Software Engineering

Part 1: Introduction

Course overview and introduction to software design

Jiwon Seo

Syllabus

- Not the one on the course registration page
- See 22 page of this slide

Learning goals

- Ability to **design** medium-scale programs
- Understanding **OO programming** concepts & design decisions
- Proficiency with basic **quality assurance** techniques for functional correctness
- Fundamentals of **concurrency and distributed systems**
- Practical skills

Meet the Instructor

- Jiwon Seo (seojiwon@hanyang.ac.kr, IT/BT 405-1)

- Research topic: Big data systems,
  Large-scale machine learning systems,
  Large-scale data analytics

2005 ~ **Python Contributor**

~ 2015. **Stanford PhD (Large-Scale Graph Analysis System)**

2014. **LinkedIn (Research Engineer)**

2015. **Pinterest (Research Engineer)**

2016~2017. **UNIST (Faculty)**

2017~ **Hanyang University (Faculty)** http://bdsa.unist.ac.kr (old homepage)

Meet the TA

TA: Bongki Cho (ite1015.staff.2017@gmail.com)

Communicating with the Staff

- Check course homepage frequently
- Use (ite1015.staff.2017@gmail.com) for administrative stuff
- Email the instructor if you need an immediate attention
- How to write emails

For programming questions, use course homepage (Q/A)

(Optional) Textbooks

- Effective Java (by Bloch, Joshua)
- Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (by Larman, Craig)
- Design Patterns Explained: A New Perspective on Object-Oriented Design (by Alan Shalloway and James Trott)
- Java Concurrency in Practice (by Goetz et al.)

Approximate grading policy

- 25% assignments
- 30% midterms
- 35% final exam
- 10% other (e.g. class participation)

This is tentative!

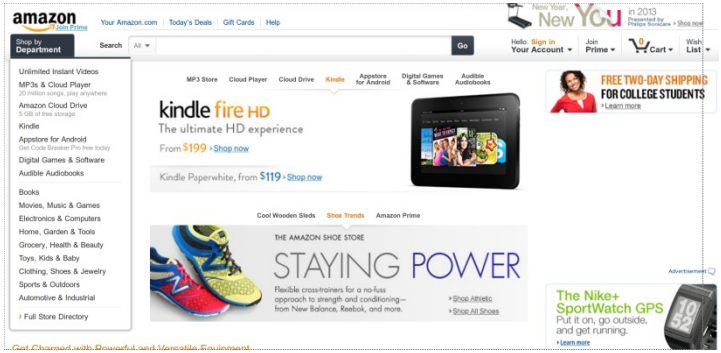
Academic Integrity

- Never (even partially) copy homeworks or projects - ask TAs or mentors for help.
- Never cheat in exams.
- Any violation of academic integrity leads to F grade.

Homework Assignment 1

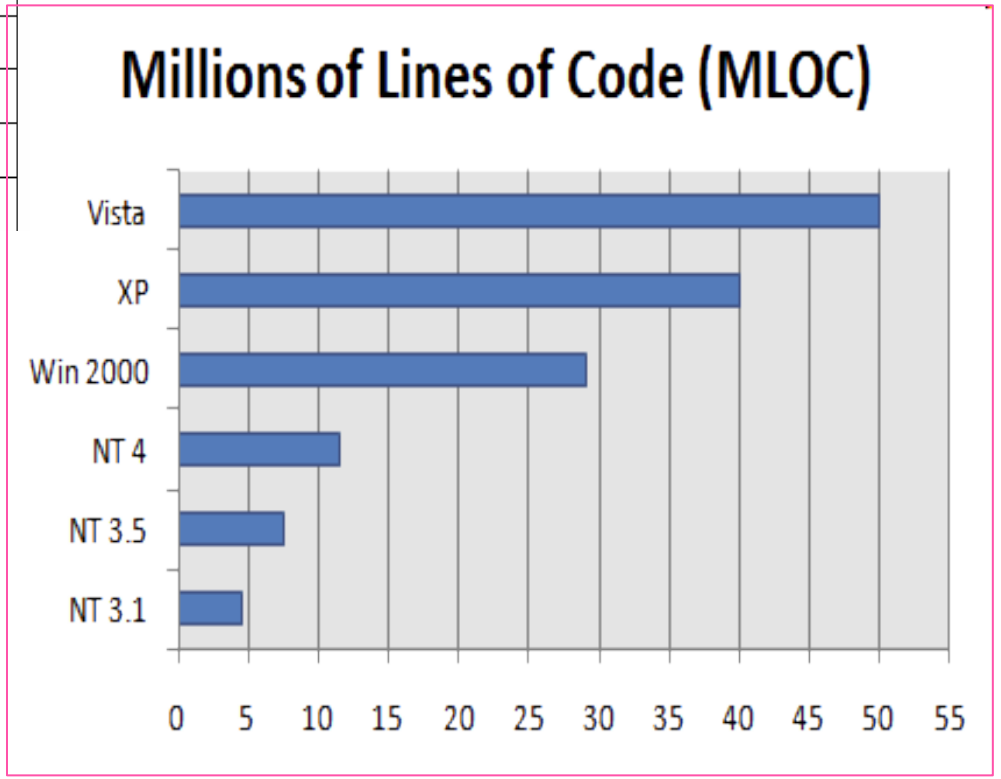
- On course homepage
- Read it today, and ask questions on Wednesday in class
- Due on Sep 14th
- Submission instruction soon (please check frequently)

