Principles of Software Construction: Objects, Design, and Concurrency

The Last One: Locking Back & Looking Forward

Claire Le Goues Bogdan Vasilescu

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Looking Back at the Semester

Principles of Software Construction: Objects, Design, and Concurrency

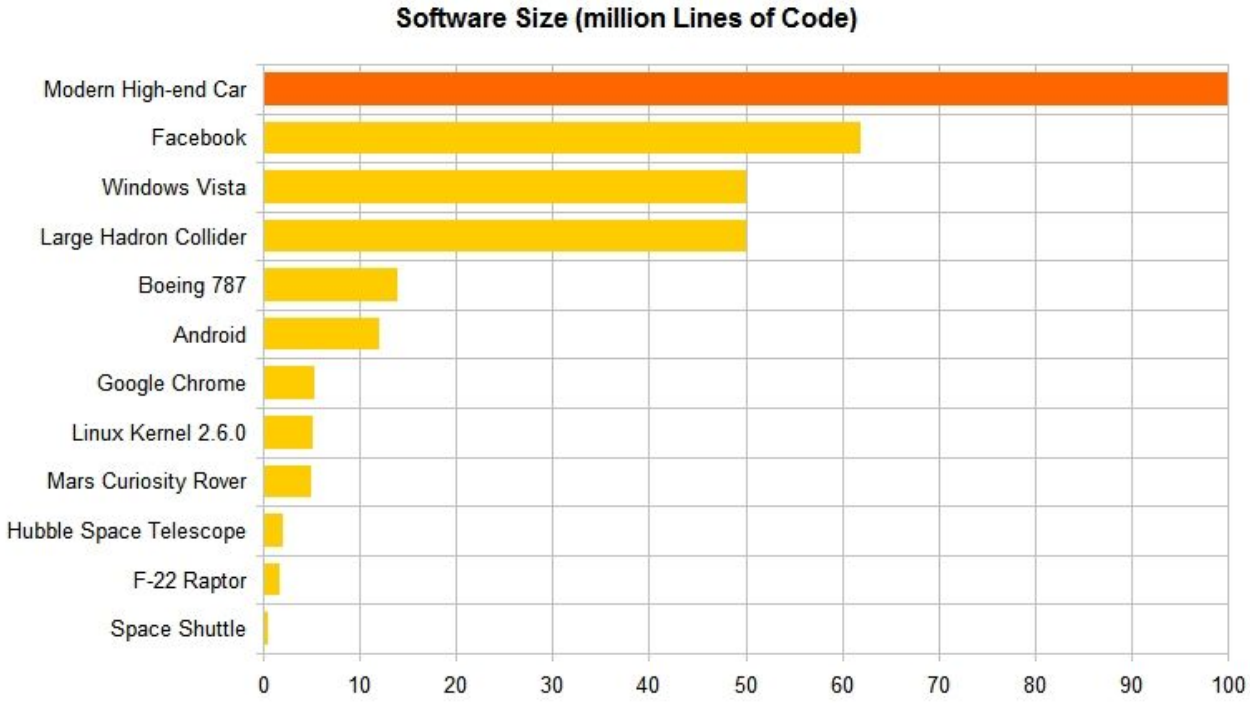
Introduction, Overview, and Syllabus

Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

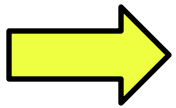
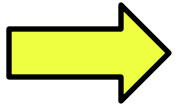
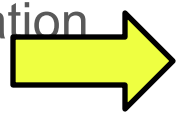
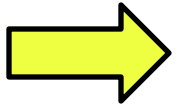
institute for SOFTWARE RESEARCH

Welcome to the era of “big code”



(informal reports)

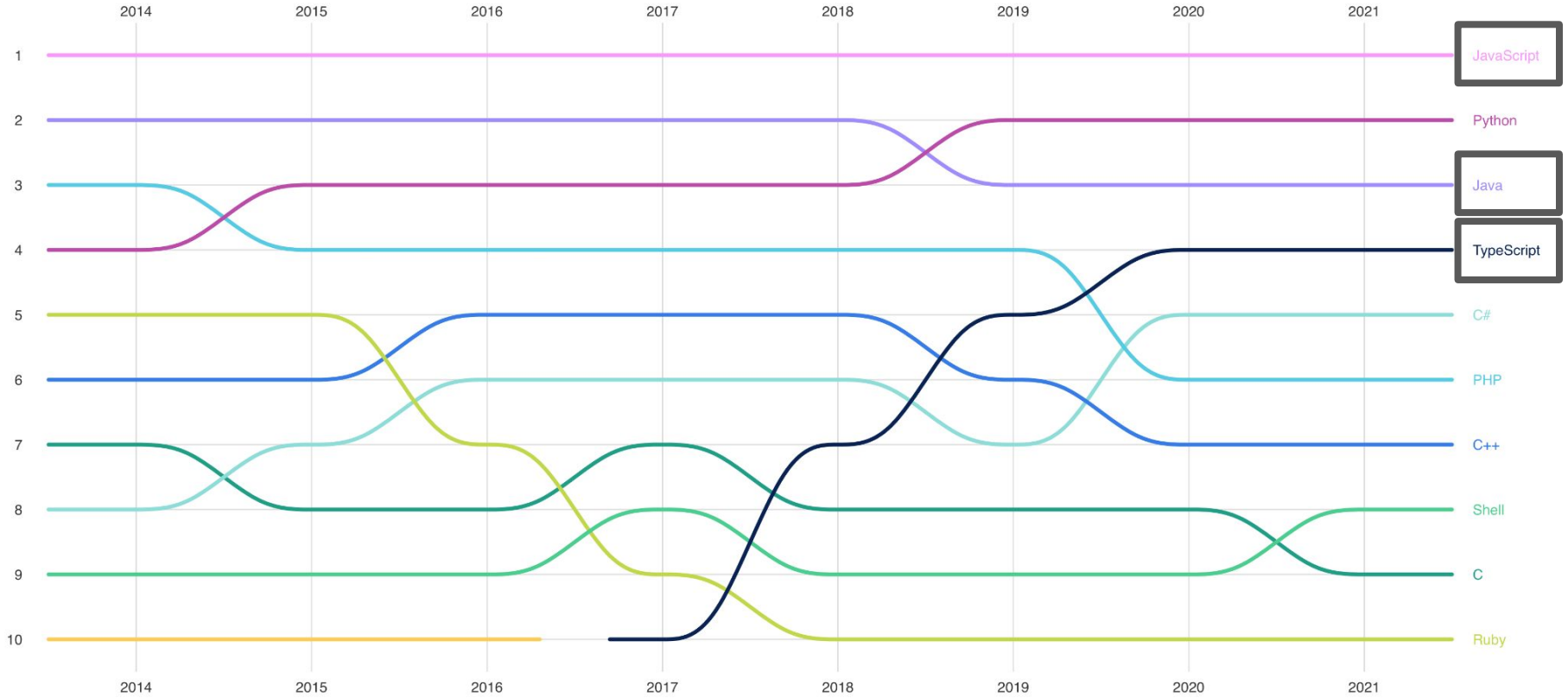
From Programs to Applications and Systems

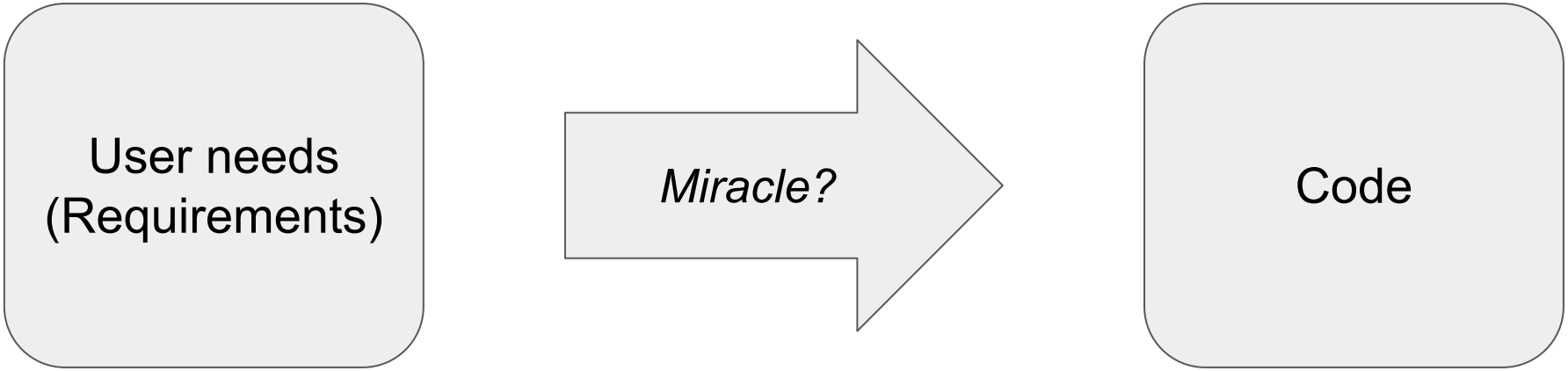
Writing algorithms, data structures from scratch		Reuse of libraries, frameworks
Functions with inputs and outputs		Asynchronous and reactive designs
Sequential and local computation		Parallel and distributed computation
Full functional specifications		Partial, composable, targeted models

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

Top languages over the years

2021 GitHub State of the Octoverse report





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

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
   }
   …
}
```

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]);
  …
```

it depends

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

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

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

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

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

Talk to us about concerns and accommodations

Principles of Software Construction (Design for change, class level)

Starting with Objects (dynamic dispatch, encapsulation, entry points)

Claire Le Goues Bogdan Vasilescu

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	**Subtype Polymorphism** ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	**Information Hiding**, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, Design Patterns, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

This is C code!

Data structures and procedures

```
struct point {
   int    x;
   int    y;
 };
void movePoint(struct point p, int deltax, int deltay) { p.x = …; }


int main() {
    struct point p = { 1, 3 };
    int deltaX = 5;
    movePoint(p, 0, deltaX);
    ...
}
```

Yellow background is Java, Black is Typescript

Interfaces and Objects in Ja

```
interface Counter {
    int get();
    int add(int y);
    void inc();
}
Counter obj = new Counter() {
    int v = 1;
    public int get() { return this.v; }
    public int add(int y) { return this.v + y; }
    public void inc() { this.v++; }
};

System.out.println(obj.add(obj.get()));
// 2
```

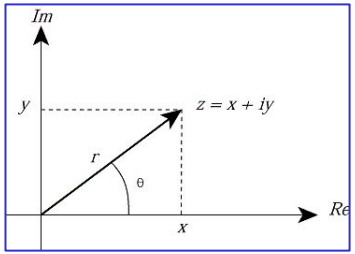
```
interface Counter {
    v: number;
    inc(): void;
    get(): number;
    add(y: number): number;
}
const obj: Counter = {
    v: 1,
    inc: function() { this.v++; },
    get: function() { return this.v; },
    add: function(y) { return this.v + y; }
}
```

This uses anonymous classes to create an object without a class. This isn't very common, it just looks a lot like the TS.

This is Java code!

Multiple Implementations of Interface

```
interface Point {
    int getX();
    int getY();
}
class PolarPoint implements Point {
    double len, angle;
    PolarPoint(double len, double angle)
        {this.len=len; this.angle=angle;}
    int getX() { return this.len * cos(this.angle);}
    int getY() { return this.len * sin(this.angle); }
    double getAngle() {…}
}
Point p = new PolarPoint(5, .245);
```



Left is Java, right is Typescript

How to hide information?

```java
class CartesianPoint {
    int x,y;
    Point(int x, int y) {
        this.x=x;
        this.y=y;
    }
    int getX() { return this.x; }
    int getY() { return this.y; }
    int helper_getAngle();
}
```

```typescript
const point = {
    x: 1, y: 0,
    getX: function() {…}
    helper_getAngle:
        function() {…}
}
```

Starting a Program

This is Typescript code!

Typescript compiles to Javascript, by the way. There are several ways to run it.

Objects do not do anything on their own, they wait for method calls

Every program needs a starting point, or waits for events

```
// start with: node file.js
function createPrinter() {
    return {
        print: function() { console.log("hi"); }
    }
}
const printer = createPrinter();
printer.print()
// hi
```

Defining interfaces, functions, classes

Starting:
Creating objects and calling methods

Principles of Software Construction: Objects, Design, and Concurrency

IDEs, Build system, Continuous Integration, Libraries

Bogdan Vasilescu **Claire Le Goues**

Carnegie Mellon University
School of Computer Science

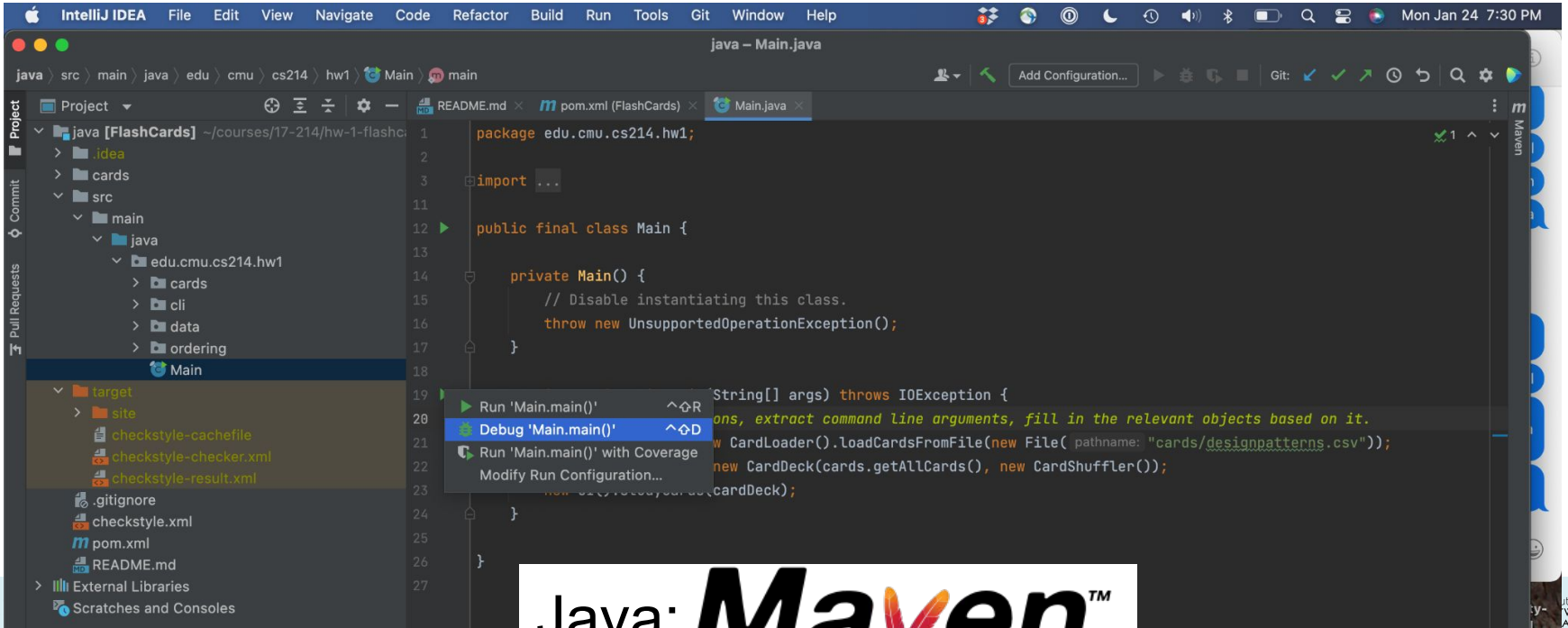
institute for SOFTWARE RESEARCH

Abstraction, Reuse, and Programming Tools

- For each in {**IDE**, Build systems, libraries, CI}:
  - What is it today?
  - **What is under the hood?**
- What is next?

