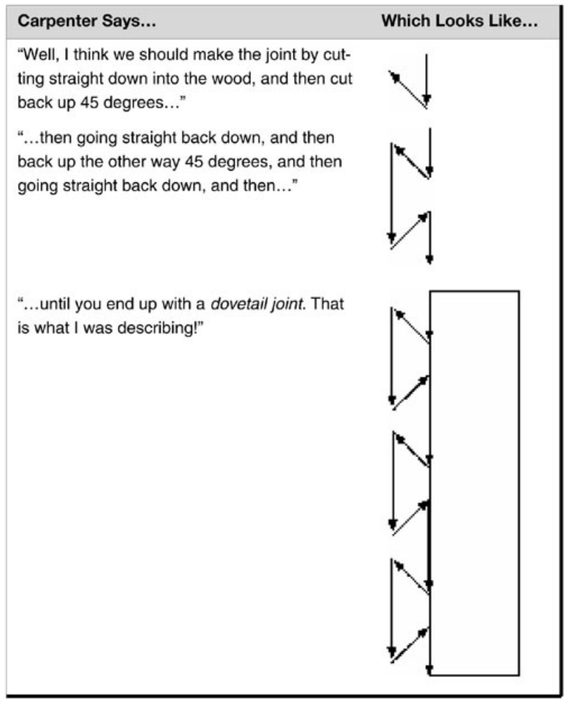
Review: Design Patterns

A design conversation

Q: How should we build a stack of drawers?

A: Well, I think we should make the joint by cutting straight down into the wood, and then cut back up 45 degrees, and then going straight back down, and then back up the other way 45 degrees, and then going straight back down, and then…





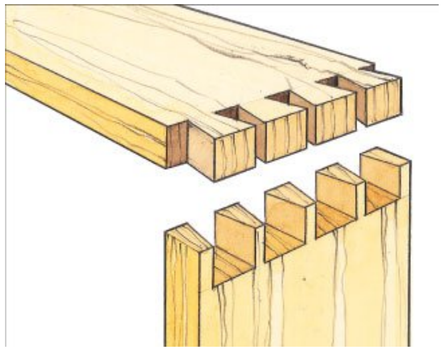
"Design Patterns Explained", Shalloway and Trott, Addison-Wesley, 2005

Review: Design Patterns

Asking the wrong questions

Carpenter: Should we use a dovetail or miter joint?

Other carpenter: Use a miter, it's: lighter and inconspicuous. A dovetail is a more complex, expensive joint that will be impervious to temperature and humidity, and it will look better. But, nobody will see the joint in the drawer, and it won't be in a situation with changing heat and humidity.



Dovetail Joint

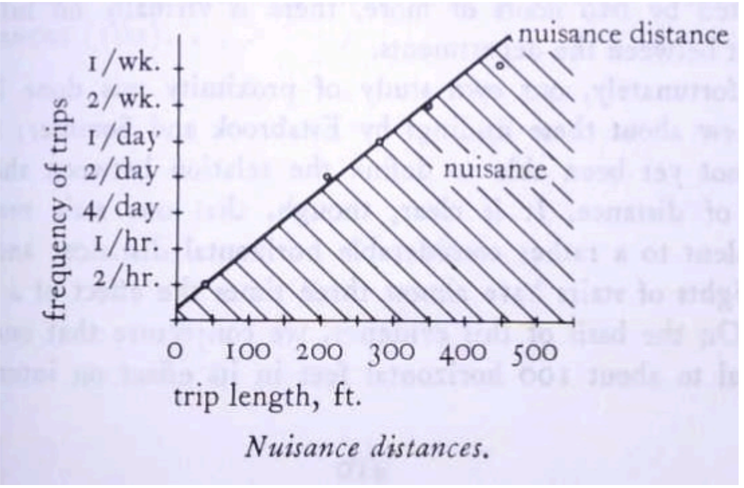
Miter Joint

"Design Patterns Explained", Shalloway and Trott, Addison-Wesley, 2005

More of Christopher Alexander's Patterns

No. 82: Office Connections

"If two parts of an office are too far apart, people will not move between them as often as they need to, and if they are more than one floor apart, there will be almost no communication between the two"

