Principles of Software Construction: Objects, Design, and Concurrency

Introduction, Overview, and Syllabus

Christian Kästner **Vincent Hellendoorn**

Principles of Software Construction: Objects, Design, and Concurrency

Introduction, Overview, and Syllabus

Christian Kästner **Vincent Hellendoorn**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

A Few Questions

How many lines of code behind twitter.com?

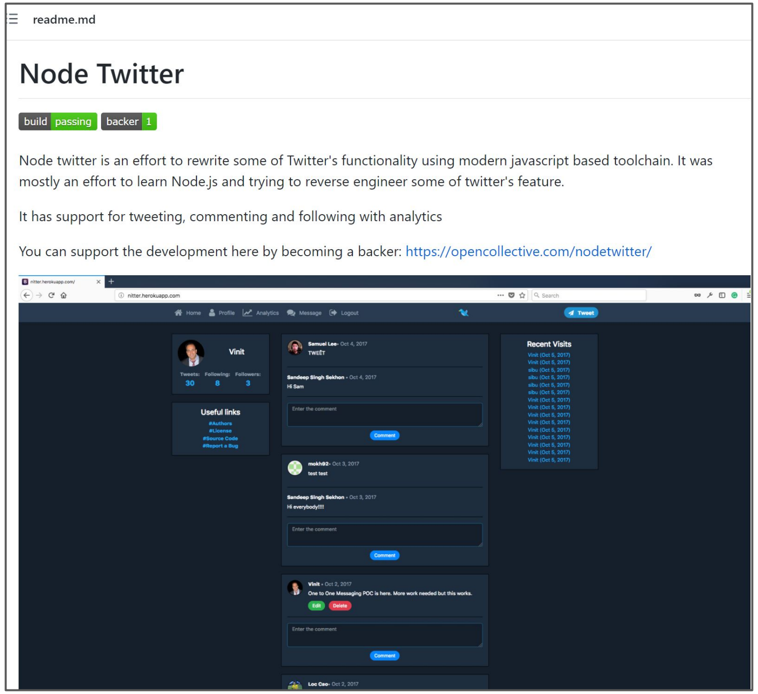
> A few million, maybe many

How many LOC to build an okay Twitter replica?

> A few 10K

How many LOC to run a Twitter replica?

> A few



readme.md

Node Twitter

build passing backer 1

Node twitter is an effort to rewrite some of Twitter's functionality using modern javascript based toolchain. It was mostly an effort to learn Node.js and trying to reverse engineer some of twitter's feature.

It has support for tweeting, commenting and following with analytics

You can support the development here by becoming a backer: https://opencollective.com/nodetwitter/