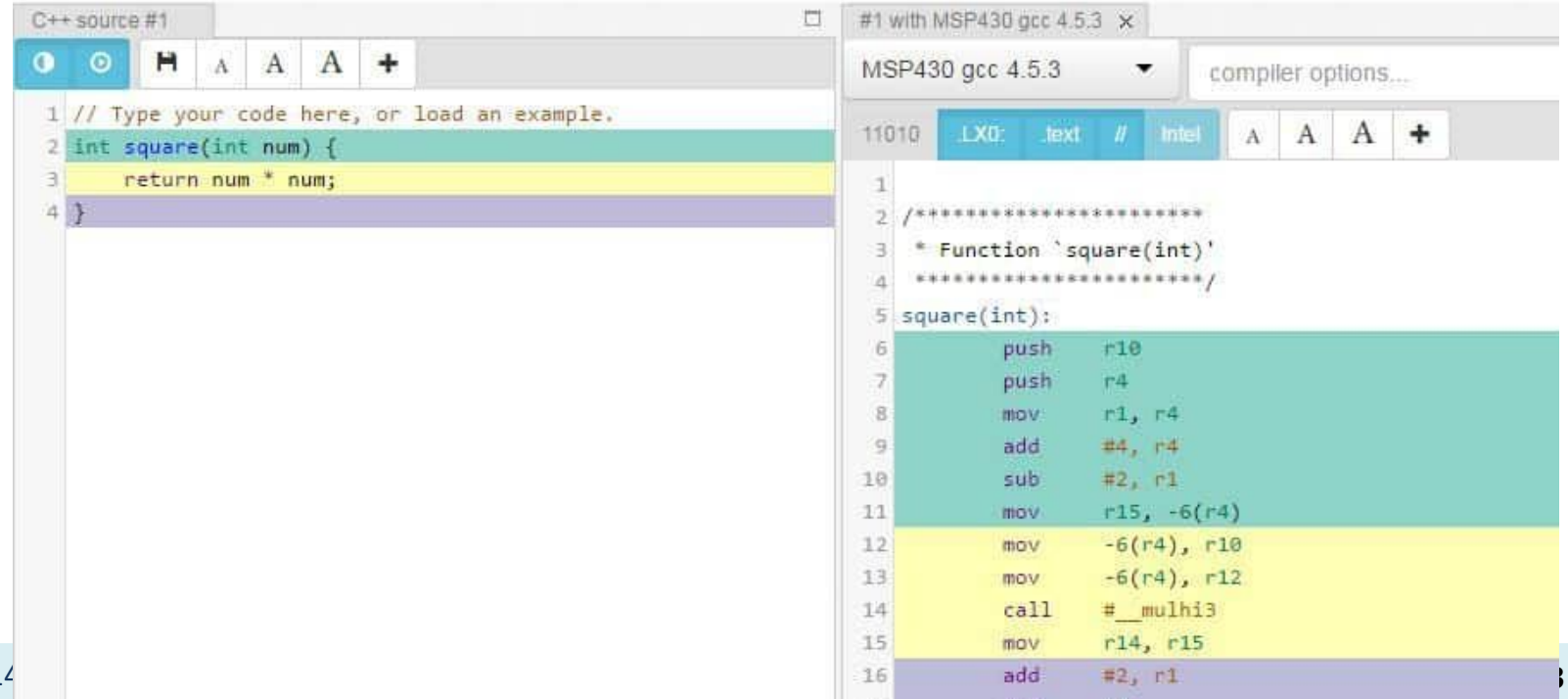
Under the Hood: IDEs

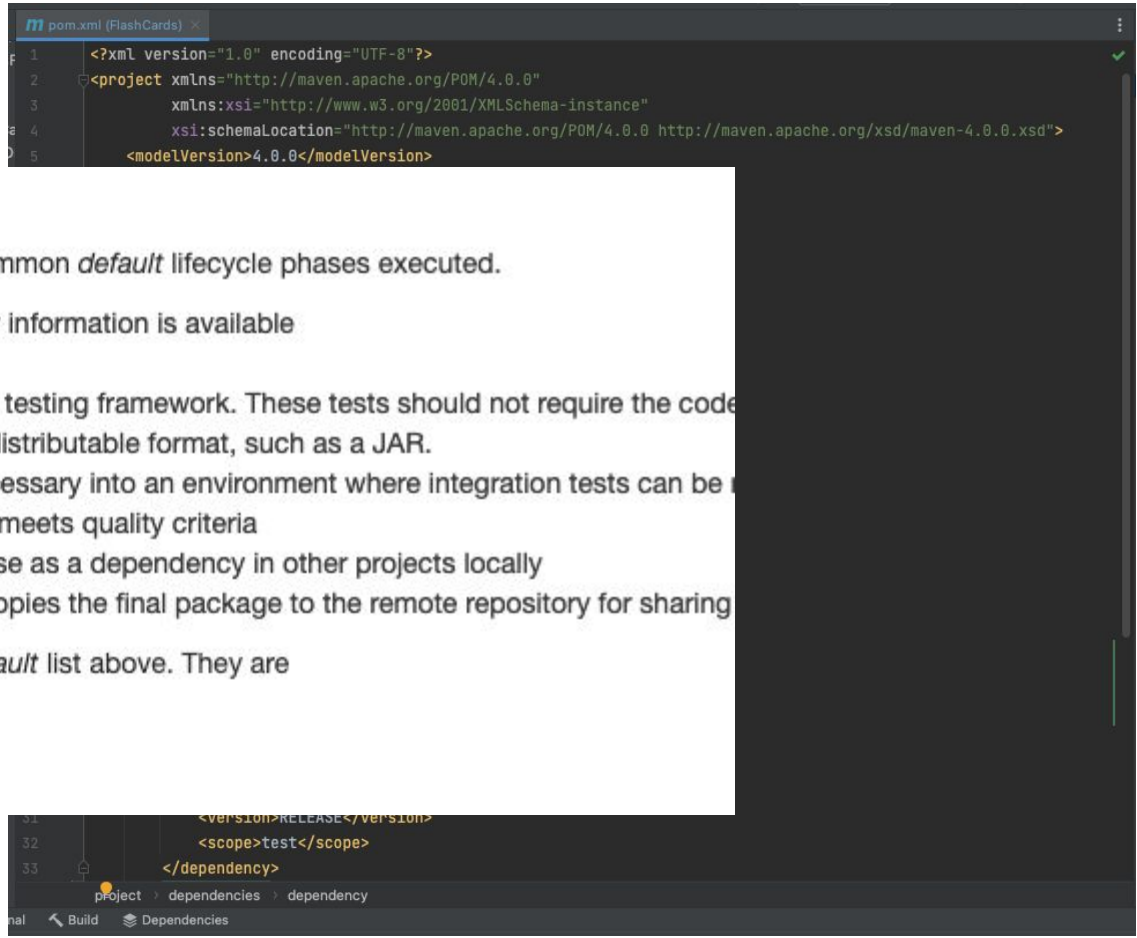
Automate common programming actions, like debugging, which is often the default mode when you run in the IDE (like in VSCode)



Quick overview of today's toolchain: Build Systems

How does this happen?





Maven Phases

Although hardly a comprehensive list, these are the most common *default* lifecycle phases executed.

- **validate**: validate the project is correct and all necessary information is available
- **compile**: compile the source code of the project
- **test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code
- **package**: take the compiled code and package it in its distributable format, such as a JAR.
- **integration-test**: process and deploy the package if necessary into an environment where integration tests can be
- **verify**: run any checks to verify the package is valid and meets quality criteria
- **install**: install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing

There are two other Maven lifecycles of note beyond the *default* list above. They are

- **clean**: cleans up artifacts created by prior builds
- **site**: generates site documentation for this project

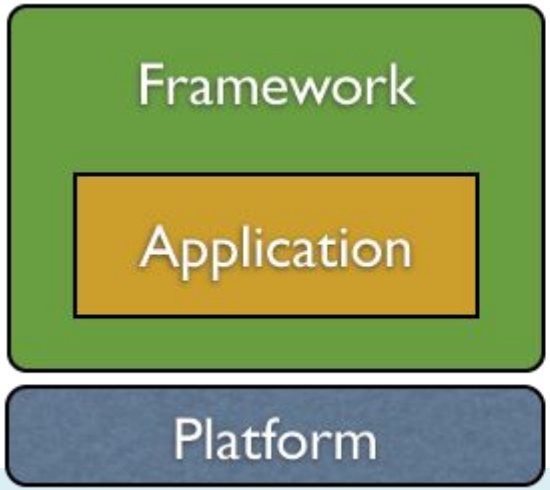
https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Under the Hood: Libraries & Frameworks

Which kind is a command-line parsing package?

Which kind is Android?

How about a tool that runs tests based on annotations you add in your code?



http://tom.lokhorst.eu/2010/09/why-libraries-are-better-than-frameworks`

HW1: Extending the Flash Card System

Principles of Software Construction: Objects, Design, and Concurrency

Specifications and unit testing, exceptions

Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Where we are

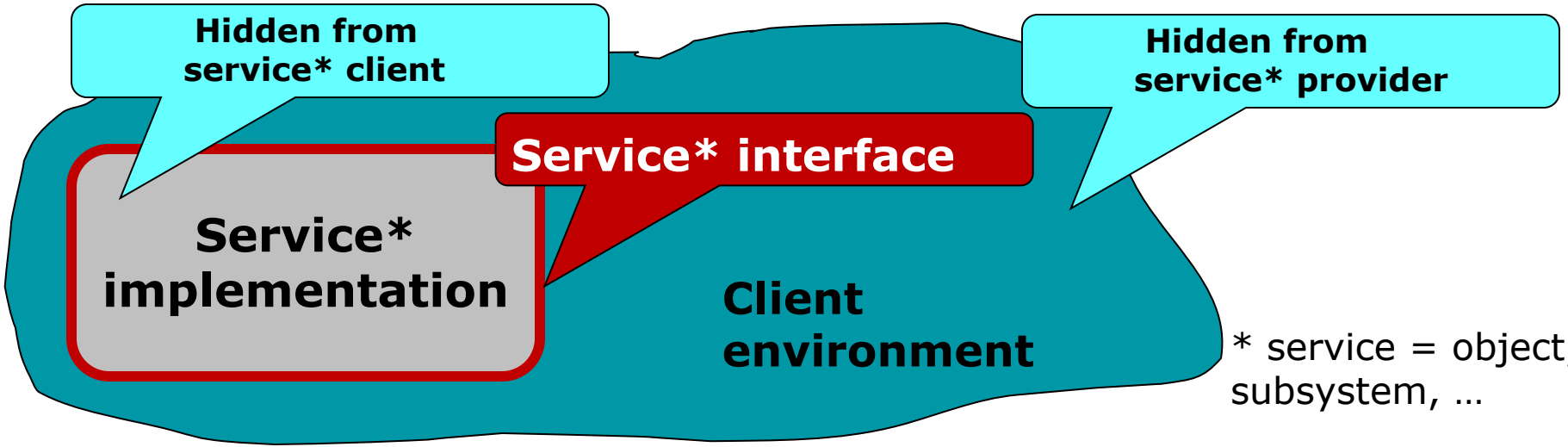
Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, **Contracts** ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, Design Patterns, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	**Unit Testing ✓**	Integration Testing ✓	CI ✓, DevOps ✓, Teams

Who’s to blame?

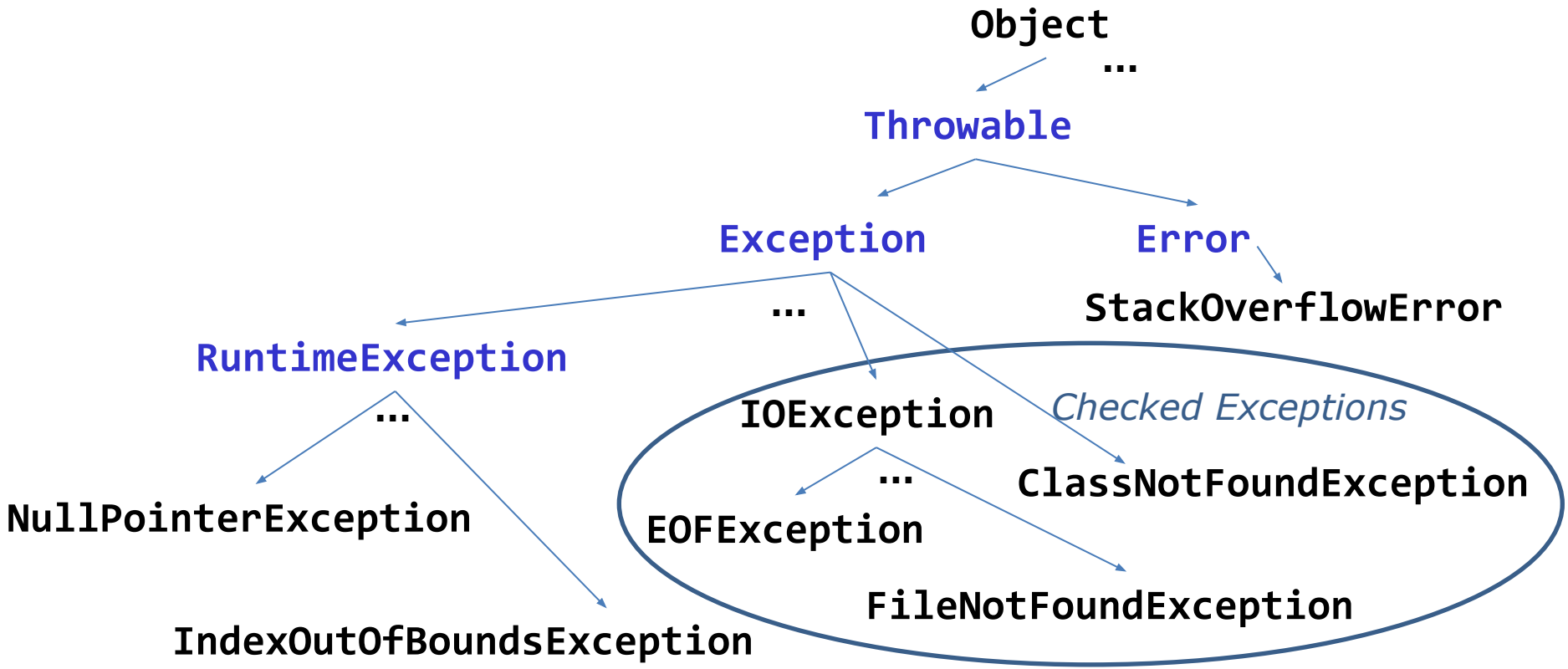
```
Algorithms.shortestDistance(g, “Tom”, “Anne”);

> ArrayOutOfBoundsException
```

Most real-world code has a contract

- Imperative to build systems that scale!
- This is why we:
  - Encode specifications
  - Test



Java's exception hierarchy (messy)



This is Java code

Testing

How do we know this works?

Testing

Are we done?

```
int isPos(int x) {
  return x >= 1;
}

@Test
void testIsPos() {
  assertTrue(isPos(1));
}

@Test
void testNotPos() {
  assertFalse(isPos(-1));
}
```

Docstring Specification

```
class RepeatingCardOrganizer {
  ...
  /**
   * Checks if the provided card has been answered correctly the required
number of times.
   * @param card The {@link CardStatus} object to check.
   * @return {@code true} if this card has been answered correctly at least
{@code this.repetitions} times.
   */
  public boolean isComplete(CardStatus card) {
    // IGNORE THIS WHEN SPECIFICATION TESTING!
  }
}
```

Boundary Value Testing

This is Java code

We cannot test for every integer.

Choose *representative* values:
1 for positives, -1 for negatives

And *boundary cases*: 0 is a likely candidate for mistakes

- Think like an attacker

```java
int isPos(int x) {
  return x >= 0;  // What if?
}

@Test
void test1IsPos() {
  assertTrue(isPos(1));
}

@Test
void test0IsNotPos() {
  assertFalse(isPos(0)); // Fails
}
```

Principles of Software Construction: Objects, Design, and Concurrency

Test case design

Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Specification vs. Structural Testing

- Specification-based testing: test solely the specification
  - Ignores implementation, use inputs/outputs only
  - Typical objective: Cover all specified behavior

- Structural Testing: consider implementation
  - Typical objective: Optimize for various kinds of code coverage
    - Line, Statement, Data-flow, etc.

CreditWallet.pay()

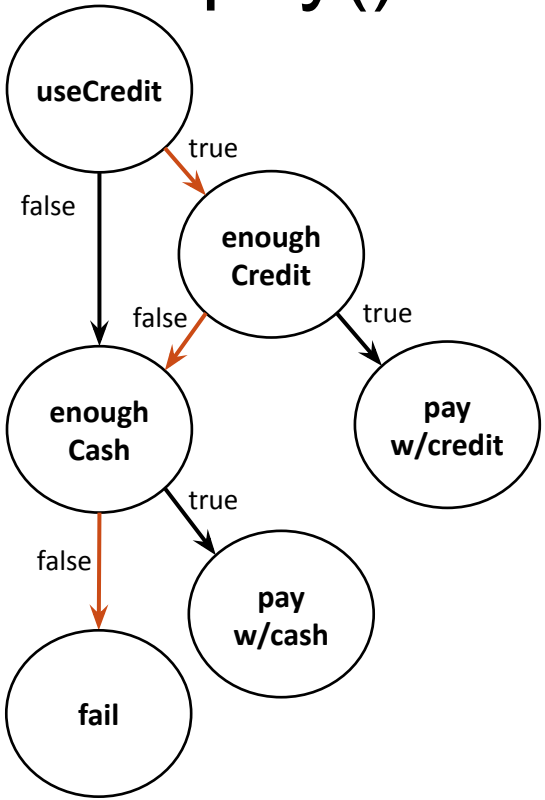
```
public boolean pay(int cost, boolean useCredit) {
    if (useCredit) {
        if (enoughCredit) {
            return true;
        }
    }
    if (enoughCash) {
        return true;
    }
    return false;
}
```

Test case	useCredit	Enough Credit	Enough Cash	Result	Coverage
1	T	T	-	Pass	--
2	F	-	T	Pass	--
3	F	-	F	Fails	Statement

Control-Flow of CreditCard.pay()

Paths:

- {true, true}: pay w/credit
- {false, true}: pay w/cash
- {false, false}: fail
- {true, false, true}: pay w/cash after failing credit
- {true, false, false}: try credit, but fail, **and** no cash



Writing Testable Code

Aim to write easily testable code

- Which is almost by definition more modular

```
public List<String> getLines(String path) throws IOException {
   return Files.readAllLines(Path.of(path));
}

public boolean hasHeader(List<String> lines) {
   return !lines.get(0).isEmpty()
}

// Test:
// - hasHeader with empty, non-empty first line
// - getLines (if you must) with null, real path
```

Boundary Value Testing

We need a *strategy* to identify plausible mistakes

- Boundary Value Testing: errors often occur at boundary conditions
  - Identify equivalence partitions: regions where behavior should be the same
    - `cost <= money: true, cost > money: false`
    - Boundary value: `cost == money`

```
/** Returns true and subtracts cost if enough
  * money is available, false otherwise.
  */
public boolean pay(int cost) {
   if (cost < this.money) {
       this.money -= cost;
       return true;
   }
   return false;
}
```

HW 2: Testing the Flash Card System

Principles of Software Construction: Objects, Design, and Concurrency

Object-oriented Analysis

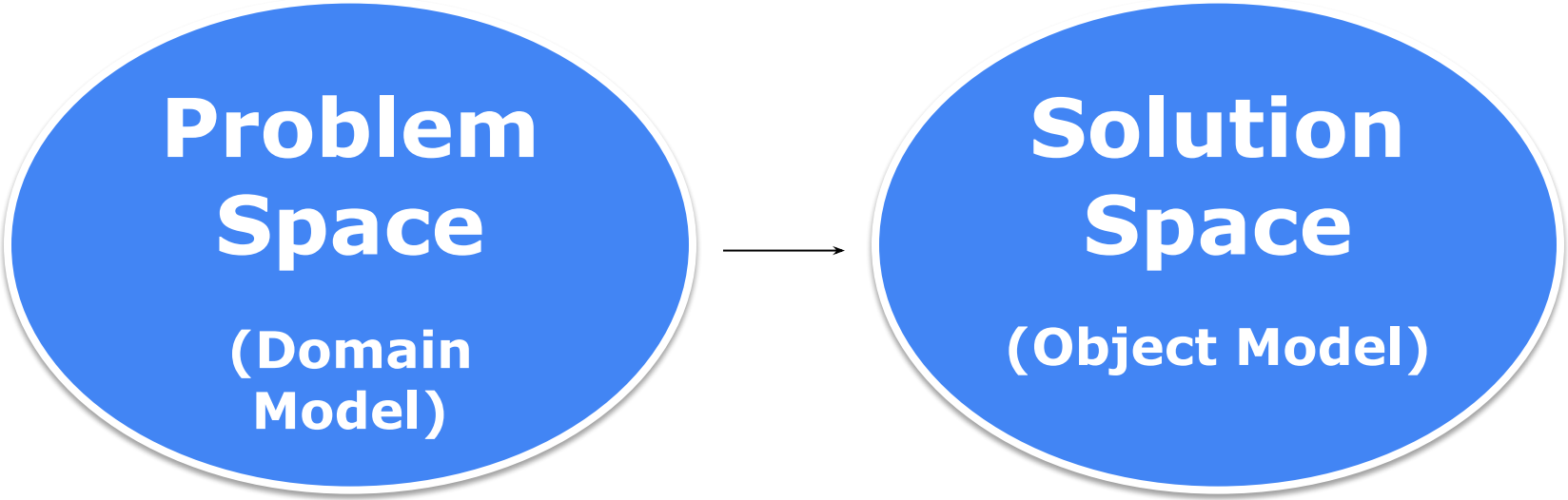
Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	**Domain Analysis ✓**	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, Design Patterns, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams



- Real-world concepts
- Requirements, Concepts
- Relationships among concepts
- Solving a problem
- Building a vocabulary

- System implementation
- Classes, objects
- References among objects and inheritance hierarchies
- Computing a result
- Finding a solution