Software is everywhere



Growth of code and complexity over time

n System	Year	% of Functions Performed in Software
F-4	1960	8
A-7	1964	10
F-111	1970	20
F-15	1975	35
F-16	1982	45
B-2	1990	65
F-22	2000	80



(informal reports)

COMMENTARY

Chris Murphy

Editor, InformationWeek

See more from this author

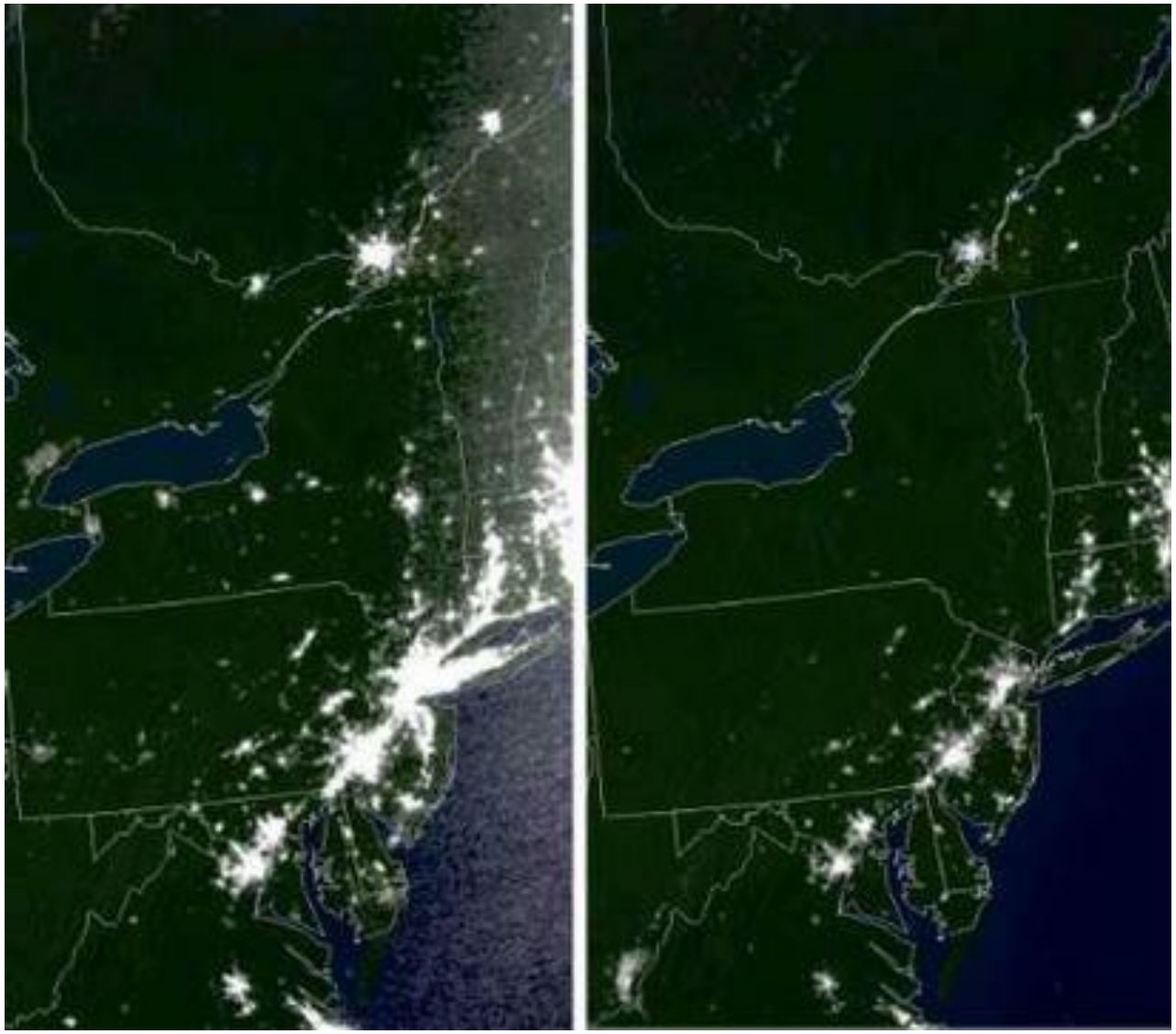
Why Ford Just Became A Software Company

Ford is upgrading its in-vehicle software on a huge scale, embracing all the customer expectations and headaches that come with the development lifecycle.