https://github.com/vinitkumar/node-twitter

Modern Software Engineering

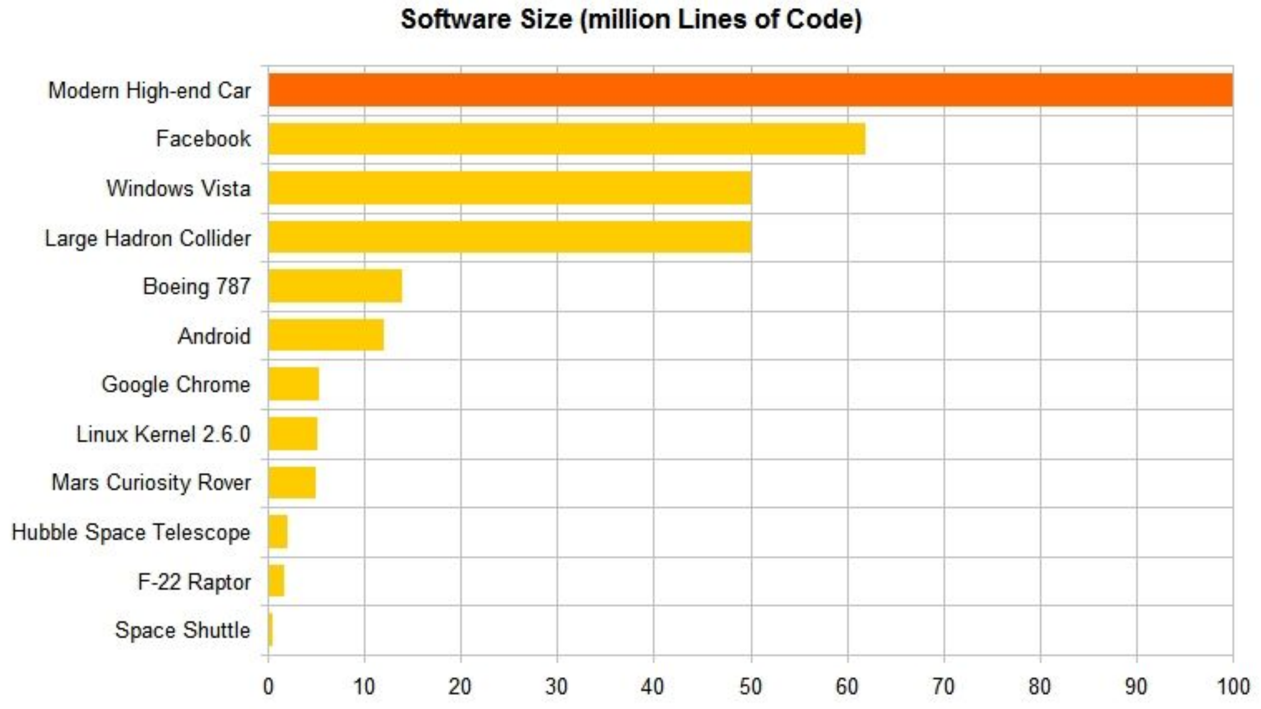
So, why learn "principles of software construction"?

- You don't want to build Twitter
  - But you can reuse the components
- "A few lines of code" does not mean easy
  - You'll need to become fluent in using new systems

<u>There are many ways to compose applications</u>

- An engineer knows the pieces **and** how to put them together.

Welcome to the era of "big code"

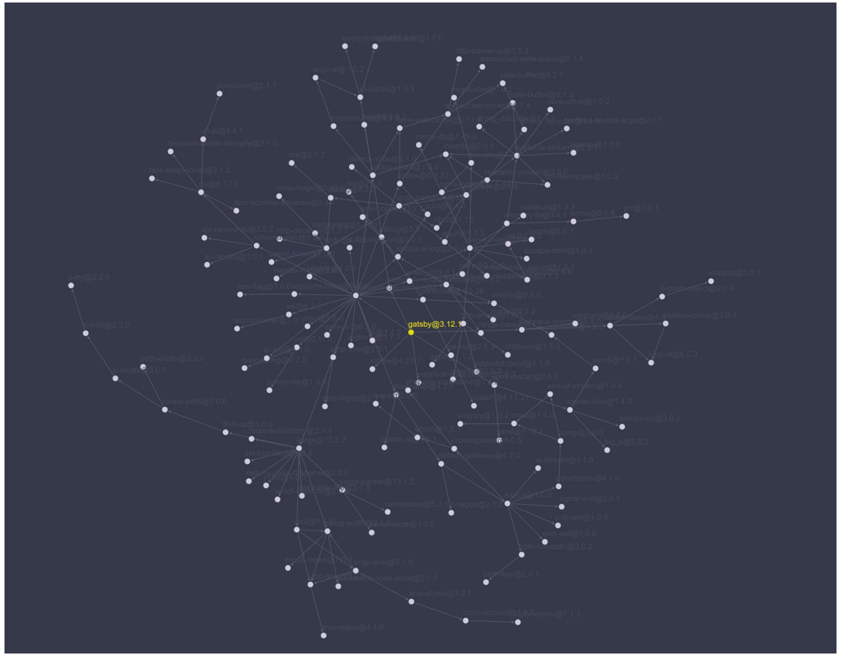


(informal reports)

Modern Software Engineering

Nobody wants to write a million lines of code.