An object-oriented design process

Model / diagram the problem, define concepts

- **Domain model** (a.k.a. conceptual model), **glossary**

Define system behaviors

- **System sequence diagram**
- **System behavioral contracts**

Assign object responsibilities, define interactions

- **Object interaction diagrams**

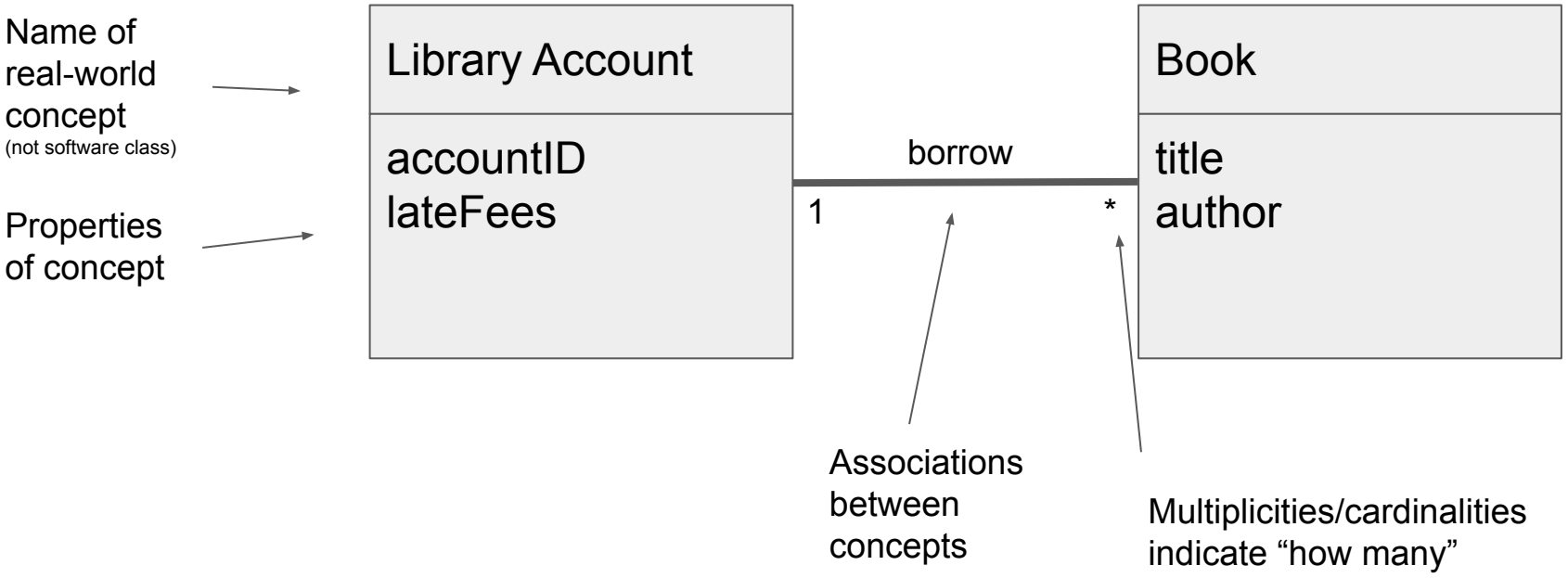
Model / diagram a potential solution

- **Object model**

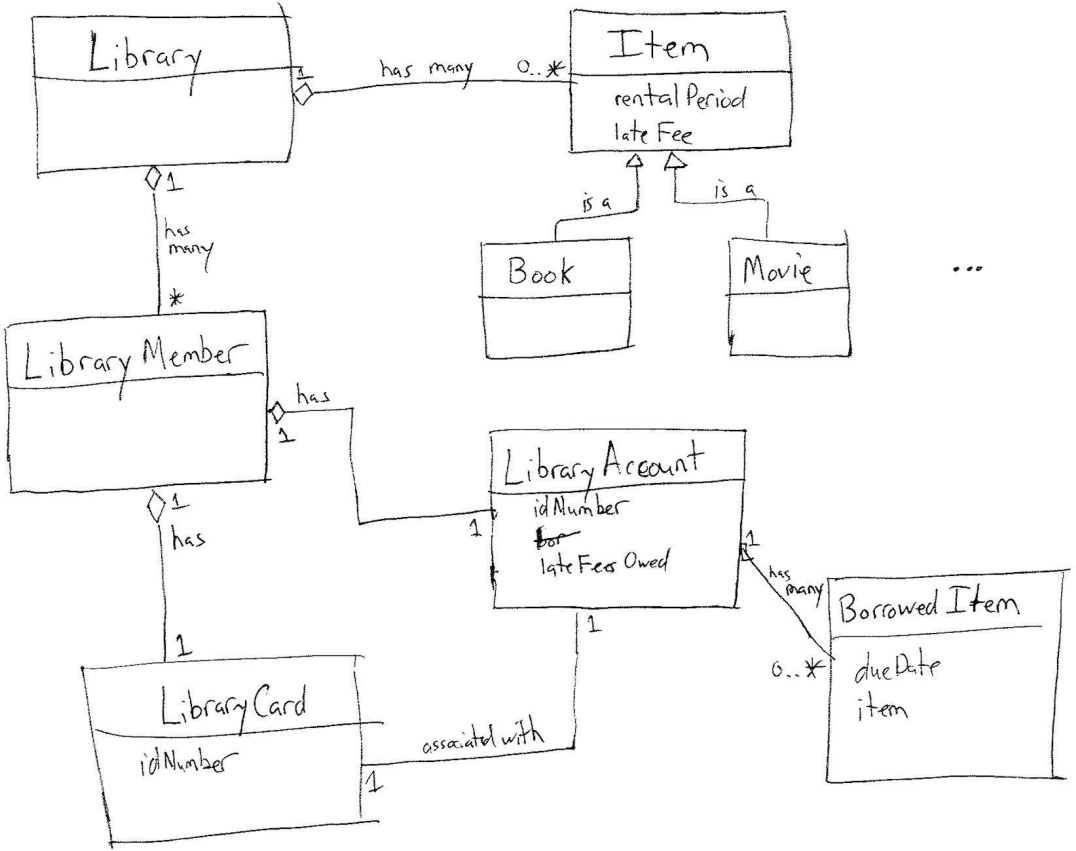
OO Analysis: Understanding the problem

OO Design: Defining a solution

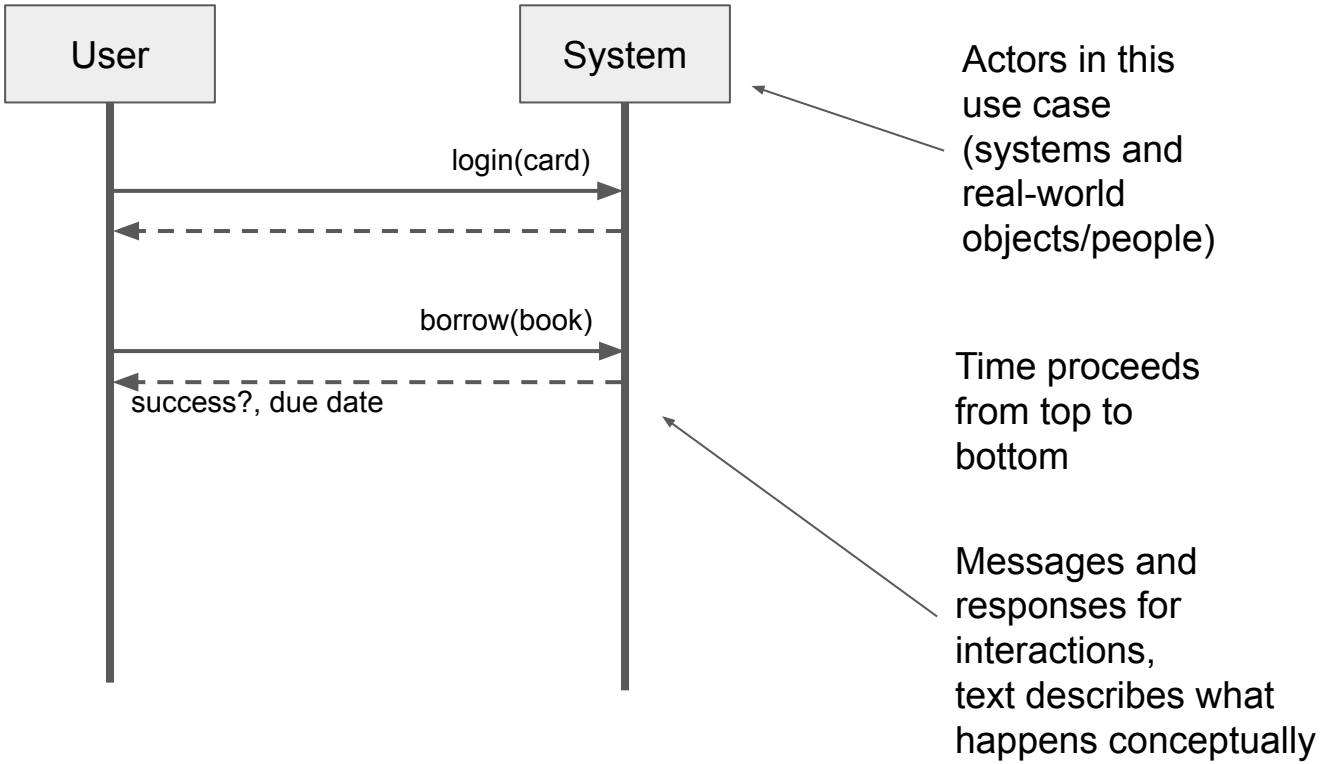
Visual notation: UML



One domain model for the library system



UML Sequence Diagram Notation



Principles of Software Construction: Objects, Design, and Concurrency

Object-oriented Design

Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

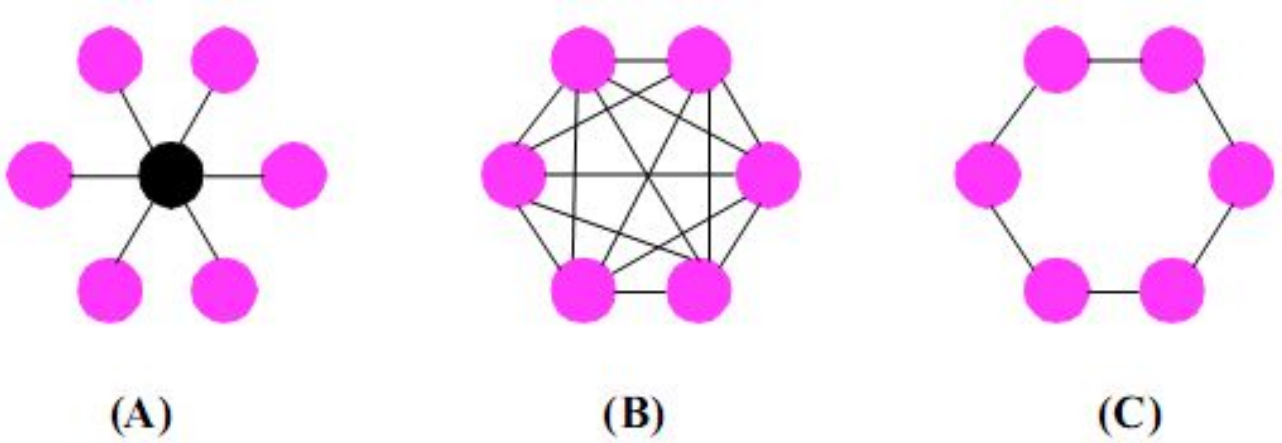
institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	**Responsibility Assignment,** Design Patterns, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

Topologies with different coupling

Types of module interconnection structures



Design Heuristic: Law of Demeter

- *Each module should have only limited knowledge about other units: only units "closely" related to the current unit*
- In particular: Don’t talk to strangers!
- For instance, no a.getB().getC().foo()

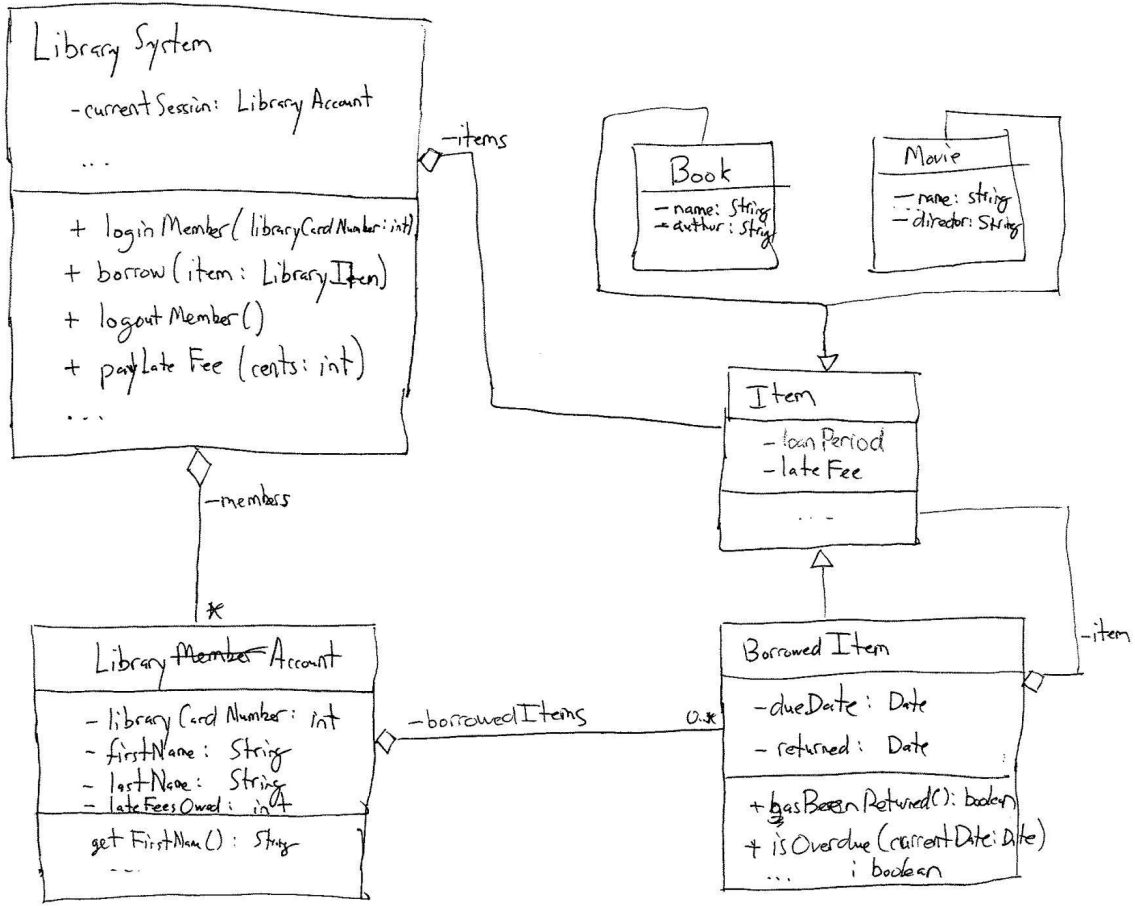
```
for (let i of shipment.getBox().getItems())
   shipmentWeight += i.getWeight() …
```

Object Diagrams

Objects/classes with fields and methods

Interfaces with methods

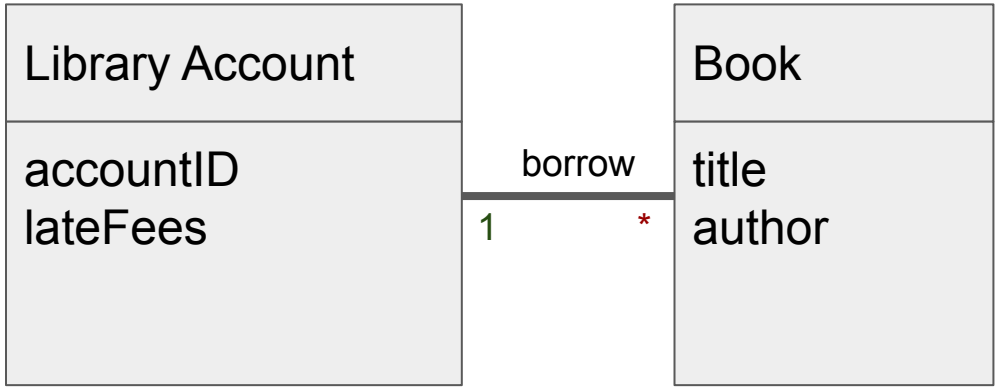
Associations, visibility, types



Low Representational Gap

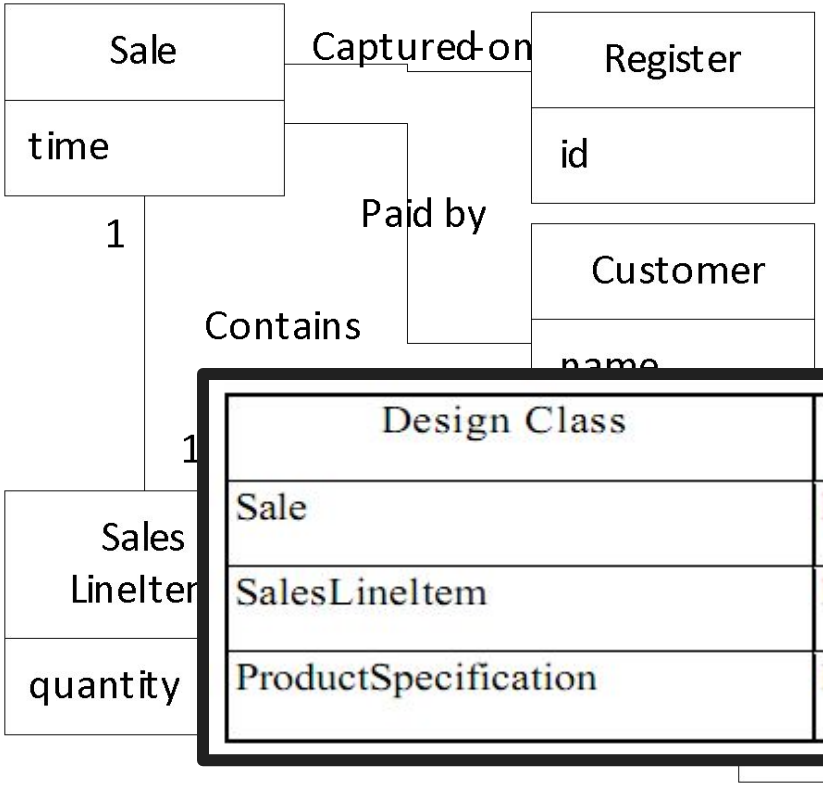
Identified concepts provide inspiration for classes in the implementation

Classes mirroring domain concepts often intuitive to understand, rarely change (low representational gap)




```
class LibraryDatabase {
    Map<Int, List<Int>>
        borrowedBookIds;
    Map<Int, Int> lateFees;
    Map<Int, String>
        bookTitles;
}
class DatabaseRow { … }
```

Who should be responsible for knowing the grand total of a sale?



Design Class	Responsibility
Sale	knows sale total
SalesLineItem	knows line item subtotal
ProductSpecification	knows product price

Anti-Pattern: God Object

```
class Chat {
    Content content;
    AccountMgr accounts;
    File logFile;
    ConnectionMgr conns;
}
class ChatUI {
    Chat chat;
    Widget sendButton, …;
}
class AccountMgr {
    … acounts, bannedUsr…
}
```

```
class Chat {
    List<String> channels;
    Map<String, List<Msg>> messages;
    Map<String, String> accounts;
    Set<String> bannedUsers;
    File logFile;
    File bannedWords;
    URL serverAddress;
    Map<String, Int> globalSettings;
    Map<String, Int> userSettings;
    Map<String, Graphic> smileys;
    CryptStrategy encryption;
    Widget sendButton, messageList;
```

Information Expert -> "Do It Myself Strategy"

Expert usually leads to designs where a software object does those operations that are normally done to the inanimate real-world thing it represents

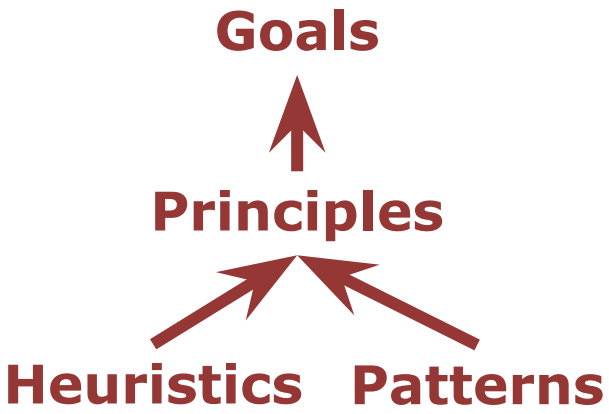
- a sale does not tell you its total; it is an inanimate thing

In OO design, all software objects are "alive" or "animated," and they can take on responsibilities and do things.