6 Comments | Chris Murphy | November 14, 2011 09:31 AM

Sometime early next year, Ford will mail USB sticks to about 250,000 owners of vehicles with its advanced touchscreen control panel. The stick will contain a major upgrade to the software for that screen. With it, Ford is breaking from a history as old as the auto industry, one in which the technology in a car essentially stayed unchanged from assembly line to junk yard.

Ford is significantly changing what a driver or passenger experiences in its cars years after they're built. And with it, Ford becomes a software company--with all the associated high customer expectations and headaches.



Normal night-time image

Blackout of 2003

The blackout's primary cause was a programming error or "bug" in the alarm system at the control room -- Wikipedia

primes

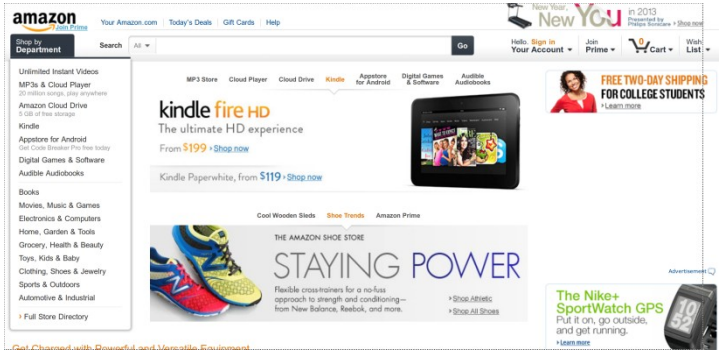
graph search

binary tree

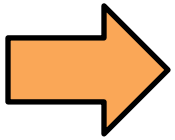
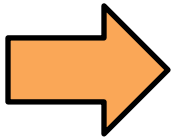
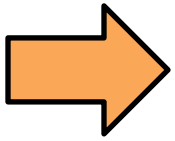
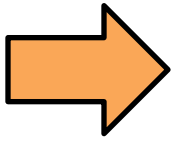
GCD

sorting

BDDs



From programs to systems

Writing algorithms, data structures from scratch		Reuse of libraries, frameworks
Functions with inputs and outputs		Asynchronous and reactive designs
Sequential and local computation		Parallel and distributed computation
Full functional specifications		Partial, composable, targeted models

Our goal: understanding both the building blocks and the design principles for construction of software systems

Themes:

1. Object Orientation
2. Software Design
3. Concurrency

Themes:

1. **Object Orientation**
2. Software Design
3. Concurrency

Objects in the real world

Object-oriented programming

- Programming based on structures that contain both data and methods

```
public class Bicycle {
  private int speed;
  private final Wheel frontWheel, rearWheel;
  private final Seat seat;
  ...

  public Bicycle(...) { ... }

  public void accelerate() {
    speed++;
   }

  public int speed() { return speed; }
}
```

Themes:

1. Object Orientation
2. **Software Design**
3. Concurrency

Semester overview

- Overview of O-O
- Introduction to **design**
  - **Design** goals, principles, patterns
- **Design**ing classes
  - **Design** for change
  - **Design** for reuse
- **Design**ing (sub)systems
  - **Design** for robustness
  - **Design** for change (cont.)
- **Design** case studies
- **Design** for large-scale reuse

- Explicit concurrency
- Distributed systems

- Crosscutting topics:
  - Modern development tools: IDEs, version control, build automation, continuous integration, static analysis
  - Modeling and specification, formal and informal
  - Functional correctness: Testing, static analysis, verification

Sorting with a configurable order, version A

```
static void sort(int[] list, boolean ascending) {
    ...
    boolean mustSwap;
    if (ascending) {
        mustSwap = list[i] < list[j];
    } else {
        mustSwap = list[i] > list[j];
    }
    ...
}
```

Sorting with a configurable order, version B

```
interface Comparator {
  boolean compare(int i, int j);
}

class AscendingComparator  implements Comparator {
  public boolean compare(int i, int j){return   i < j; }
}
class DescendingComparator implements Comparator {
  public boolean compare(int i, int j){return   i > j; }
}

static void sort(int[] list, Comparator cmp) {
  ...
  boolean mustSwap =
    cmp.compare(list[i], list[j]);
  ...
}
```

Sorting with a configurable order, version B'

```
interface Comparator{
  boolean compare(int i, int j);
}

final Comparator ASCENDING  = (i, j) -> i <  j;
final Comparator DESCENDING = (i, j) -> i >  j;

static void sort(int[] list, Comparator cmp) {
   ...
   boolean mustSwap =
      cmp.compare(list[i], list[j]);
   ...
}
```

Which version is better?