- Instead, you use libraries
  - E.g., import Android => +12M LOC
  - You don’t write most of the code you use
    - And why would you want to?
- And your libraries use libraries
  - Et cetera
  - https://npm.anvaka.com/#/view/2d/gatsby

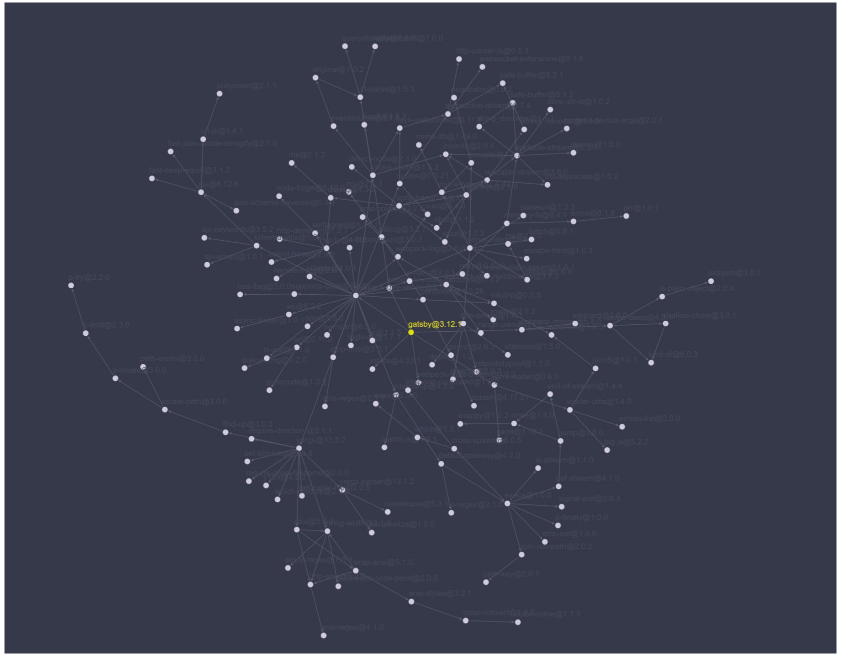


Modern Software Engineering

Nobody wants to write a million lines of code.

- Instead, you use libraries
  - E.g., import Android => +12M LOC
  - You don’t write most of the code you use
    - And why would you want to?
- And your libraries use libraries
  - Et cetera
  - https://npm.anvaka.com/#/view/2d/gatsby

What are the implications?



primes

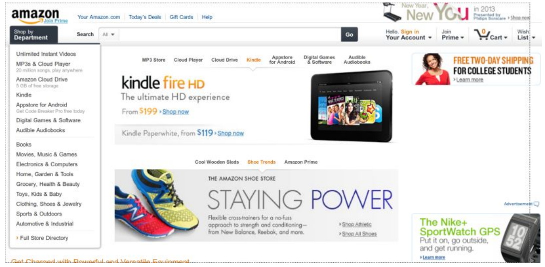
graph search

binary tree

GCD

sorting

BDDs



Equipment of a Modern Programmer

Less emphasis on: *(though not unimportant!)*

- Clever algorithmics
- Low-level code (kernels, drivers)
- Writing common components (command-line parsers, HTML)

Equipment of a Modern Programmer

More emphasis on:

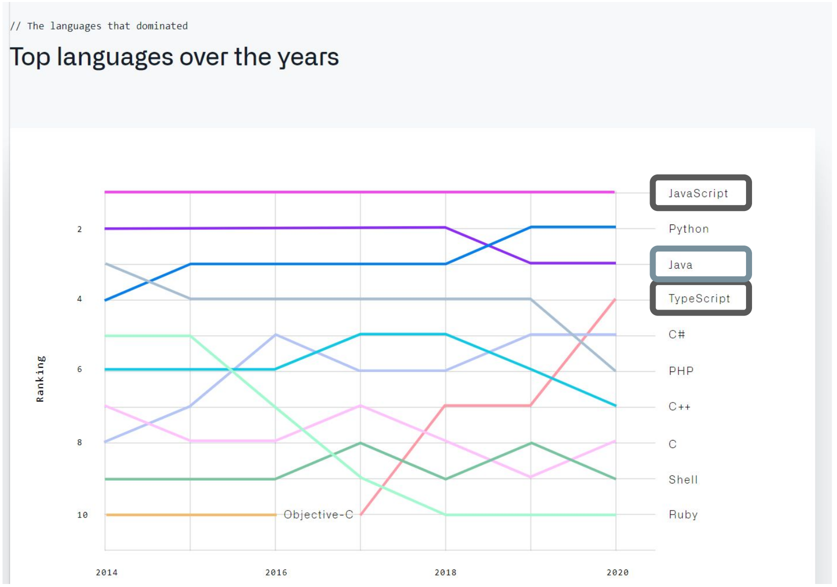
- Using APIs, libraries (hw1)
- Quality assurance (hw2)
- Design for reuse, extension (hw3+)
- Flexibility in ecosystems (all)

Flexibility & Ecosystems

Flexibility is perhaps the key skill, besides good design.