They do things related to the information they know.

Design Goals, Principles, and Patterns

- Design Goals
  - Design for change, understanding, reuse, division of labor, …
- Design Principle
  - Low coupling, high cohesion
  - Low representational gap
- Design Heuristics
  - Law of demeter
  - Information expert
  - Creator
  - Controller



HW3: Santorini (Base game)

Need Help?

Video Tutorials More of a visual learner? We've got you covered! Head over to roxley.com/santorini-video for video tutorials on how to play, as well as complete visual demonstrations of all God Powers!

Santorini App Can't decide which God Powers to match up? Head over to Google Play Store or the Apple App Store and download the Santorini App absolutely free. Complete with video tutorials, match randomizer and much more!

Setup

1. Place the smaller side of the Cliff Pedestal (A) on the Ocean Board (B), using the long and short tabs on the Cliff Pedestal to guide assembly.
2. Place the Island Board (C) on top of the Cliff Pedestal (A), again using the long and short tabs to guide assembly.
3. The youngest player is the Start Player, who begins by placing 2 Workers (D) of their chosen color into any unoccupied spaces on the board. The other player(s) then places their Workers (E).

How To Play

Players take turns, starting with the Start Player, who first placed their Workers. On your turn, select one of your Workers. You must move and then build with the selected Worker.

Move your selected Worker into one of the (up to) eight neighboring spaces .

A Worker may move up a maximum of one level higher, move down any number of levels lower, or move along the same level. A Worker may not move up more than one level .

The space your Worker moves into must be unoccupied (not containing a Worker or Dome).

Build a block ( ) or dome ( ) on an unoccupied space neighboring the moved Worker.

Winning the Game

1. If one of your Workers moves up on top of level 3 during your turn, you instantly win!
2. You must always perform a move then build on your turn. If you are unable to, you lose.

Components

Principles of Software Construction: Objects, Design, and Concurrency

Inheritance and delegation

Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

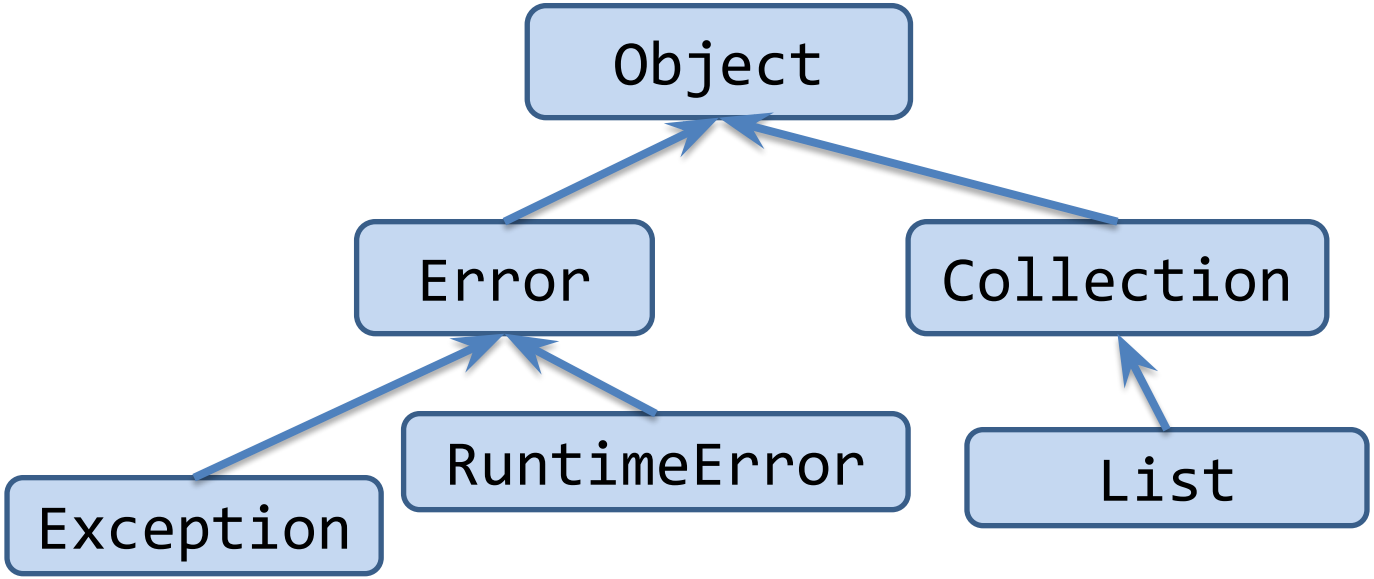
institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext. reuse	Information Hiding, Contracts ✓	**Inheritance & Del.** ✓	Frameworks and Libraries ✓, APIs ✓
robustness	Immutability ✓	Responsibility Assignment, Design Patterns, Antipattern ✓	Module systems, microservices ✓
...	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

Class Hierarchy

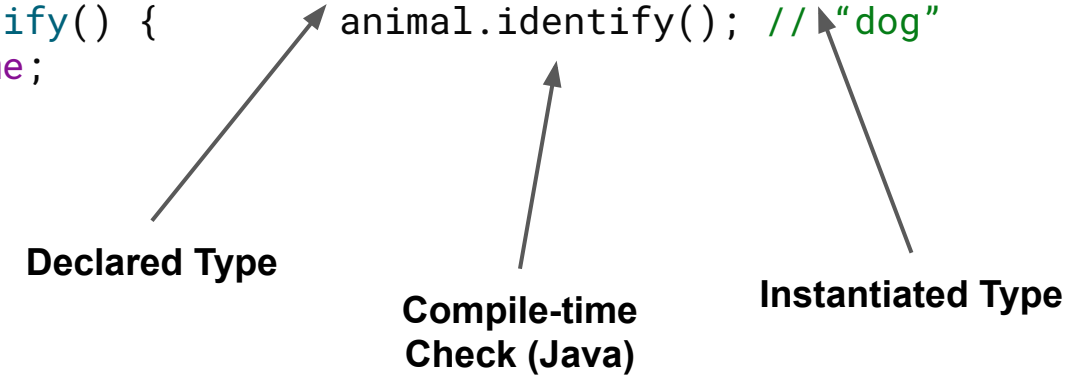
In Java:



Inheritance enables Extension & Reuse

```
class Animal {
   final String name;

   public Animal(String name) {
       this.name = name;
   }

   public String identify() {
       return this.name;
   }
}
```

```
class Dog extends Animal {
   public Dog() {
       super("dog");
   }
}

Animal animal = new Dog();
animal.identify(); // "dog"
```



Is this Square a behavioral subtype of Rectangle?

```
class Rectangle {
     //@ invariant h>0 && w>0;
     int h, w;

     Rectangle(int h, int w) {
          this.h=h; this.w=w;
     }
     //@ requires factor > 0;
     void scale(int factor) {
          w=w*factor;
          h=h*factor;
     }
     //@ requires neww > 0;
     void setWidth(int neww) {
          w=neww;
     }
}
```

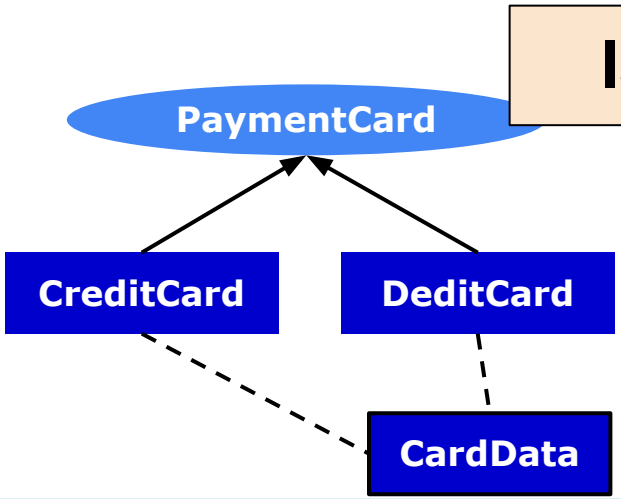
```
class Square extends Rectangle {
     //@ invariant h>0 && w>0;
     //@ invariant h==w;
     Square(int w) {
          super(w, w);
     }
}
```

```
class GraphicProgram {
    void scale(Rectangle r, int factor) {
        r.setWidth(r.getWidth() * factor);
    }
}
```

Technically yes! But: `Square` is not a square :(

Reuse does not require Inheritance, Delegation is enough

```
public interface PaymentCard {
   CardData getCardData();
   int getValue();
   boolean pay(int amount);
}
```




```
class CardData {
   private final String cardHolderName;
   private final BigInteger digits;
   private final Date expirationDate;

   ardData(String cardHolderName,
           Integer digits, Date expirationDate) {
       .cardHolderName = cardHolderName;
       this.digits = digits;
       this.expirationDate = expirationDate;
   }

   @Override
   public String getCardHolderName() {
       return this.cardHolderName;
   }
```

Inheritance limits information hiding!

```
public class InstrumentedHashSet<E> extends HashSet<E> {

   public int addCount = 0;

   @Override
   public boolean add(E a) {
       addCount += 1;
       return super.add(a);
   };

   @Override
   public boolean addAll(Collection<? extends E> c) {
       addCount += c.size();
       return super.addAll(c);
   }
}
```

```
public static void main(String[] args) {
   InstrumentedHashSet<String> set = new
InstrumentedHashSet<String>();

   set.addAll(List.of("A", "B", "C"));

   System.out.println(set.addCount);
}
```

What will this print?

Principles of Software Construction: Objects, Design, and Concurrency

Design Patterns

Claire Le Goues Bogdan Vasilescu

Carnegie Mellon University
School of Computer Science

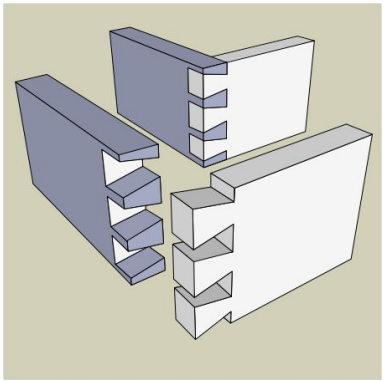
institute for SOFTWARE RESEARCH

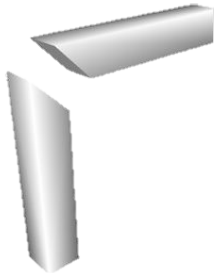
Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, **Design Patterns**, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

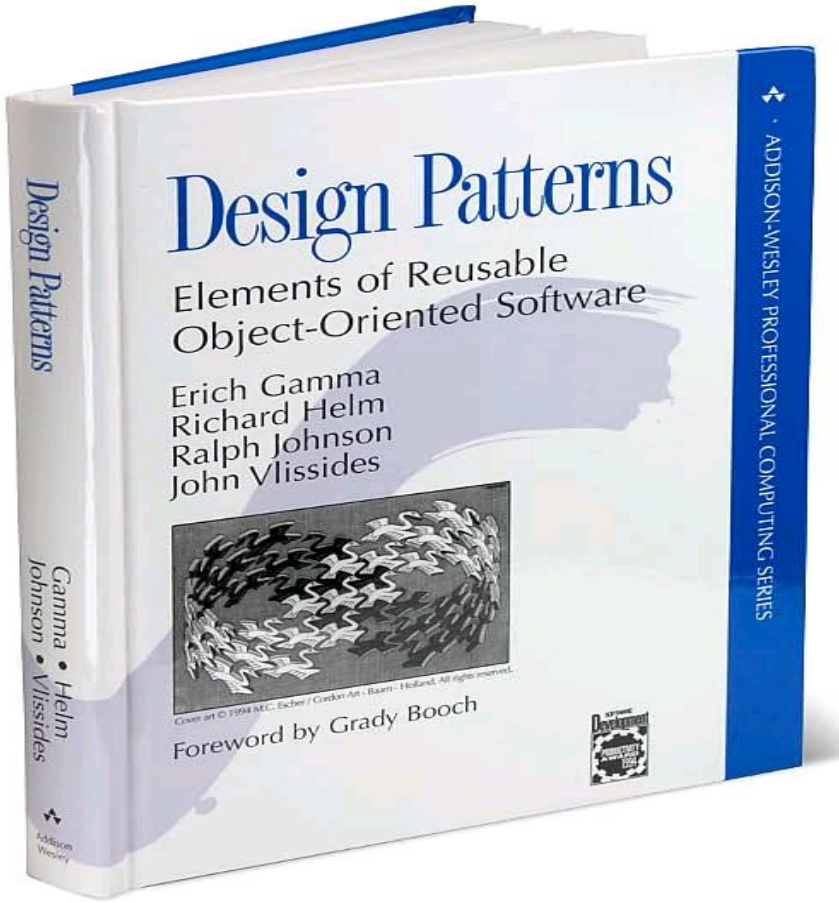
Discussion with design patterns

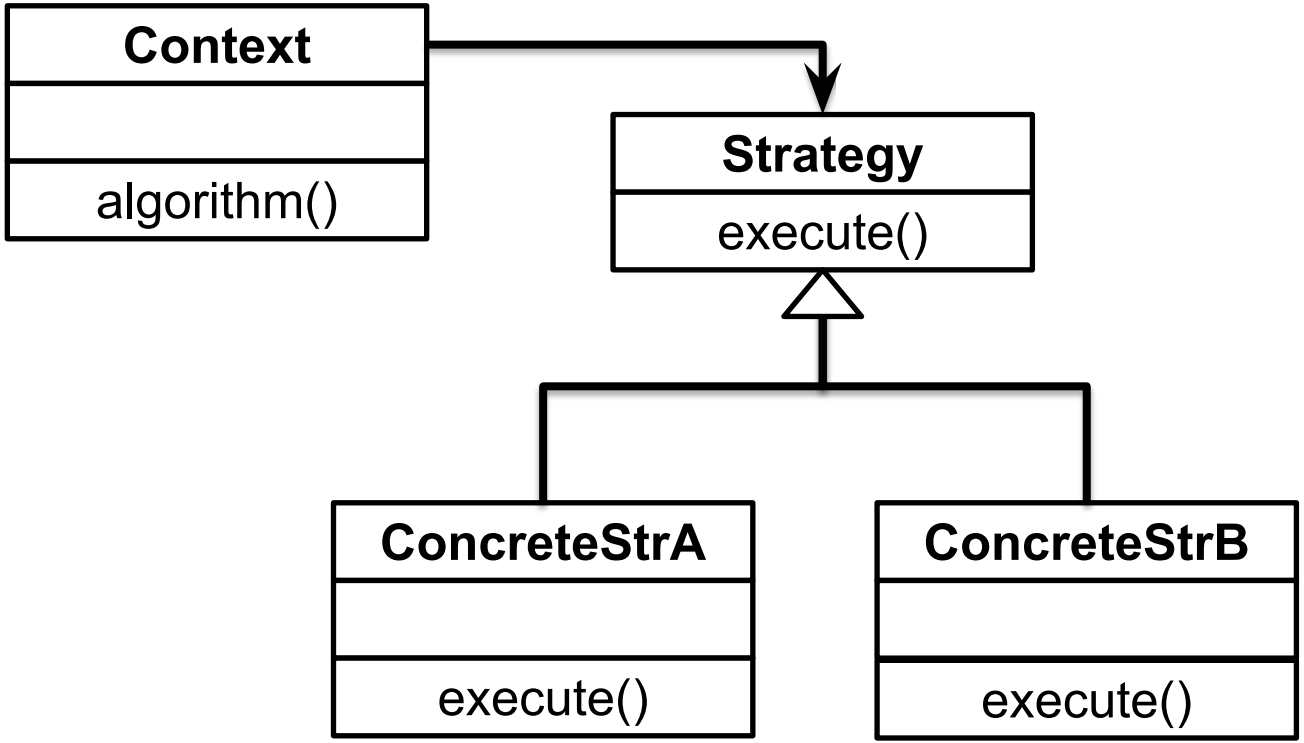
- Carpentry:
  - "Is a dovetail joint or a miter joint better here?"
- Software Engineering:
  - "Is a strategy pattern or a template method better here?"





History: *Design Patterns* (1994)




Context
algorithm()
Strategy
execute()
ConcreteStrA
execute()
ConcreteStrB
execute()

One design scenario

- Amazon.com processes millions of orders each year, selling in 75 countries, all 50 states, and thousands of cities worldwide. These countries, states, and cities have hundreds of distinct sales tax policies and, for any order and destination, Amazon.com must be able to compute the correct sales tax for the order and destination.