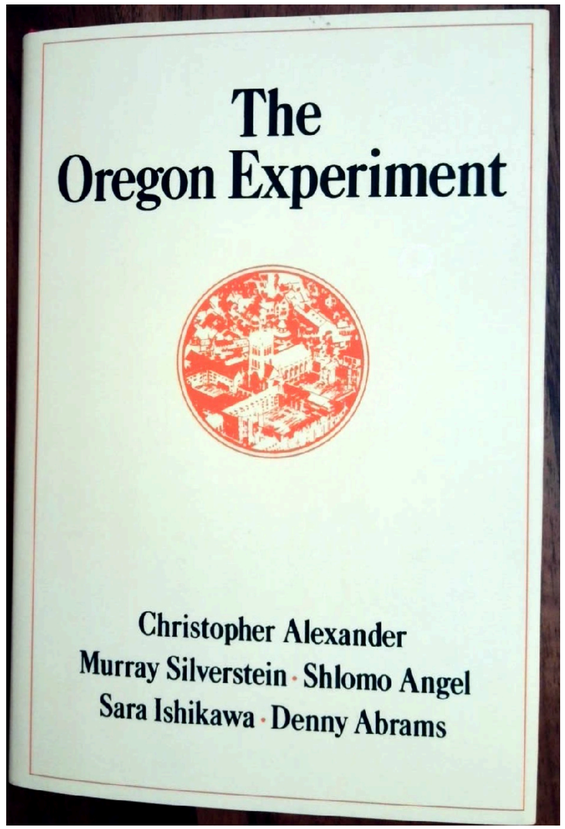


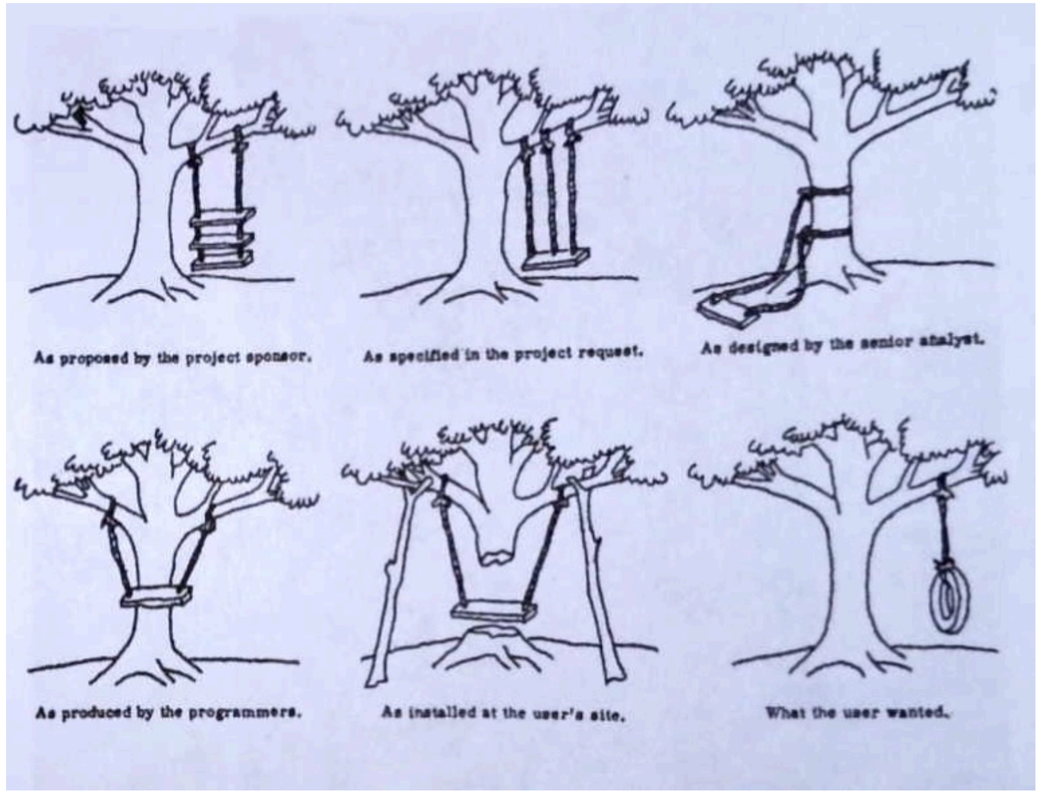
Nuisance distances.

"A Pattern Language: Towns, Buildings, Construction," C Alexander et al. pg 408-409

Why Design Patterns?