In this course:

- learn to choose & use libraries
- Adopting new tools, troubleshooting
- Also, Java vs. JavaScript/TypeScript



From Programs to Applications and 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 also the **design principles** for construction of software systems **at scale**

Outcomes, hopefully

You'll learn to be:

- An architect, approaching programming as design
  - This is the only way to scale up to larger systems
  - You'll learn a rich vocabulary, of both components and their combinations
- An omniglot, able to pick up new languages and libraries
  - Because you know the underlying concepts
  - And you've had plenty of practice reading documentation, debugging setups
- An engineer, safeguarding the quality of your programs
  - You'll get dextrous at testing, be explicit about specification
  - You'll know the tools that improve your work

Applications in this Course (Homeworks)

Flashcard learning app (command line)

Static website generator / CMS (command line application)

Board game with web interface (could also be a mobile app)

Data analysis and visualization tool (desktop/web application)

Principles of Software Construction: $\boxed{\text{Objects,}}$ Design, and Concurrency

Introduction, Overview, and Syllabus

Christian Kästner Vincent Hellendoorn

Objects in the real world

Object-oriented programming

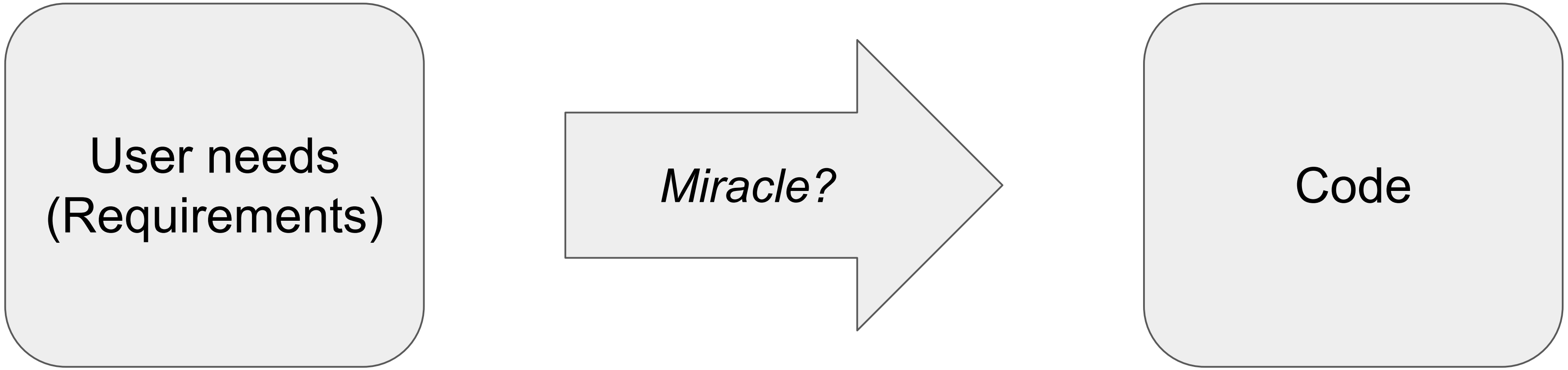
Programming based on structures that contain both data and methods