Module pattern: Hide internals in closure

```
(function () {
    // ... all vars and functions are in this scope only
    // still maintains access to all globals
}());
```

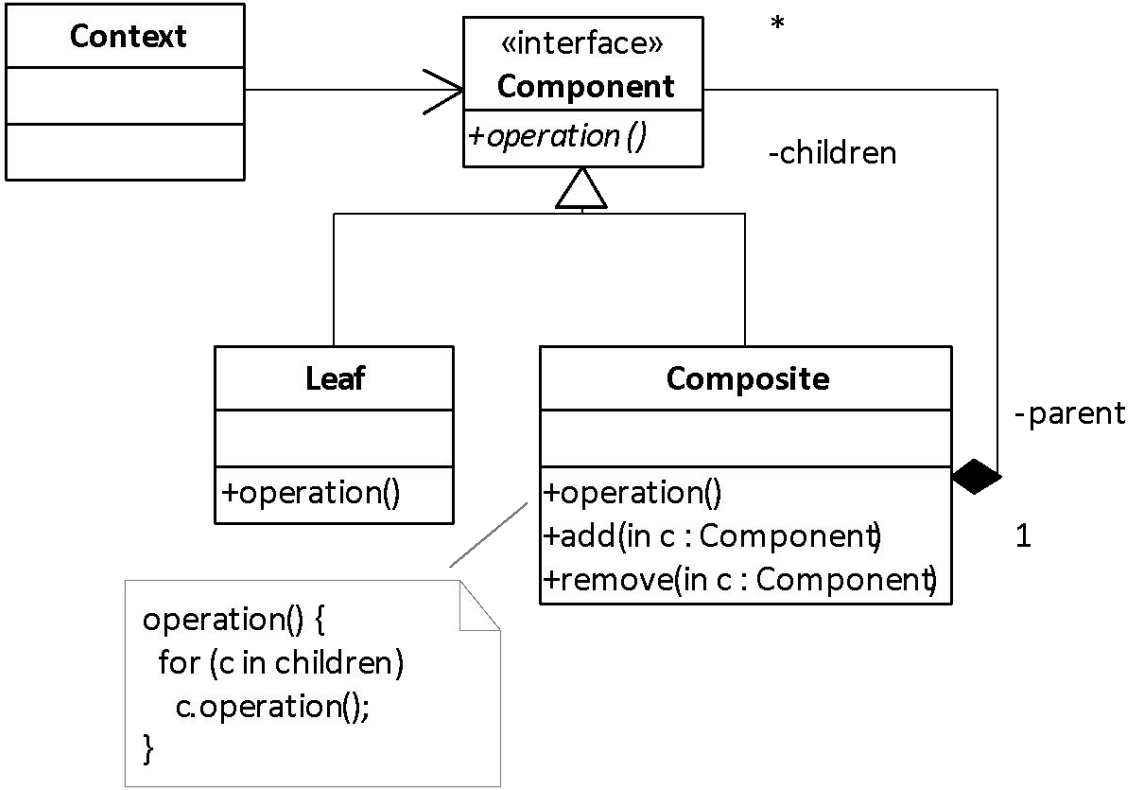
Function provides local scope, internals not accessible

Function directly invoked to execute it once

Wrapped in parentheses to make it expression

Discovered around 2007, became very popular, part of Node

The Composite Design Pattern



Principles of Software Construction: Objects, Design, and Concurrency

Refactoring & Anti-patterns

Claire Le Goues Bogdan Vasilescu

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

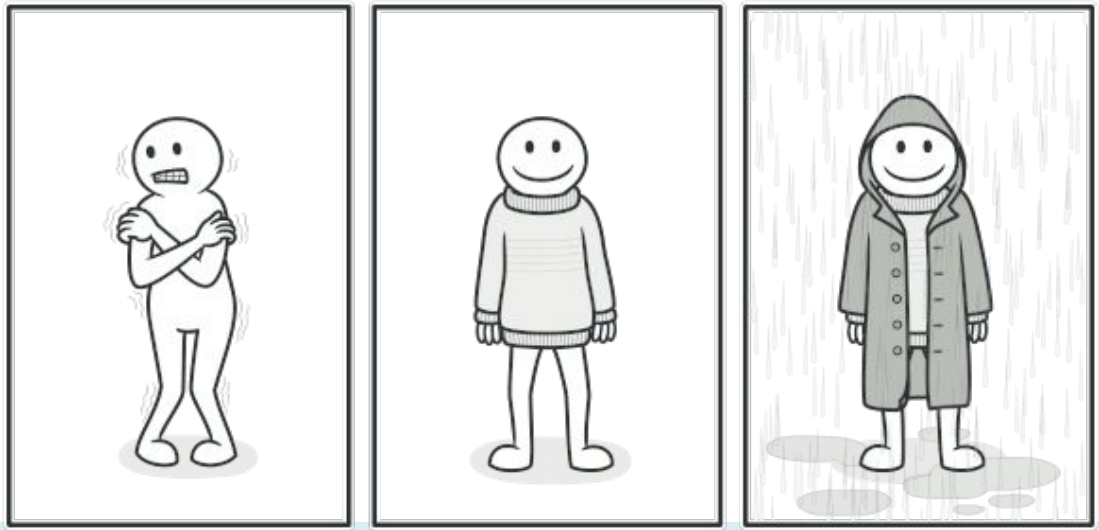
Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, Design Patterns, **Antipattern** ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

The Decorator Pattern

You have a complex drawing that consists of many shapes and want to save it. Some logic of the saving functionality is always the same (e.g., going through all shapes, reducing them to drawable lines), but others you want to vary to support saving in different file formats (e.g., as png, as svg, as pdf). You want to support different file formats later.

Why is this not:



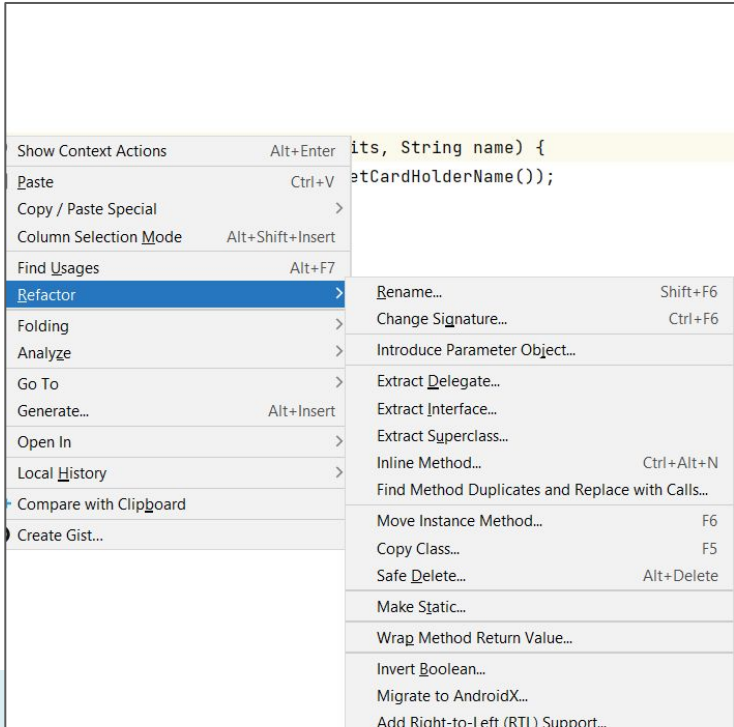
https://refactoring.guru/design-patterns/decorator

Anti-patterns

- We have talked a fair bit about bad design heuristics
  - High coupling, low cohesion, law of demeter, …
- You will see a much larger vocabulary of related issues
  - Commonly called code/design “smells”
  - Worthwhile reads:
    - **A short overview: https://refactoring.guru/refactoring/smells**
    - Wikipedia: https://en.wikipedia.org/wiki/Anti-pattern#Software_engineering
    - Book on the topic (no required reading): Refactoring for Software Design Smells: Managing Technical Debt, Suryanarayana, Samarthyam and Sharma
      - S.O. summary: https://stackoverflow.com/a/27567960

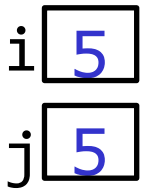
Refactoring: IDE support

- Rename class, method, variable to something not in-scope
- Extract method/inline method
- Extract interface
- Move method (up, down, laterally)
- Replace duplicates

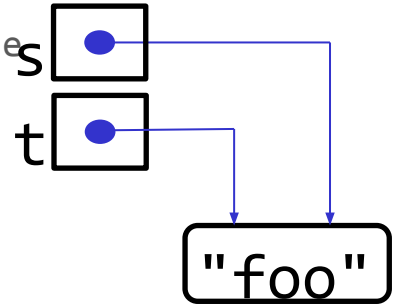


True or false?

```
int i = 5;
int j = 5;
System.out.println(i == j);
---------------------------
true
```

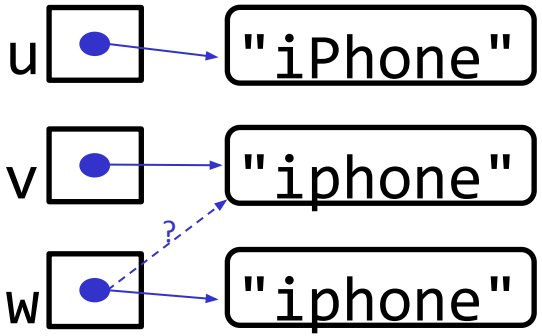



```
String s = "foo";
String t = s;
System.out.println(s == t);
---------------------------
true
```




```
String u = "iPhone";
String v = u.toLowerCase();
String w = "iphone";
System.out.println(v == w);
---------------------------
```

false (in practice)



Liquid APIs

Each method changes state,
then returns `this`

(Immutable version:
Return modified copy)

```
class OptBuilder {
    private String argName = "";
    private boolean hasArg = false;
    ...
    OptBuilder withArgName(String n) {
        this.argName = n;
        return this;
    }
    OptBuilder hasArg() {
        this.hasArg = true;
        return this;
    }
    ...
    Option create() {
        return new Option(argName,
            hasArgs, ...)
    }
}
```

Traversing a collection

- Since Java 1.0:

```
Vector arguments = …;
for (int i = 0; i < arguments.size(); ++i) {
  System.out.println(arguments.get(i));
}
```

- Java 1.5: enhanced for loop

```
List<String> arguments = …;
for (String s : arguments) {
  System.out.println(s);
}
```

- Works for every implementation of `Iterable`

```
public interface Iterable<E> {
  public Iterator<E> iterator();
}
public interface Iterator<E> {
  boolean hasNext();
  E next();
  void remove();
```

- In JavaScript (ES6)

```
let arguments = …
for (const s of arguments) {
  console.log(s)
}
```

- Works for every implementation with a “magic” function `[Symbol.iterator]` providing an iterator

```
interface Iterator<T> {
  next(value?: any): IteratorResult<T>;
  return?(value?: any): IteratorResult<T>;
  throw?(e?: any): IteratorResult<T>;
}
interface IteratorReturnResult<TReturn> {
  done: true;
  value: TReturn;
}
```

HW4: Refactoring of Static Website Generator

Principles of Software Construction: Objects, Design, and Concurrency

Asynchrony and Concurrency

Claire Le Goues **Bogdan Vasilescu**

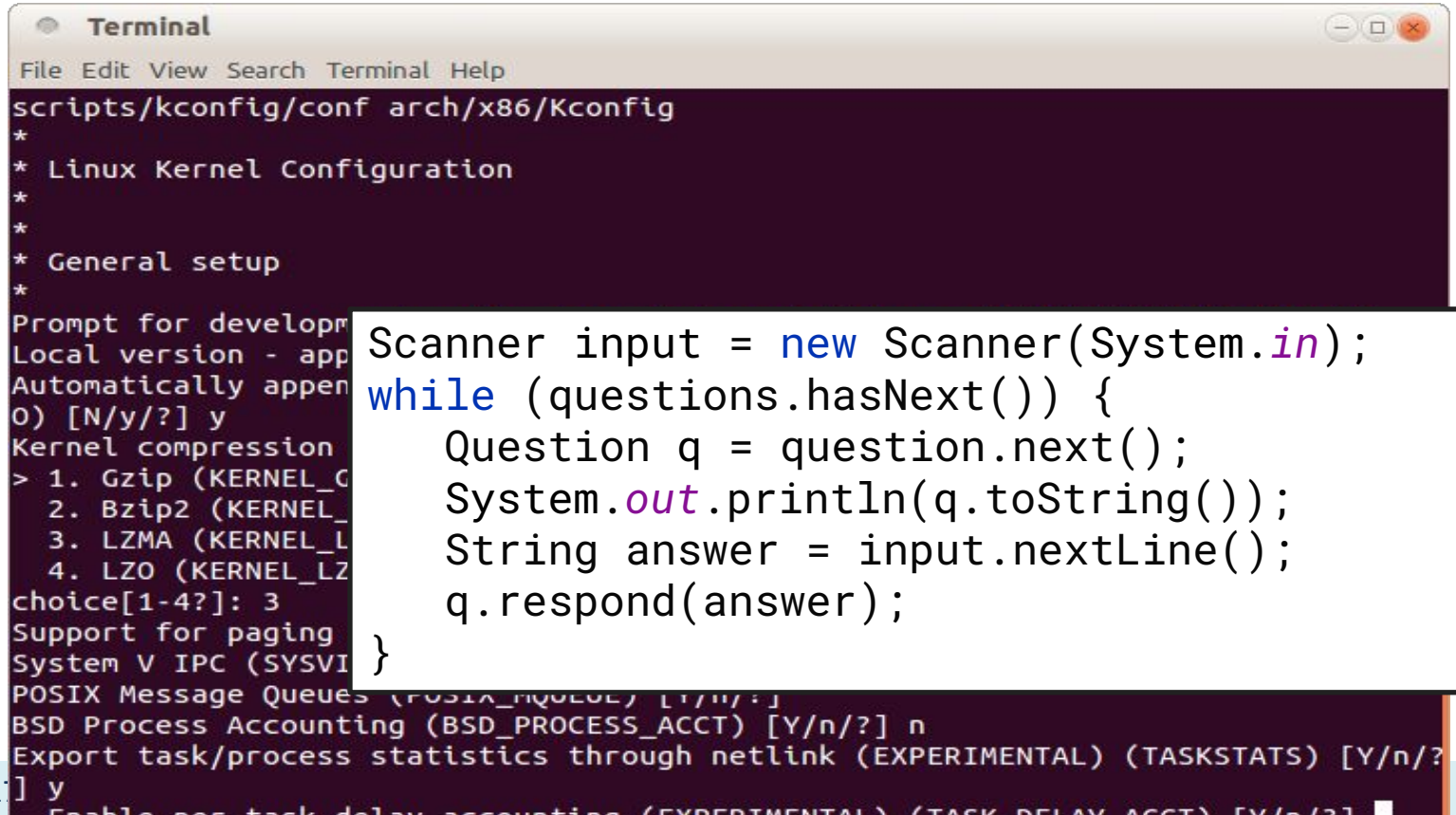
Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	**GUI vs Core** ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, **Design Patterns**, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	**Promises**/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

Interaction with CLI



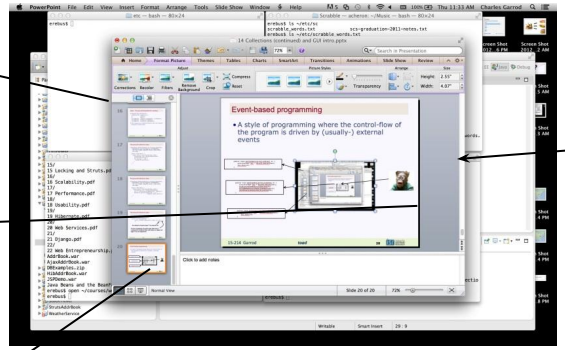
Event-based programming

- Style of programming where control-flow is driven by (usually external) events

```
public void performAction(ActionEvent e) {
    List<String> lst = Arrays.asList(bar);
    foo.peek(42)
}
```

```
public void performAction(ActionEvent e) {
    bigBloatedPowerPointFunction(e);
    withANameSoLongIMadeItTwoMethods(e);
    yesIKnowJavaDoesntWorkLikeThat(e);
}
```

```
public void performAction(ActionEvent e) {
    List<String> lst = Arrays.asList(bar);
    foo.peek(40)
}
```



Concurrency with file I/O

Asynchronous code requires Promises

- Captures an intermediate state
  - Neither fetched, nor failed; we'll find out eventually

```
let imageToBe: Promise<Image> = fetch('myImage.png');
imageToBe.then((image) => display(image))
         .catch((err) => console.log('aw: ' + err));
```

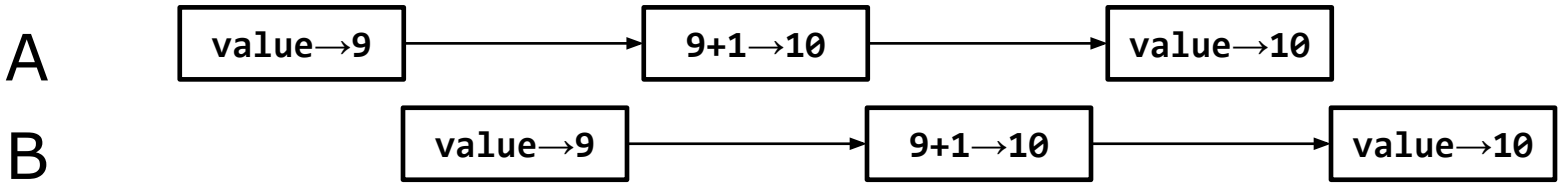
1. Safety Hazard

- The ordering of operations in multiple threads is **unpredictable**.

```
@NotThreadSafe
public class UnsafeSequence {
    private int value;

    public int getNext() {
        return value++;
    }
}
```

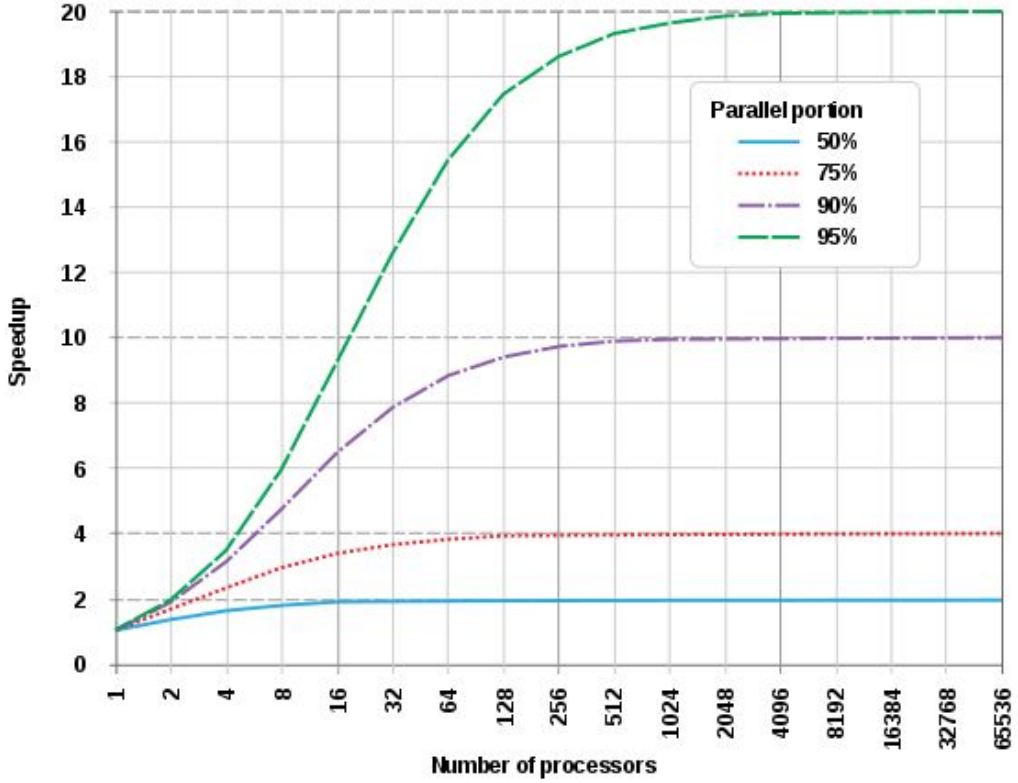
Not atomic

- Unlucky execution of `UnsafeSequence.getNext`

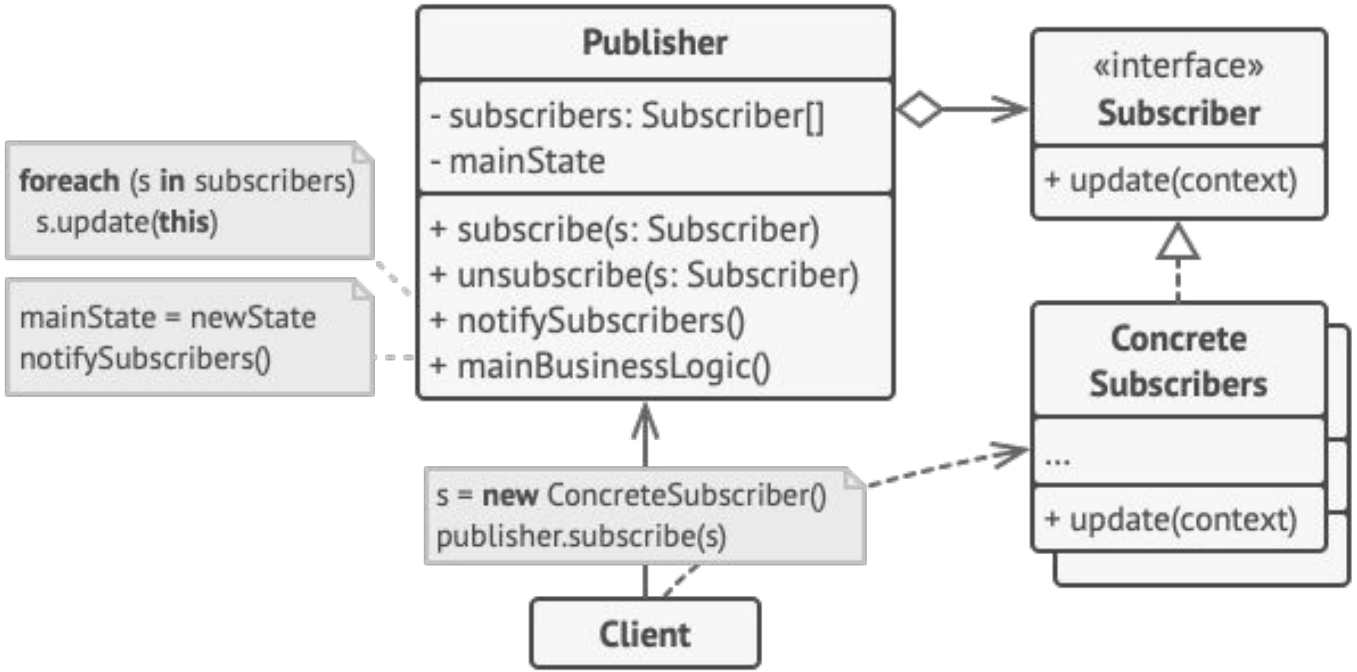


Amdahl's law

- The speedup is limited by the serial part of the program.