The OG meme (1975)





More of Christopher Alexander's Patterns

Domain-Specific Patterns

Every particular community will always need to do the same to supplement the general patterns from *A Pattern Language*. The patterns are:

UNIVERSITY POPULATION
OPEN UNIVERSITY
STUDENT HOUSING DISTRIBUTION
UNIVERSITY SHAPE AND DIAMETER
UNIVERSITY STREETS
LIVING LEARNING CIRCLE
FABRIC OF DEPARTMENTS
DEPARTMENTS OF 400
DEPARTMENT SPACE
LOCAL ADMINISTRATION
STUDENT COMMUNITY
SMALL STUDENT UNIONS
PARKING SPACES
CLASSROOM DISTRIBUTION
FACULTY STUDENT MIX
STUDENT WORKPLACE
REAL LEARNING IN CAFES
DEPARTMENT HEARTH

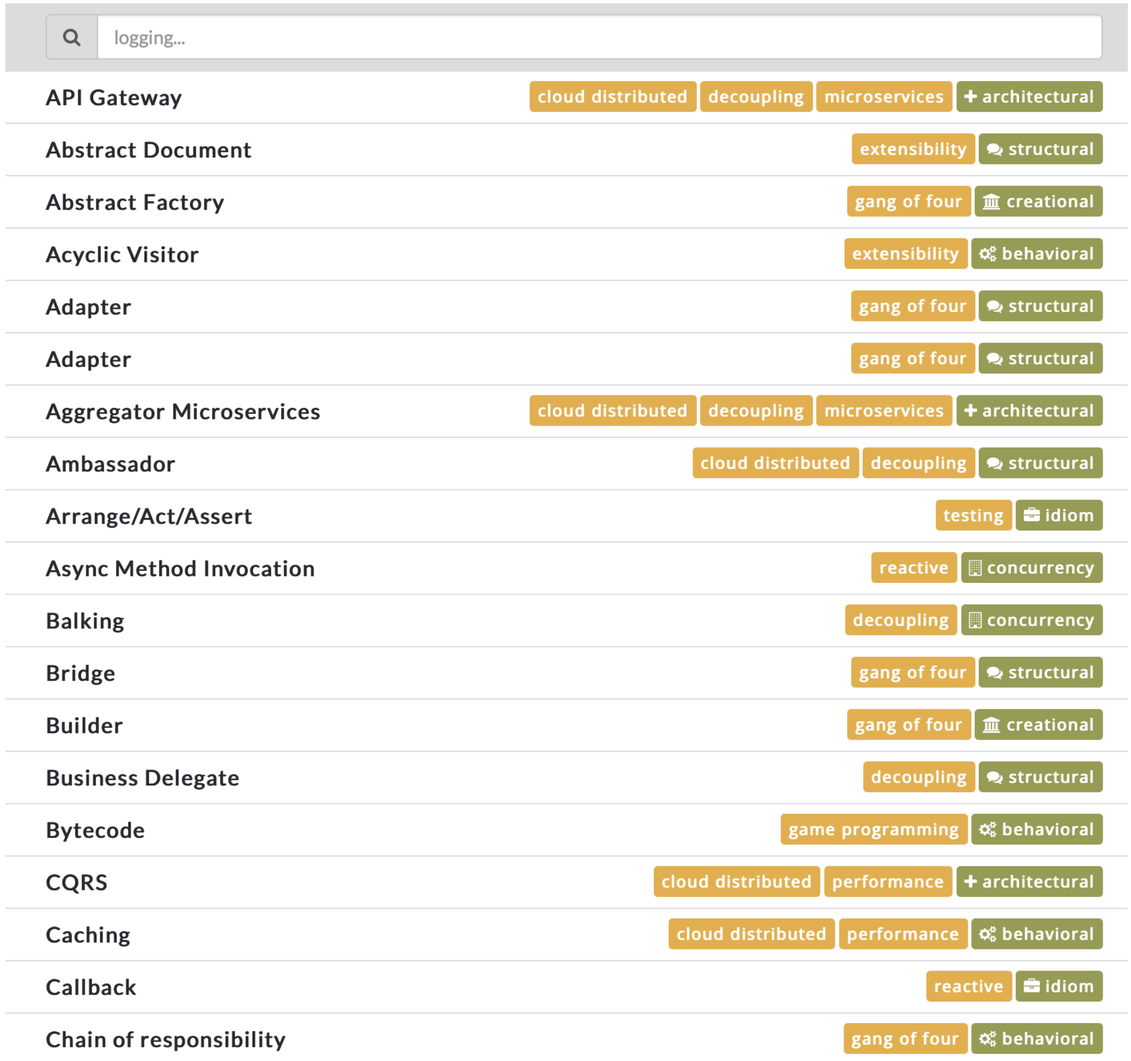
9. Living Learning Circle:
Students who want to live closely related to the university want their housing integrated with the university; yet most on-campus housing provided today is zoned off from academic departments. Therefore: Provide housing for 25 per cent of the student population within the 3000 for inner university diameter. Do not zone this housing off from academic departments…"