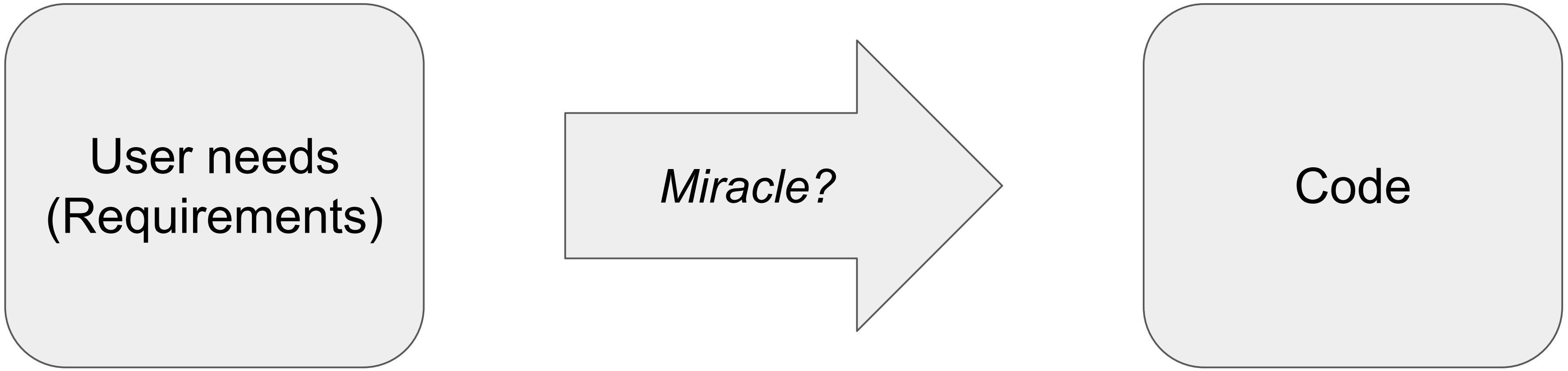
```
public class Bicycle {
  private int speed;
  private final Wheel frontWheel, rearWheel;
  private final Seat seat;
  …
  public Bicycle(…) { … }

  public void accelerate() {
    speed++;
  }

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

Principles of Software Construction: Objects, $\boxed{\text{Design,}}$ and Concurrency

Introduction, Overview, and Syllabus

Christian Kästner Vincent Hellendoorn


User needs (Requirements)
Miracle?
Code



Maintainable?
Testable?
Extensible?
Scalable?
Robust? ...

Semester overview

- Introduction to Object-Oriented Programming
- Introduction to **design**
  - **Design** goals, principles, patterns
- **Design**ing objects/classes
  - **Design** for change
  - **Design** for reuse
- **Design**ing (sub)systems
  - **Design** for robustness
  - **Design** for change (cont.)
- **Design** for large-scale reuse

Crosscutting topics:

- Building on libraries and frameworks
- Building libraries and frameworks
- Modern development tools: IDEs, version control, refactoring, build and test automation, static analysis
- Testing, testing, testing
- Concurrency basics

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 Order {
  boolean lessThan(int i, int j);
}

class AscendingOrder implements Order {
  public boolean lessThan(int i, int j) { return i < j; }
}
class DescendingOrder implements Order {
  public boolean lessThan(int i, int j) { return i > j; }
}

static void sort(int[] list, Order order) {
  …
  boolean mustSwap =
    order.lessThan(list[j], list[i]);
  …
```

Sorting with a configurable order, version B'

```
const ASC = function(i: number, j: number): boolean {
    return i < j;
}
const DESC = function(i: number, j: number): boolean {
    return i > j;
}

function sort(
    list: number[],
    order: (number, number) => boolean) {
  …
  boolean mustSwap = order(list[j], list[i]);
  …
}
> sort(list, ASC);
```

Which version is better?

Version A:

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

```
interface Order {
  boolean lessThan(int i, int j);
}
class AscendingOrder implements Order {
  public boolean lessThan(int i, int j) { return i < j;
}
class DescendingOrder implements Order {
  public boolean lessThan(int i, int j) { return i > j;
}

static void sort(int[] list, Order order) {
  …
  boolean mustSwap =
    order.lessThan(list[j], list[i]);
  …
```

Version B':

it depends

it depends

Depends on what?
What are scenarios?
What are tradeoffs?

it depends

Depends on what?
What are scenarios?
What are tradeoffs?

In this specific case, what would you recommend?
(Engineering judgement)

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** is the branch of computer science that creates practical, cost-effective solutions to computing and information processing problems, preferentially 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. [

Goal of software design

- Think before coding
- 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?
- Consider qualities: Maintainability, extensibility, performance, …
- Make explicit design decisions

Tradeoffs?