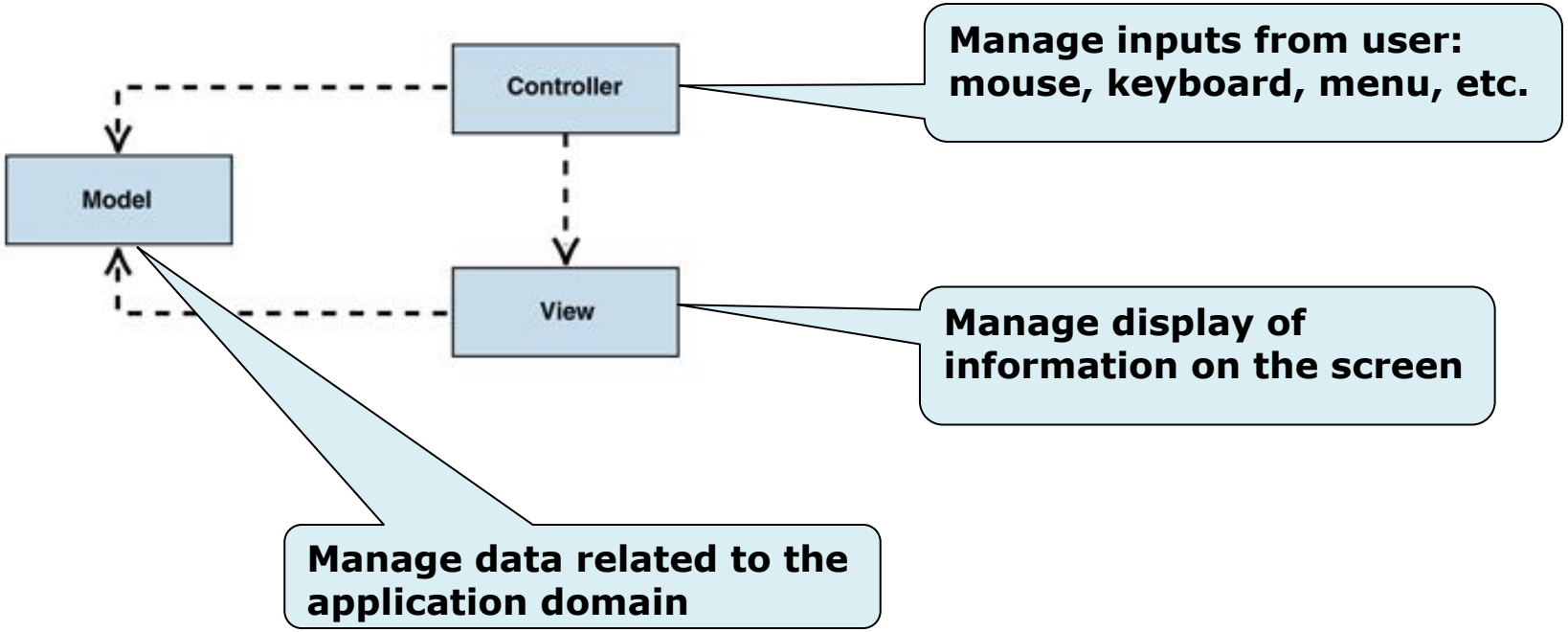


Recall the Observer



https://refactoring.guru/design-patterns/observer

An architectural pattern: Model-View-Controller (MVC)



Principles of Software Construction: Objects, Design, and Concurrency

Basic GUI concepts, HTML

Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

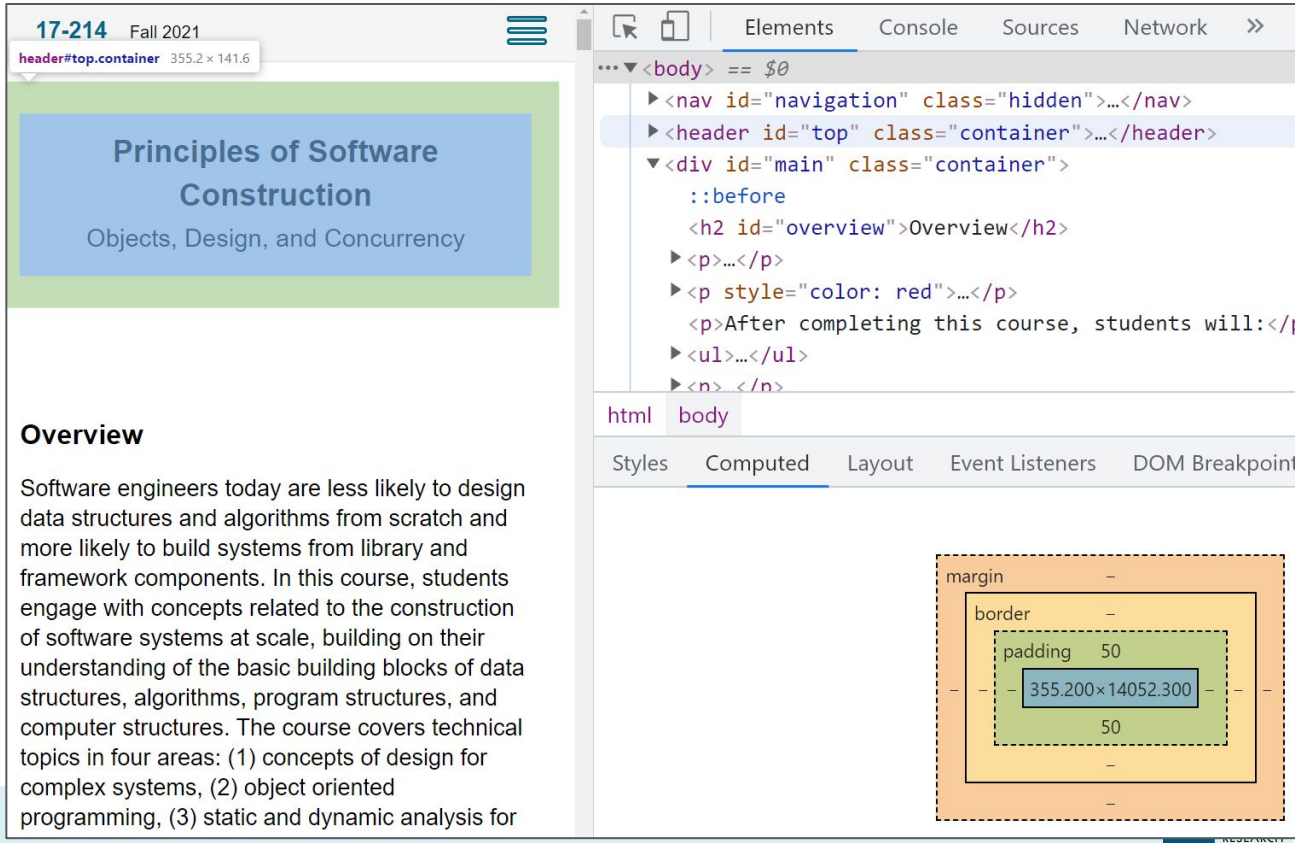
Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	**GUI** vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, **Design Patterns**, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

Anatomy of an HTML Page

Nested elements

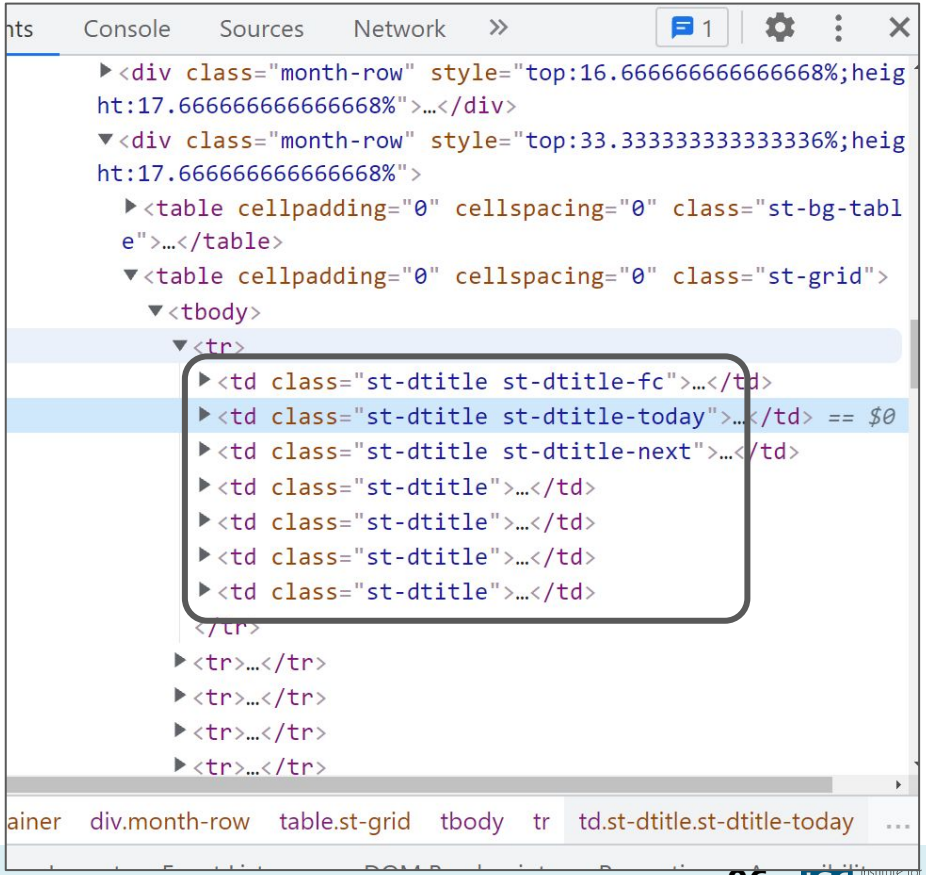
- Sizing
- Attributes
- Text



Strategy or Observer?

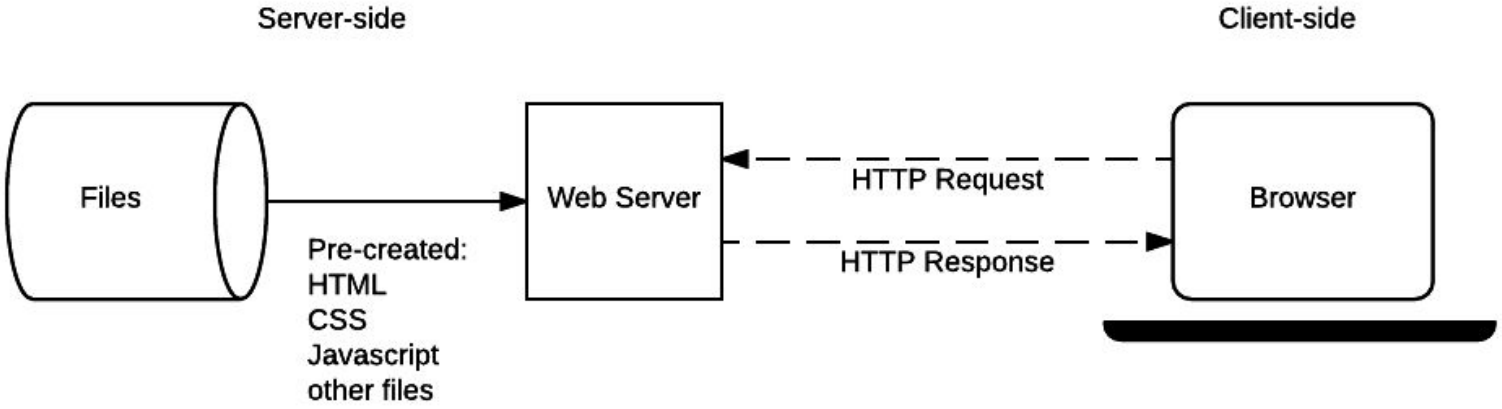
Either could apply

- Both involve callback
- Strategy:
  - Typically single
  - Often involves a return
- Observer:
  - Arbitrarily many
  - Involves external updates



Static Web Pages

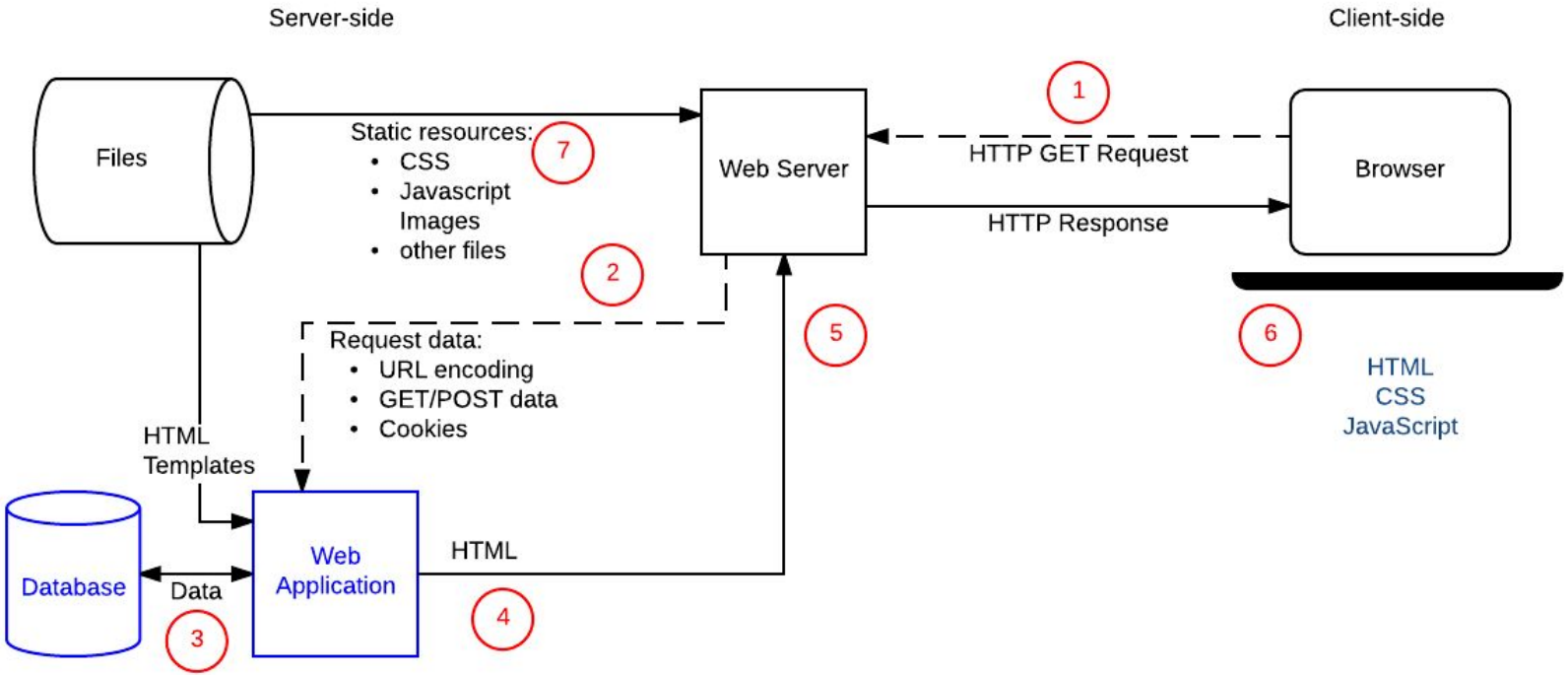
- Delivered as-is, final
  - Consistent, often fast
  - Cheap, only storage needed
- “Static” a tad murky with JavaScript
  - We can still have buttons, interaction



https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview#anatomy_of_a_dynamic_request

Web Servers

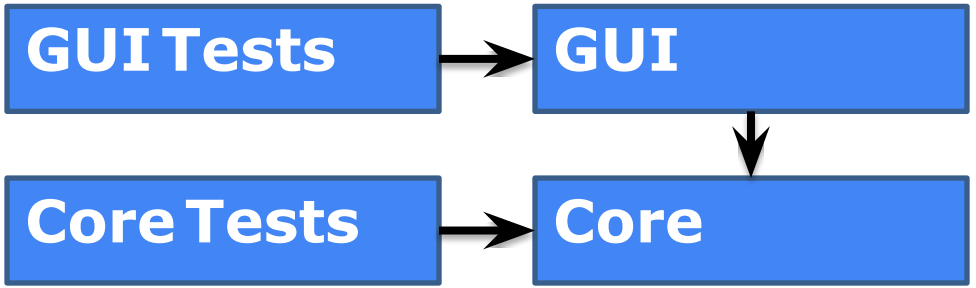
Dynamic sites can do more *work*



https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview#anatomy_of_a_dynamic_request

Separating application core and GUI

- Reduce coupling: do not allow core to depend on UI
- Create and test the core without a GUI
  - Use the Observer pattern to communicate information from the core (Model) to the GUI (View)



Principles of Software Construction: Objects, Design, and Concurrency

Concurrency: Safety & Immutability

Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

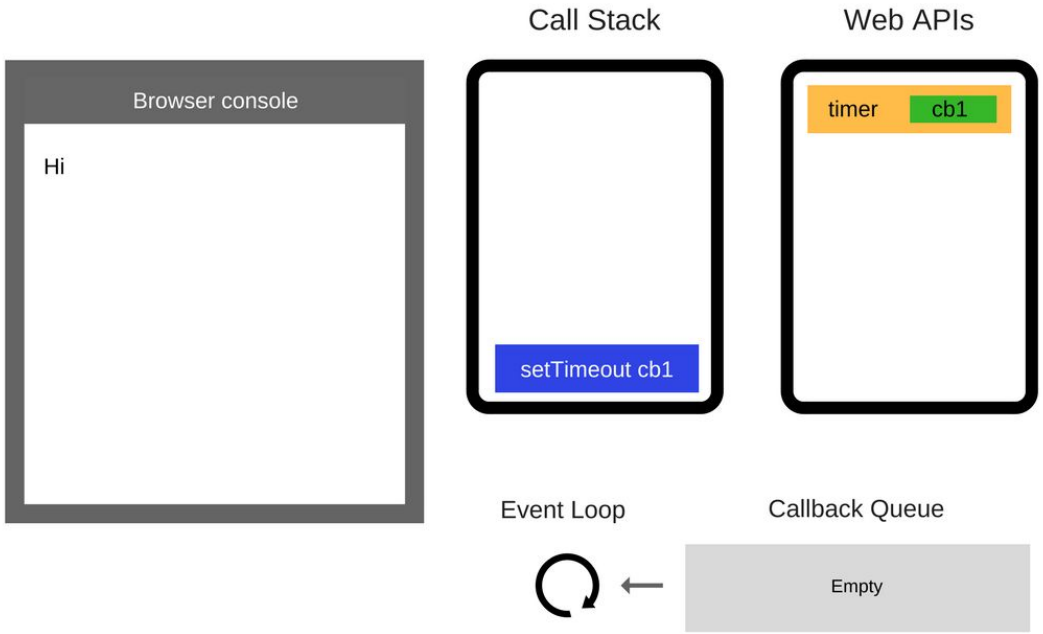
Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	**Immutability** ✓	Responsibility Assignment, Design Patterns, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

The Event Loop

6 / 16

```
console.log('Hi');
setTimeout(function cb1() {
    console.log('cb1');
}, 5000);
console.log('Bye');
```



`setTimeout(function cb1() {...});` is executed.