"The Oregon Experiment," C Alexander et al. pg 114

More (Software) Design Patterns

Just like Alexander expected more patterns to be “discovered” in architecture, same happens in code…

https://java-design-patterns.com/patterns/ : 109+ patterns

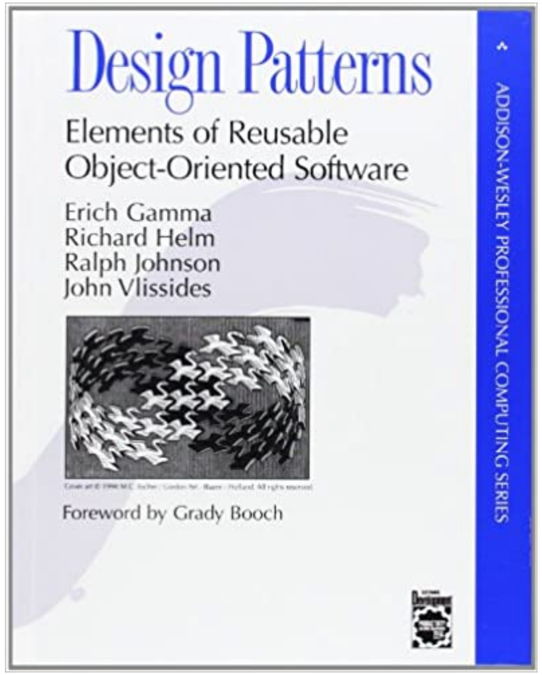


Patterns might be local to:
A language
A framework
A project

Design Patterns

Discusison: Is it about vocabulary, or is it about training?

"None of the design patterns in this book describes new or unproven designs. We have included only designs that have been applied more than once in different systems. Most of these designs have never been documented before. They are either part of the folklore of the object-oriented community or are elements of some successful object-oriented systems—neither of which is easy for novice designers to learn from."



Review: Observer Pattern

Aka Publish/Subscribe

- The object being observed (the "subject") keeps a list of the people who need to be notified when something changes.
- When a new object wants to be notified when the subject changes, it registers with ("subscribes to") with the subject

Observer

- defines an interface for objects that should be notified of changes

ConcreteObserver

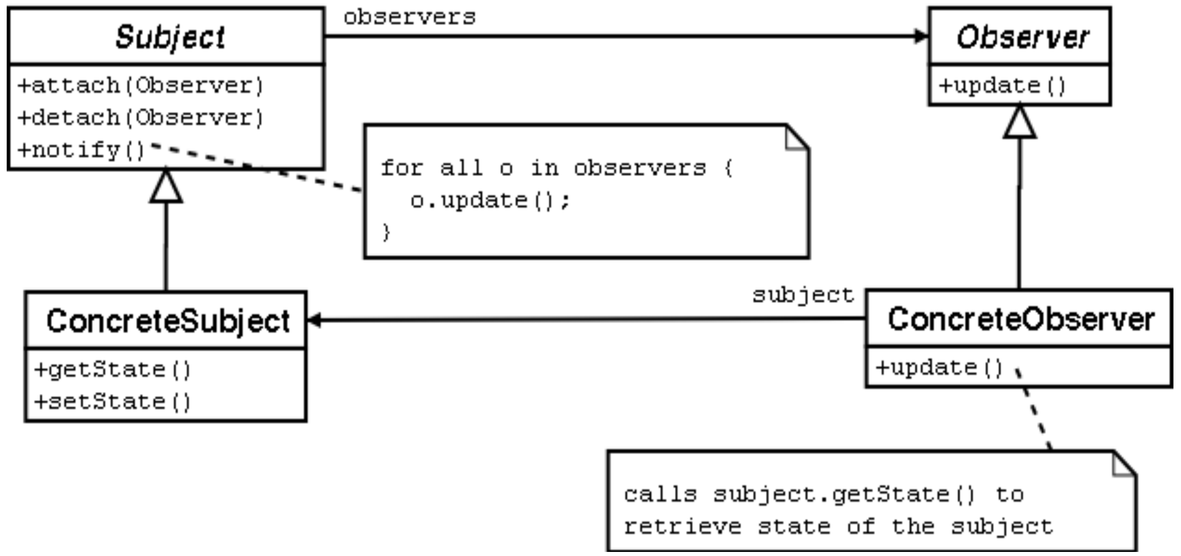
- maintains reference to an object
- stores state that should stay consistent with subject's
- implements the Observer interface to keep state consistent with the subject's