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

```
interface Order {
  boolean lessThan(int i, int j);
}
class AscendingOrder implements Order {
  public boolean lessThan(int i, int j) { return i < j;
}
class DescendingOrder implements Order {
  public boolean lessThan(int i, int j) { return i > j;
}

static void sort(int[] list, Order order) {
  …
  boolean mustSwap =
    order.lessThan(list[j], list[i]);
  …
```

Some qualities of interest, i.e., *design goals*

Functional correctness	Adherence of implementation to the specifications
Robustness	Ability to handle anomalous events
Flexibility	Ability to accommodate changes in specifications
Reusability	Ability to be reused in another application
Efficiency	Satisfaction of speed and storage requirements
Scalability	Ability to serve as the basis of a larger version of the application
Security	Level of consideration of application security

Source: Braude, Bernstein, Software Engineering. Wiley 2011

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

Better software design

- Think before coding: broadly consider 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

Software Engineering at CMU

- 17-214: “Code-level” design
  - extensibility, reuse, concurrency, functional correctness, medium-size to large programs
- 17-313: “Human aspects” of software development
  - requirements, team work, balancing qualities, scheduling, costs, risks, business models
- 17-413 Practicum, Seminar, Internship
- SE electives: SE4Startups, Program Analysis, SE4ML
- Various master-level courses on requirements, architecture, software analysis, etc
- SE Minor/Concentration: http://isri.cmu.edu/education/undergrad/

This is not a Java course

This is not a Java course

but you will write a lot of Java/JavaScript code

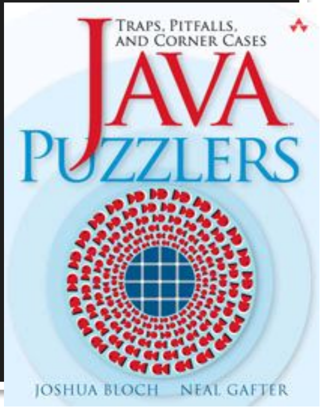
```java
int a = 010 + 3;
System.out.println("A" + a);
```

```java
int a = 010 + 3;
System.out.println("A" + a);
```

```javascript
const a = 010 + 3;
console.log("A" + a);
```

```java
int a = 010 + 3;
System.out.println("A" + a);
```

```javascript
const a = 010 + 3;
console.log("A" + a);
```



Java + JavaScript / TypeScript

Focus on design concepts, not programming language

Language proficiency through practice and homeworks

Lectures show examples in pseudo code, Java, JavaScript, TypeScript, and other languages

Pick Java or TypeScript for homeworks

```
int a = 010 + 3;
System.out.println("A"
```

```
const a = 010 + 3;
console.log("A" + a);
```

Java **OR** TypeScript/JavaScript

Your choice!

Pick in each homework assignment

- Most students will use one language for all assignments

Each recitation focuses on tools/examples in one language

- tentatively: A, C, E for Java - B, D, F for TypeScript

COURSE ORGANIZATION

SENIOR CENTER
WEAR A MASK
WASH YOUR HANDS
SOCIAL DISTANCE
STAY SAFE
COME JOIN US

Trying to get back to normal with … *gestures widely* everything

Talk to us about concerns and accommodations

Disclaimer:
This semester, we are changing a lot in this course.
Some things will go wrong.
Have patience with us.
Give us feedback.

Course materials

Course website (syllabus, slides, calendar): https://cmu-17-214.github.io/f2021/

Discussions, questions, announcements: Piazza

Assignments, readings, and grades: Canvas and Gradescope

Homework submission: GitHub (signup instructions in assignment) and other tools

Course preconditions

- 15-122 or equivalent: Basic programming skills in any language, algorithms and data structures (lists, graphs, sorting, binary search)
- 21-127 or equivalent: Basic discrete math concepts, logic

Course staff

- Christian Kästner

  kaestner@cs.cmu.edu, TCS 345

- Vincent Hellendoorn

  vhellendoorn@cmu.edu, TCS 320

- Teaching assistants:

  Esther, Katrina, Kevin, Jeff, Olivia, Sean, Sophie, Ye, and Zhifeng

Course meetings

- Lectures: Tuesday and Thursday 11:50 – 1:10pm here :)
- Recitations: Wednesdays 9:30 - … - 3:20pm
  - Preparing for homeworks, hands-on practice, supplementary material
  - Starting tomorrow! (version control and git -- relevant for HW1)
- Office hours: see course web page