The browser creates a timer as part of the Web APIs. It will handle the countdown for you.

“Callback Hell”?

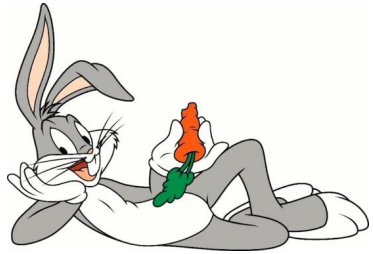
- Issue caused by coding with complex nested callbacks.
- Every callback takes an argument that is a result of the previous callbacks.

If asynchronous:

```
const makeBurger = nextStep => {
  getBeef(function (beef) {
    cookBeef(beef, function (cookedBeef) {
      getBuns(function (buns) {
        putBeefBetweenBuns(buns, beef, function(burger) {
          nextStep(burger)
        })
      })
    })
  })
}

// Make and serve the burger
makeBurger(function (burger) => {
  serve(burger)
})
```

Remember the money-grab example?

```
public static void main(String[] args) throws InterruptedException {
    BankAccount bugs = new BankAccount(1_000_000);
    BankAccount daffy = new BankAccount(1_000_000);

    Thread bugsThread = new Thread(()-> {
        for (int i = 0; i < 1_000_000; i++)
            transferFrom(daffy, bugs, 1);
    });

    Thread daffyThread = new Thread(()-> {
        for (int i = 0; i < 1_000_000; i++)
            transferFrom(bugs, daffy, 1);
    });

    bugsThread.start(); daffyThread.start();
    bugsThread.join(); daffyThread.join();
    System.out.println(bugs.balance() - daffy.balance());
}
```



Making a Class Immutable

```
public final class Complex {
    private final double re, im;

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }

    // Getters without corresponding setters
    public double getRealPart()      { return re; }
    public double getImaginaryPart() { return im; }

    // subtract, multiply, divide similar to add
    public Complex add(Complex c) {
        …
    }
```

What will Happen:

Where does this fail?

What if single threaded?

Could we make it work with 2 threads?

```java
public class Synchronization {
    static long balance1 = 100;
    static long balance2 = 100;

    public static void main(String[] args) throws InterruptedException {
        Thread thread1 = new Thread(Synchronization::from1To2);
        Thread thread2 = new Thread(Synchronization::from2To1);

        thread1.start(); thread2.start();
        thread1.join(); thread2.join();
        System.out.println(balance1 + ", " + balance2);
    }

    private static void from1To2() {
        for (int i = 0; i < 10000; i++) {
            balance1 -= 100;
            balance2 += 100;
        }
    }

    private static void from2To1() {
        for (int i = 0; i < 10000; i++) {
            balance2 -= 100;
            balance1 += 100;
        }
    }
}
```

Principles of Software Construction: Objects, Design, and Concurrency

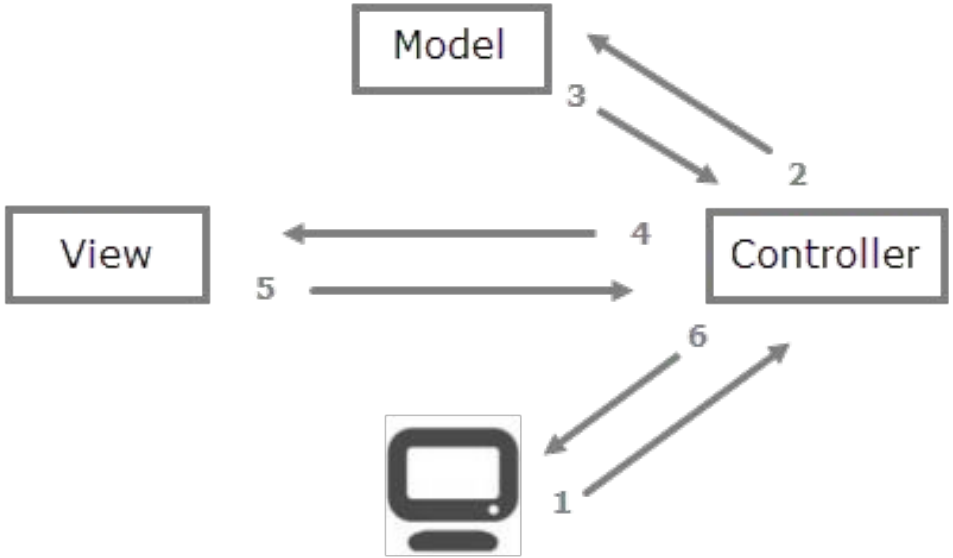
Events Everywhere!

Claire Le Goues Bogdan Vasilescu

Carnegie Mellon University
School of Computer Science

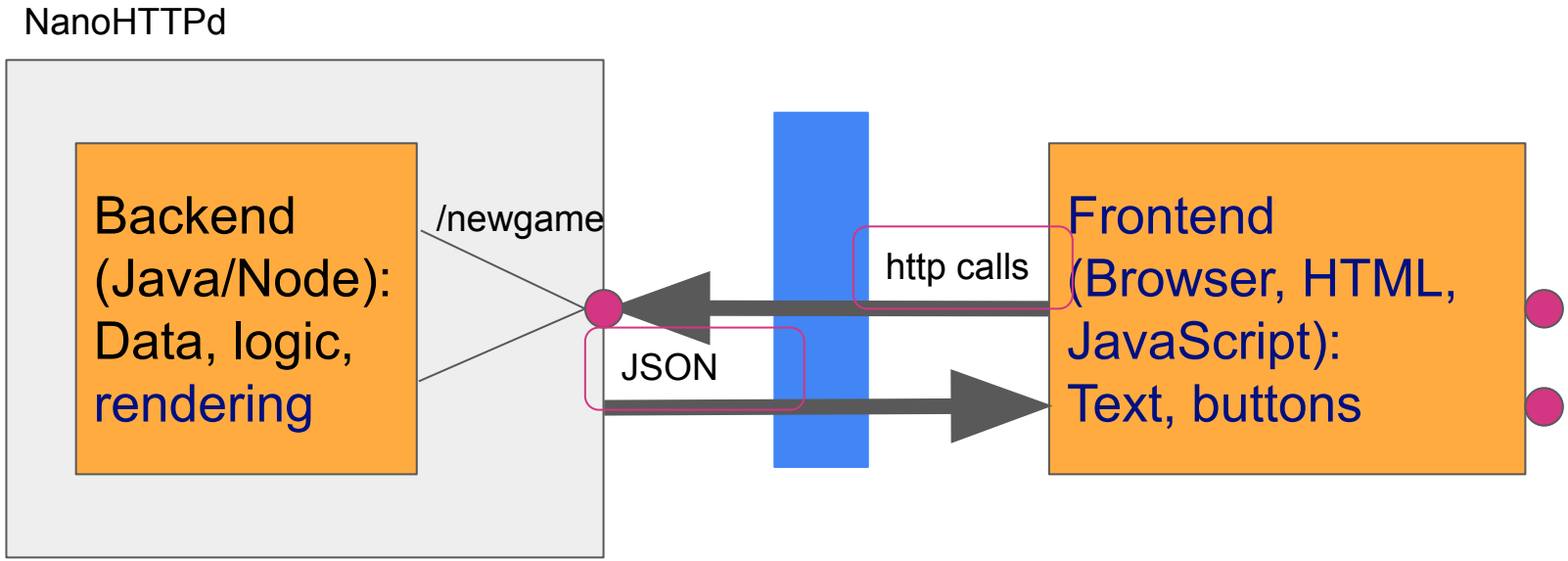
institute for SOFTWARE RESEARCH

Model View Controller in Santorini?



https://overiq.com/django-1-10/mvc-pattern-and-django/

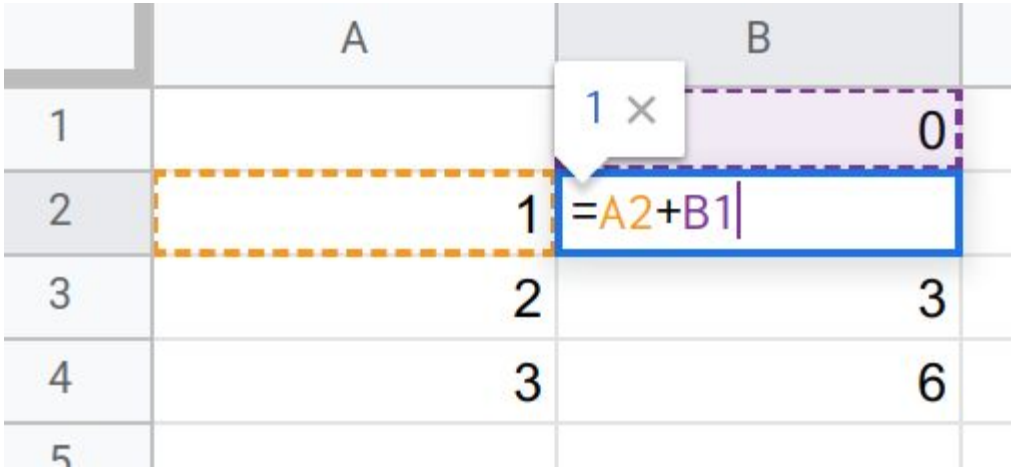
TicTacToe


NanoHTTPd
Backend (Java/Node): Data, logic, rendering
/newgame
http calls
JSON
Frontend (Browser, HTML, JavaScript): Text, buttons

Useful analogy: Spreadsheets

Cells contain data or formulas

Formula cells are computed automatically whenever input data changes



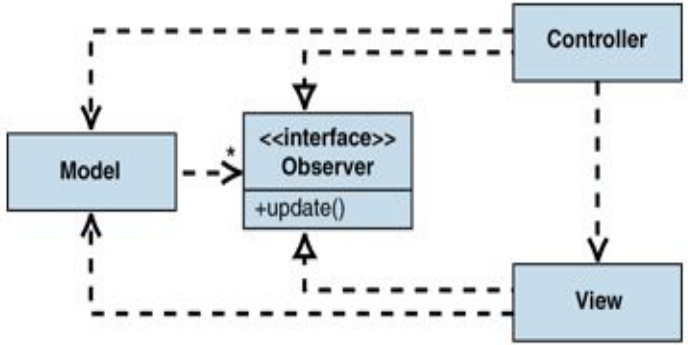
Reactive Programming and GUIs

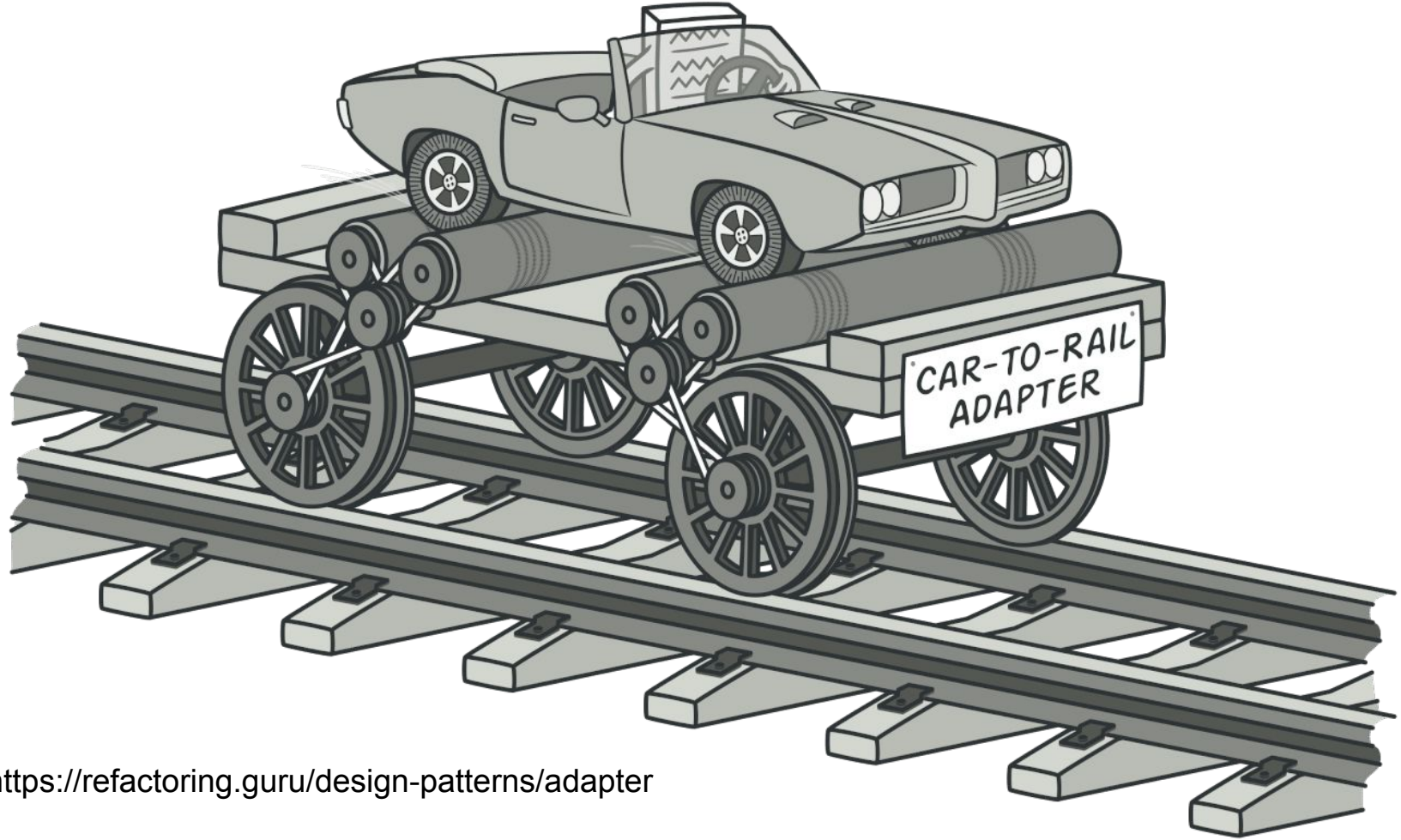
Store state in observable cells, possibly derived

Have GUI update automatically on state changes

Have buttons perform state changes on cells

Mirrors active model-view-controller pattern, discussed later
(model is observable cell)





https://refactoring.guru/design-patterns/adapter

Adapters for Collections/Streams/Observables

```
var lines = IOHelper.readLinesFromFile(file);
var linesObs = Observable.fromIterable(lines);
linesObs.
       map(Parser::getURLColumn).
       groupBy(...).
       sorted(comparator).
       subscribe(IOHelper.writeToFile(outFile));
```

Any others?

Principles of Software Construction: Objects, Design, and Concurrency

Immutability, Promises, Patterns

Claire Le Goues Bogdan Vasilescu

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, **Design Patterns**, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	**Promises/ Reactive P.** ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

Immutable?

Inner mutable state (List in Java)

Create copy of mutable object (new ArrayList(old) in Java)

Return new immutable object

```
class Stack {
    readonly #inner: any[]
    constructor (inner: any[]) {
        this.#inner=inner
    }
    push(o: any): Stack {
        const newInner = this.#inner.slice()
        newInner.push(o)
        return new Stack(newInner)
    }
    peek(): any {
        return this.#inner[this.#inner.length-1]
    }
    getInner(): any[] {
        return this.#inner
    }
}
```

A simple function

...in sync world

```
function copyFileSync(source: string, dest: string) {
    // Stat dest.
    try {
        fs.statSync(dest);
    } catch {
        console.log("Destination already exists")
        return;
    }

    // Open source.
    let fd;
    try {
        fd = fs.openSync(source, 'r');
    } catch {
        console.log("Destination already exists")
        return;
    }

    // Read source.
    let buff = Buffer.alloc(1000)
    try {
        fs.readSync(fd, buff, 0, 0, 1000);
    } catch (_) {
        console.log("Could not read source file")
        return;
    }

    // Write to dest.
    try {
        fs.writeFileSync(dest, buff)
    } catch (_) {
        console.log("Failed to write to dest")
    }
```

Event Handling in JS: Callback Hell

What if our callbacks need callbacks?

```
// Callback Hell


a(function (resultsFromA) {
    b(resultsFromA, function (resultsFromB) {
        c(resultsFromB, function (resultsFromC) {
            d(resultsFromC, function (resultsFromD) {
                e(resultsFromD, function (resultsFromE) {
                    f(resultsFromE, function (resultsFromF) {
                        console.log(resultsFromF);
                    })
                })
            })
        })
    })
});

```

```
function makeRangeIterator(start = 0, end = Infinity, step =
1) {
    let nextIndex = start;
    let iterationCount = 0;

    const rangeIterator = {
       next: function() {
           let result;
           if (nextIndex < end) {
               result = { value: nextIndex, done: false }
               nextIndex += step;
               iterationCount++;
               return result;
           }
           return { value: iterationCount, done: true }
       }
    };
    return rangeIterator;
}
```

Tradeoffs?

```
function* makeRangeIterator(start = 0, end = 100, step = 1) {
    let iterationCount = 0;
    for (let i = start; i < end; i += step) {
        iterationCount++;
        yield i;
    }
    return iterationCount;
}
```

Observer vs. Generator

Push vs. Pull

- In Observer, the publisher controls information flow
  - When it pushes, everyone must listen
- In generators, the listener “pulls” elements
  - Generator may only prepare the next element upon/after pull
- Which is better?
  - Generators are in a sense ‘observers’ to their clients.
  - This inversion of control can make flow management easier

HW5: Santorini with God Cards and GUI

Principles of Software Construction: Objects, Design, and Concurrency

Libraries and Frameworks

(Design for large-scale reuse)

Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	**Frameworks and Libraries** ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, Design Patterns, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

Earlier in this course: **Class-level** reuse

Language mechanisms supporting reuse

- Inheritance
- Subtype polymorphism (dynamic dispatch)
- Parametric polymorphism (generics)*