Subject

- provides an interface for attaching/ detaching observers

ConcreteSubject

- stores state of interest to observers
- sends a notification to its observers when state changes



Review: Observer Pattern

```
export interface IPublishingClock {
    // reset the tick counter
    reset(): void
    // increment the tick counter
    tick(): void
    // subscribe a new observer
    subscribe(obs:ClockObserver) : void
}

export interface ClockObserver {
    // action to take when clock ticks
    onTick(time:number):void

    // action to take when the clock resets
    onReset():void

}
```

```
class Clock implements IPublishingClock {

    // clock functionality
    private clockTime = 0
    public tick () {this.clockTime++; this.publishTickEvent()}
    public reset() {this.clockTime=0; this.publishResetEvent()}

    private observers : ClockObserver[]

    // register responds with the current time, so the observer
    // will be initialized
    public subscribe(obs:ClockObserver): void {
        this.observers.push(obs);
        obs.onTick(this.clockTime)
    }
    private publishTickEvent() {
        this.observers.forEach(obs => {obs.onTick(this.clockTime)})
    }

    private publishResetEvent() {
        this.observers.forEach(obs => {obs.onReset()})
    }
}
```

Programming Activity

Weather Station

- We have prepared some code for a weather station, with a class WeatherData that stores the current temperature, pressure and humidity, and several display classes that format the data, e.g.:

```
export default class StatisticsDisplay {
  private _maxTemp = 0;

  private _minTemp = 0;

  private _tempSum = 0;

  private _numReadings = 0;

  displayStatistics(currentData: WeatherData): void {
    this._tempSum += currentData.temperature;
    this._numReadings += 1;
    if (this._maxTemp < currentData.temperature) {
      this._maxTemp = currentData.temperature;
    }
    if (this._minTemp > currentData.temperature) {
      this._minTemp = currentData.temperature;
    }

    // eslint-disable-next-line
    console.log('Avg/max/min temperature = %f/%i/%i', this._tempSum / this._numReadings, this._maxTemp, this._minTemp);
  }
}
```

Sample output from running all of the displays:

```
Avg/max/min temperature = 80/80/0
Forecast: Improving weather on the way!
Current conditions: 80F degrees and 65% humidity
Heat Index: 82.95535063710001
```

Programming Activity

Observer Pattern

CRC card for the current class:

Class: WeatherData	
State: current temperature, humidity, pressure	

Responsibilities:
Keep track of current weather
Push changes to displays
	Collaborators:
	`StatisticsDisplay`
	`ForecastDisplay`
	`CurrentConditionsDisplay`
	`HeatIndexDisplay`

File: WeatherData.ts

```
public setMeasurements(temperature: number, humidity: number, pressure: number): void {
  this._temperature = temperature;
  this._humidity = humidity;
  this._pressure = pressure;
  this.measurementsChanged();
}

private measurementsChanged() {
  this._statisticsDisplay.displayStatistics(this);
  this._forecastDisplay.displayForecast(this);
  CurrentConditionsDisplay.displayCurrentConditions(this);
  HeatIndexDisplay.displayHeatIndex(this);
}
```

Your task: rewrite these classes to follow the observer pattern

CRC card after your changes:

Class: WeatherData	
State: current temperature, humidity, pressure	
Responsibilities: Keep track of current weather Push changes to displays	**Collaborators:** `WeatherDataObserver`

Activity is online at:
https://neu-se.github.io/CS4530-CS5500-Spring-2021/Activities/Activity2.2_Observer/

This work is licensed under a Creative Commons Attribution-ShareAlike license