Version A:

```
static void sort(int[] list, boolean ascending) {
    ...
    boolean mustSwap;
    if (ascending) {
        mustSwap = list[i] < list[j];
    } else {
        mustSwap = list[i] > list[j];
    }
    ...
}
```

Version B':

```
interface Comparator{
  boolean compare(int i, int j);
}
final Comparator ASCENDING = (i, j) -> i < j;
final Comparator DESCENDING = (i, j) -> i >j;

static void sort(int[] list, Comparator cmp) {
  ...
  boolean mustSwap =
    cmp.compare(list[i], list[j]);
  ...
}
```

It depends?

Software engineering is the branch of computer science that creates **practical, cost-effective solutions** to computing and information processing problems, preferably by applying scientific knowledge, developing software systems in the service of mankind.

Software engineering entails making **decisions** under constraints of limited time, knowledge, and resources…

Engineering quality resides in engineering **judgment**…

Quality of the software product depends on the engineer's **faithfulness to the engineered artifact**…

Engineering requires reconciling **conflicting constraints**…

Engineering skills improve as a result of careful systematic **reflection** on experience…

Costs and time constraints matter, **not just capability**…

Software Engineering for the 21st Century: A basis for rethinking the curriculum Manifesto, CMU-ISRI-05-108

Which version is better?

Version A:

```
static void sort(int[] list, boolean ascending) {
    ...
    boolean mustSwap;
    if (ascending) {
        mustSwap = list[i] < list[j];
    } else {
        mustSwap = list[i] > list[j];
    }
    ...
}
```

Version B':

```
interface Comparator{
  boolean compare(int i, int j);
}
final Comparator ASCENDING = (i, j) -> i < j;
final Comparator DESCENDING = (i, j) -> i >j;

static void sort(int[] list, Comparator cmp) {
  ...
  boolean mustSwap =
    cmp.compare(list[i], list[j]);
  ...
}
```

Goal of software design

- For each desired program behavior there are infinitely many programs
  - What are the differences between the variants?
  - Which variant should we choose?
  - How can we synthesize a variant with desired properties?

A typical Intro CS design process

1. Discuss software that needs to be written
2. Write some code
3. Test the code to identify the defects
4. Debug to find causes of defects
5. Fix the defects
6. If not done, return to step 1

Metrics of software quality

Source: Braude, Bernstein, Software Engineering. Wiley 2011

- Sufficiency / functional correctness
  - Fails to implement the specifications …Satisfies all of the specifications
- Robustness
  - Will crash on any anomalous event …Recovers from all anomalous events
- Flexibility
  - Must be replaced entirely if spec changes …Easily adaptable to changes
- Reusability
  - Cannot be used in another application …Usable without modification
- Efficiency
  - Fails to satisfy speed or storage requirement …satisfies requirements
- Scalability
  - Cannot be used as the basis of a larger version …is an outstanding basis…
- Security
  - Security not accounted for at all …No manner of breaching security is known

Design challenges/goal

Better software design

- Think before coding
- Consider non-functional quality attributes
  - Maintainability, extensibility, performance, …
- Propose, consider design alternatives
  - Make explicit design decisions

Using a design process

- A design process organizes your work
- A design process structures your understanding
- A design process facilitates communication

Preview: Design goals, principles, and patterns

- ***Design goals*** enable evaluation of designs
  - e.g. maintainability, reusability, scalability
- ***Design principles*** are heuristics that describe best practices
  - e.g. high correspondence to real-world concepts
- ***Design patterns*** codify repeated experiences, common solutions
  - e.g. template method pattern

Themes:

1. Object Orientation
2. Software Design
3. **Concurrency**

Concurrency

- Simply: doing more than one thing at a time

Summary: Course themes

- Object-oriented programming
- Code-level design
- Analysis and modeling
- Concurrency and distributed systems

Questions?

License of Course Material

The slides are derived from CS 15-214 (Principles of Software Construction) in CMU.