Recitation attendance is required

Homework & Exams

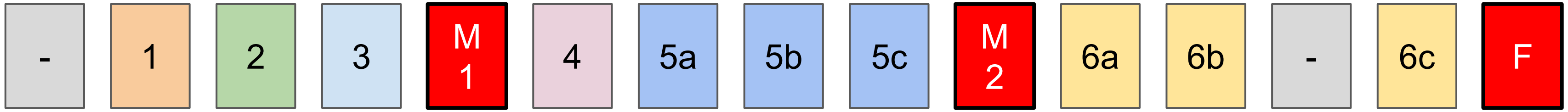
6 homeworks, 4 small + 2 large (with milestones), 1000 points total

> (1) intro, (2) testing, (3) first design, (4) fixing design,
> (5) extensibility + GUI, (6) framework and API design

Homeworks and milestones usually due Sunday night, see course calendar

Homework 1 will be released Thursday, due Sep 12

Two midterms + final

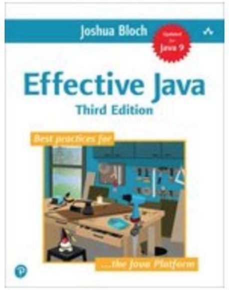


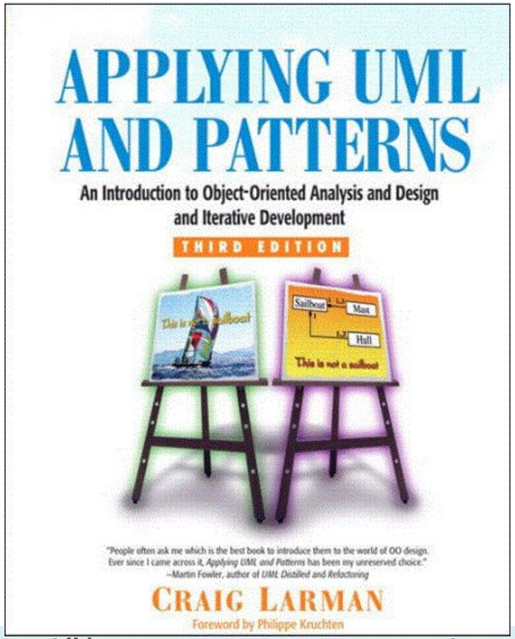
Late day policy

- See syllabus on course web page for details
- 2 possible late days per deadline (some exceptions will be announced)
    - 5 total free late days for semester (+ separate 2 late days for assignments done in pairs)
    - 10% penalty per day after free late days are used
    - but we won’t accept work 3 days late
- Extreme circumstances – talk to us

Textbooks

- Craig Larman. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development**. 3rd Edition. Prentice Hall. 2004. ISBN 0-13-148906-2
- Joshua Bloch. Effective Java, Third Edition. Addison-Wesley, ISBN 978-0-13-468599-1.
- Selective other readings throughout the semester
- Occasional in-class reading quizzes after reading assignment due
- Electronic version available through CMU library




APPLYING UML AND PATTERNS
An Introduction to Object-Oriented Analysis and Design and Iterative Development
THIRD EDITION
CRAIG LARMAN
Foreword by Philippe Kruchten

Approximate grading policy

- 50% assignments
- 20% midterms (2 x 10% each)
- 20% final exam
- 10% quizzes and participation

This course does not have a fixed letter grade policy; i.e., the final letter grades will not be A=90-100%, B=80-90%, etc.

Collaboration policy

- See course web page for details!
- We expect your work to be your own
- Do not release your solutions (not even after end of semester)
- Ask if you have any questions
- If you are feeling desperate, please reach out to us
    - Always turn in any work you've completed before the deadline
- We run cheating detection tools. Trust us, academic integrity meetings are painful for everybody

10% quizzes and participation

- Recitation participation counts toward your participation grade
- Lecture has in-class quizzes

The key to your success in this course is your regular, engagement with course activities, staff, and other students

Summary

- Software engineering requires decisions, judgment
- Good design follows a process
- You will get lots of practice in 17-214!