Design principles supporting reuse

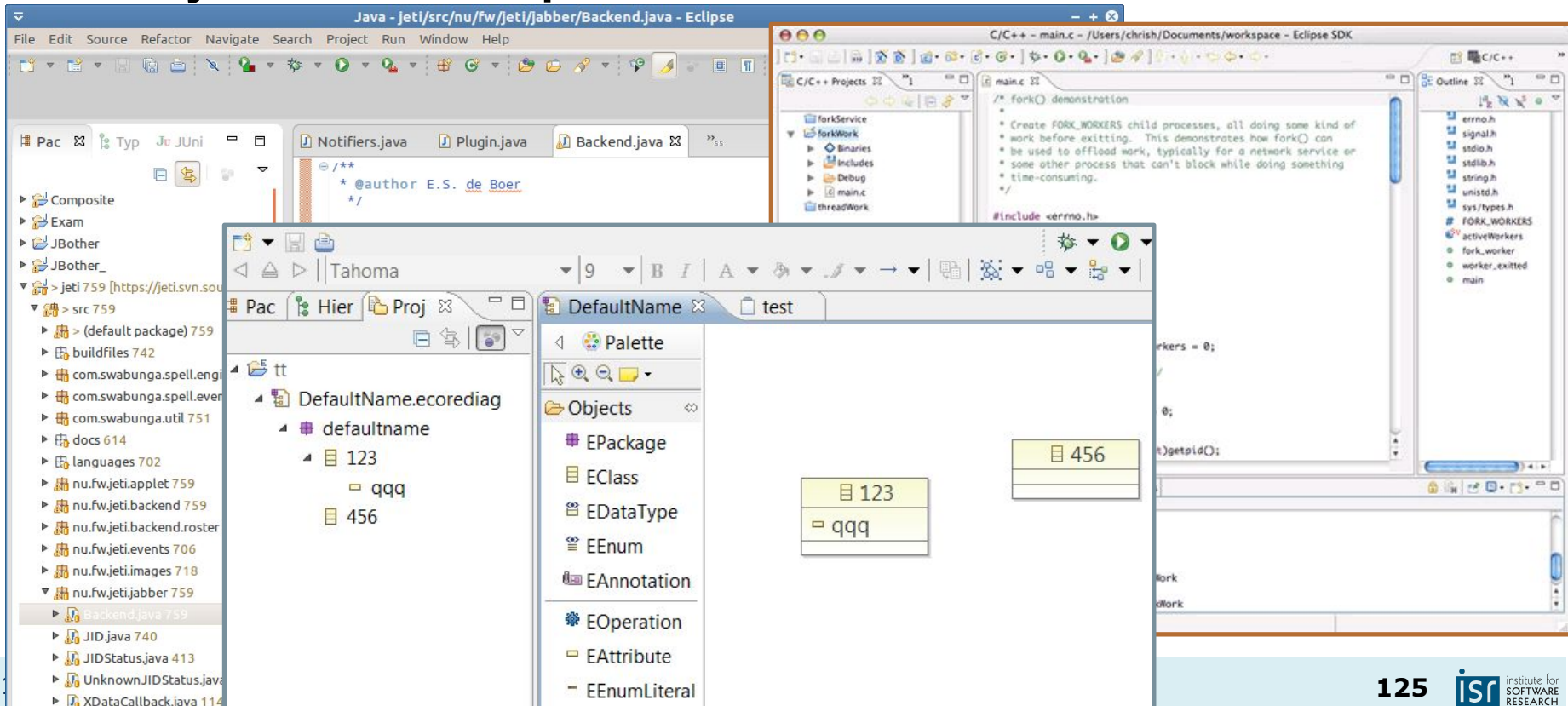
- Small interfaces
- Information hiding
- Low coupling
- High cohesion

Design patterns supporting reuse

- Template method, decorator, strategy, composite, adapter, …

* Effective Java items 26, 29, 30, and 31

Reuse and variation: Family of development tools



General distinction: Library vs. framework

user interacts

```
public MyWidget extends JContainer {

ublic MyWidget(int param) {/ setup
internals, without rendering
}

/ render component on first view and
resizing
protected void
paintComponent(Graphics g) {
// draw a red box on his
componentDimension d = getSize();
g.setColor(Color.red);
g.drawRect(0, 0, d.getWidth(),
d.getHeight());}
}
```

your code

Library

user interacts

Framework

```
public MyWidget extends JContainer {

ublic MyWidget(int param) {/ setup
internals, without rendering
}

/ render component on first view and
resizing
protected void
paintComponent(Graphics g) {
// draw a red box on his
componentDimension d = getSize();
g.setColor(Color.red);
g.drawRect(0, 0, d.getWidth(),
d.getHeight());}
```

your code

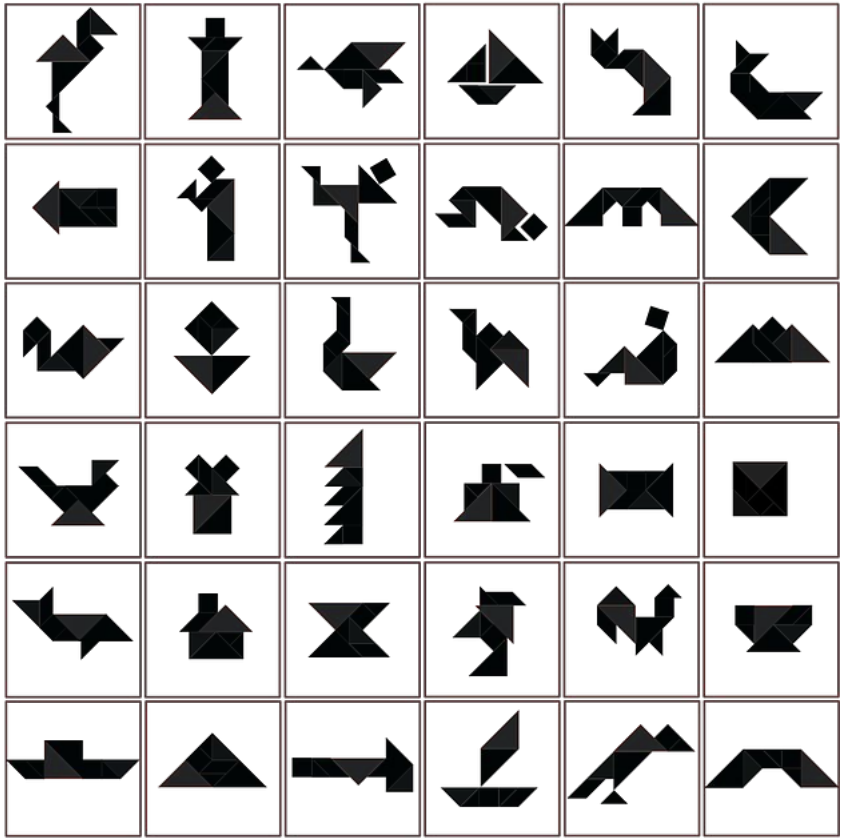
Is this a whitebox or blackbox framework?

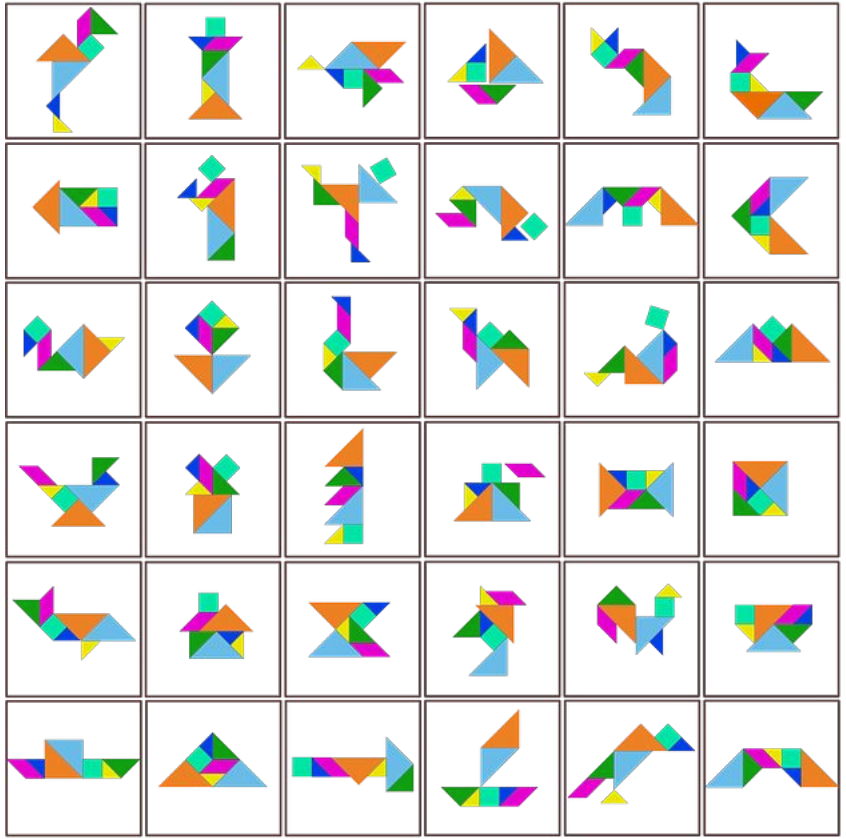
```
public abstract class Application extends JFrame {
  protected String getApplicationTitle() { return ""; }
  protected String getButtonText() { return ""; }
  protected String getInitialText() { return ""; }
```

```
public class Calculator extends Application {
  protected String getApplicationTitle() { return "My Great Calculator"; }
  protected String getButtonText() { return "calculate"; }
  protected String getInititalText() { return "(10 - 3) * 6"; }
  protected void buttonClicked() {
    JOptionPane.showMessageDialog(this, "The result of " + getInput() +
        " is " + calculate(getInput()));
  }
```

```
public class Ping extends Application {
  protected String getApplicationTitle() { return "Ping"; }
  protected String getButtonText() { return "ping"; }
  protected String getInititalText() { return "127.0.0.1"; }
  protected void buttonClicked() { ... }
}
```

Tangrams





The use vs. reuse dilemma

- Large rich components are very useful, but rarely fit a specific need
- Small or extremely generic components often fit a specific need, but provide little benefit

“maximizing reuse minimizes use”

C. Szyperski

The cost of changing a framework

Consider adding an extra method.
Requires changes to *all* plugins!

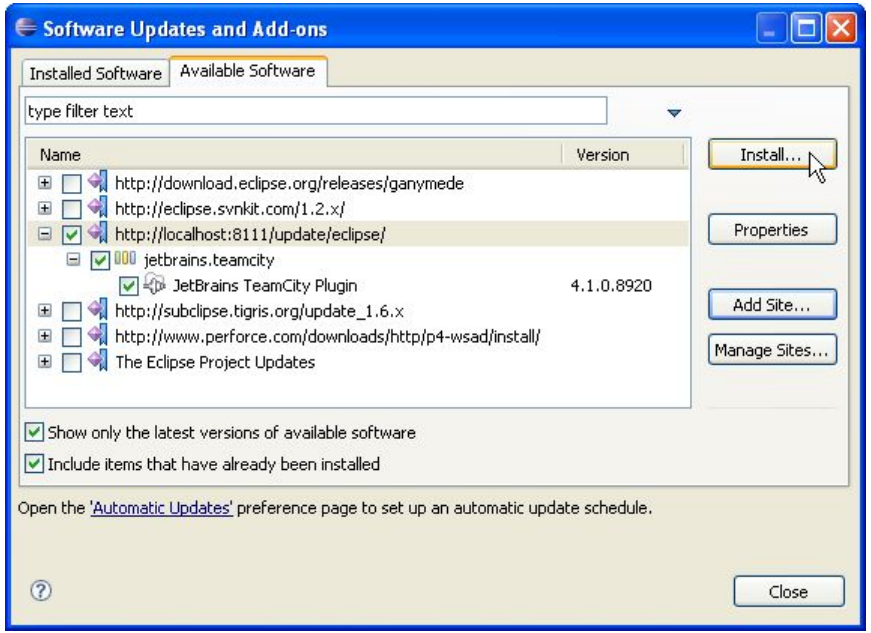
```
public class Application extends JFrame {
        private JTextField textfield;
        private Plugin plugin;
        public Application(Plugin p) { this.plugin=p; p.setA
        protected void init() {
            JPanel contentPane = new JPanel(new BorderLayou
            contentPane.setBorder(new BevelBorder(BevelBord
            JButton button = new JButton();
            if (plugin != null)
                button.setText(plugin.getButtonText());
            else
                butto
            contentPane
            textfield =
            if (plugin
                textf
            textfield.s
            this.setCon
```

```
public interface Plugin {
    String getApplicationTitle();
    String getButtonText();
    String getInititalText();
    void buttonClicked() ;
    void setApplication(Application app);
}
```

```
public class CalcPlugin implements
    private Application application;
    public void setApplication(Application app) { this.application = app; }
    public String getButtonText() { return "calculate"; }
    public String getInititalText() { return "10 / 2 + 6"; }
    public void buttonClicked() {
    JOptionPane.showMessageDialog(null, "The result of "
                                  ut() + " is "
                                  on.getText())); }
                                  e() { return "My Great Calculator"; }
}
```

```
class CalcStarter {  public static void main(String[] args) {
        new Application(new CalcPlugin()).setVisible(true); }}
```

GUI-based plugin management



Principles of Software Construction

API Design

Claire Le Goues Bogdan Vasilescu
(Many slides originally from Josh Bloch)

Carnegie Mellon University
School of Computer Science

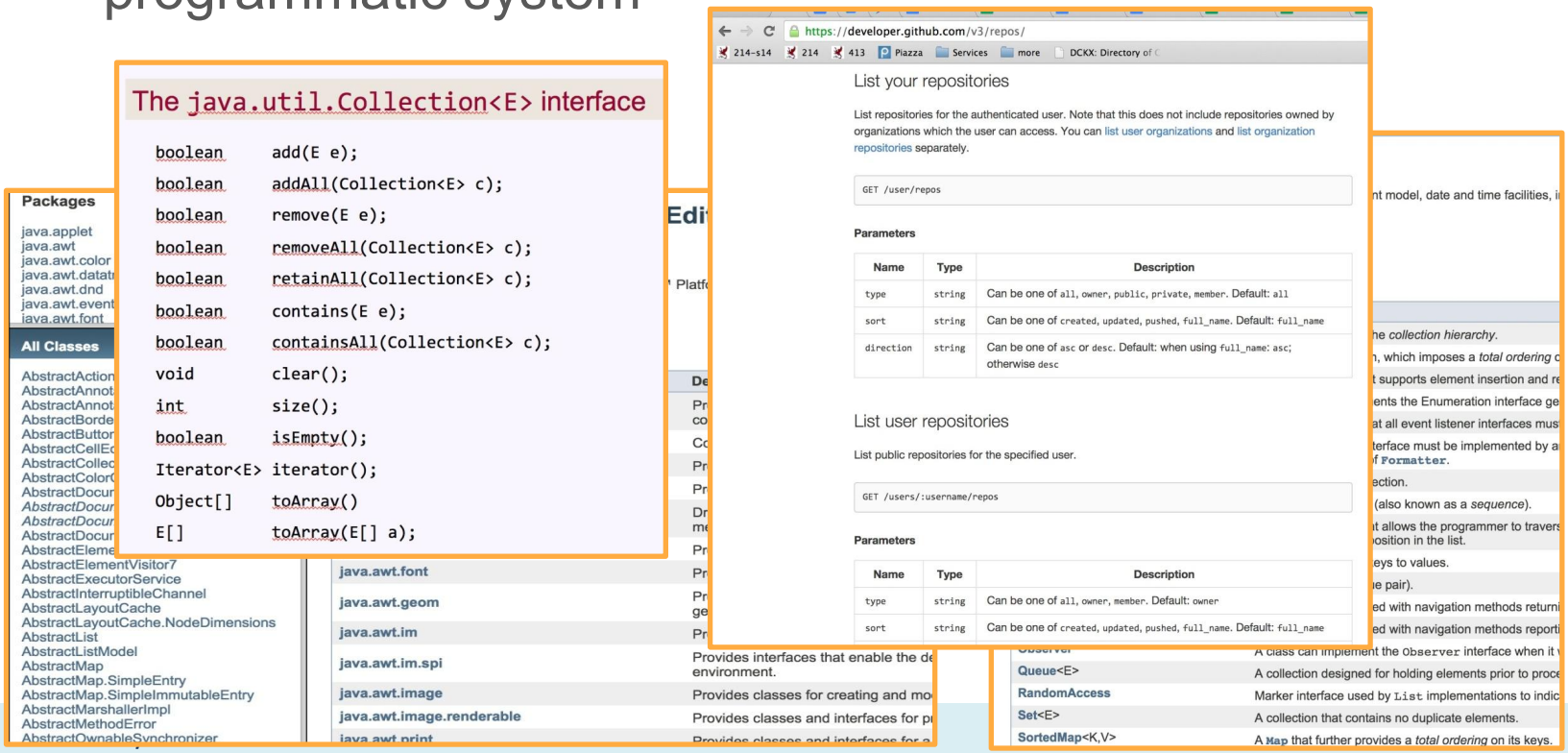
institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism	Domain Analysis	GUI vs Core
change/ext.	Information Hiding, Contracts	Inheritance & Deleg.	**Frameworks** and **Libraries**, **APIs**
reuse	Immutability	Responsibility Assignment, Design Patterns, Antipattern	Module systems, microservices
robustness	Types	Promises/Reactive P.	Testing for Robustness
...	Unit Testing	Integration Testing	CI, DevOps, Teams

API: Application Programming Interface

- An API defines the boundary between components/modules in a programmatic system



Hyrum's Law

"With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody."

https://www.hyrumslaw.com/

https://xkcd.com/1172/

The process of API design – 1-slide version

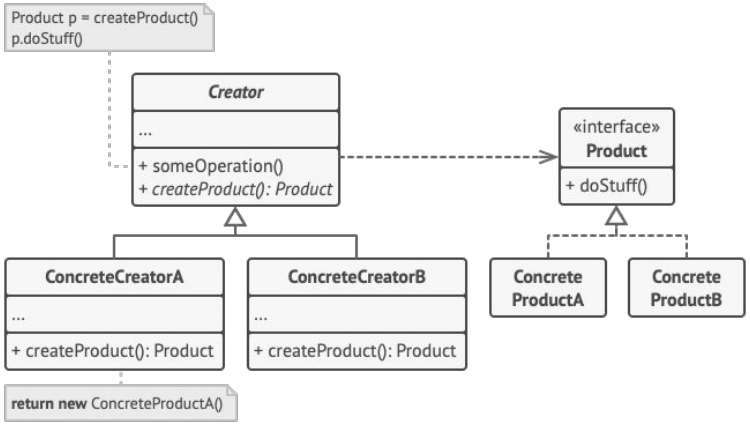
Not sequential; if you discover shortcomings, iterate!

1. **Gather requirements** skeptically, including *use cases*
2. **Choose an abstraction** (model) that appears to address use cases
3. **Compose a short API sketch** for abstraction
4. **Apply API sketch to use cases** to see if it works
   - If not, go back to step 3, 2, or even 1
5. **Show API** to anyone who will look at it
6. **Write prototype** implementation of API
7. **Flesh out** the documentation & harden implementation
8. **Keep refining it** as long as you can

Sample Early API Draft

```
// A collection of elements (root of the collection hierarchy)
public interface Collection<E> {

    // Ensures that collection contains o
    boolean add(E o);

    // Removes an instance of o from collection, if present
    boolean remove(Object o);

    // Returns true iff collection contains o
    boolean contains(Object o);

    // Returns number of elements in collection
    int size();

    // Returns true if collection is empty
    boolean isEmpty();

    ...  // Remainder omitted
}
```

Aside: The *Factory Method* Design Pattern



+ Object creation separated from object
+ Able to hide constructor from clients, control object creation
+ Able to entirely hide implementation objects, only expose interfaces + factory
+ Can swap out concrete class later
+ Can add caching (e.g. `Integer.from()`)
+ Descriptive method name possible

- Extra complexity
- Harder to learn API and write code

From: https://refactoring.guru/design-patterns/factory-method

Principles of Software Construction

API Design (Part 2)

Claire Le Goues **Bogdan Vasilescu**
(With slides from Josh Bloch & Christian Kästner)

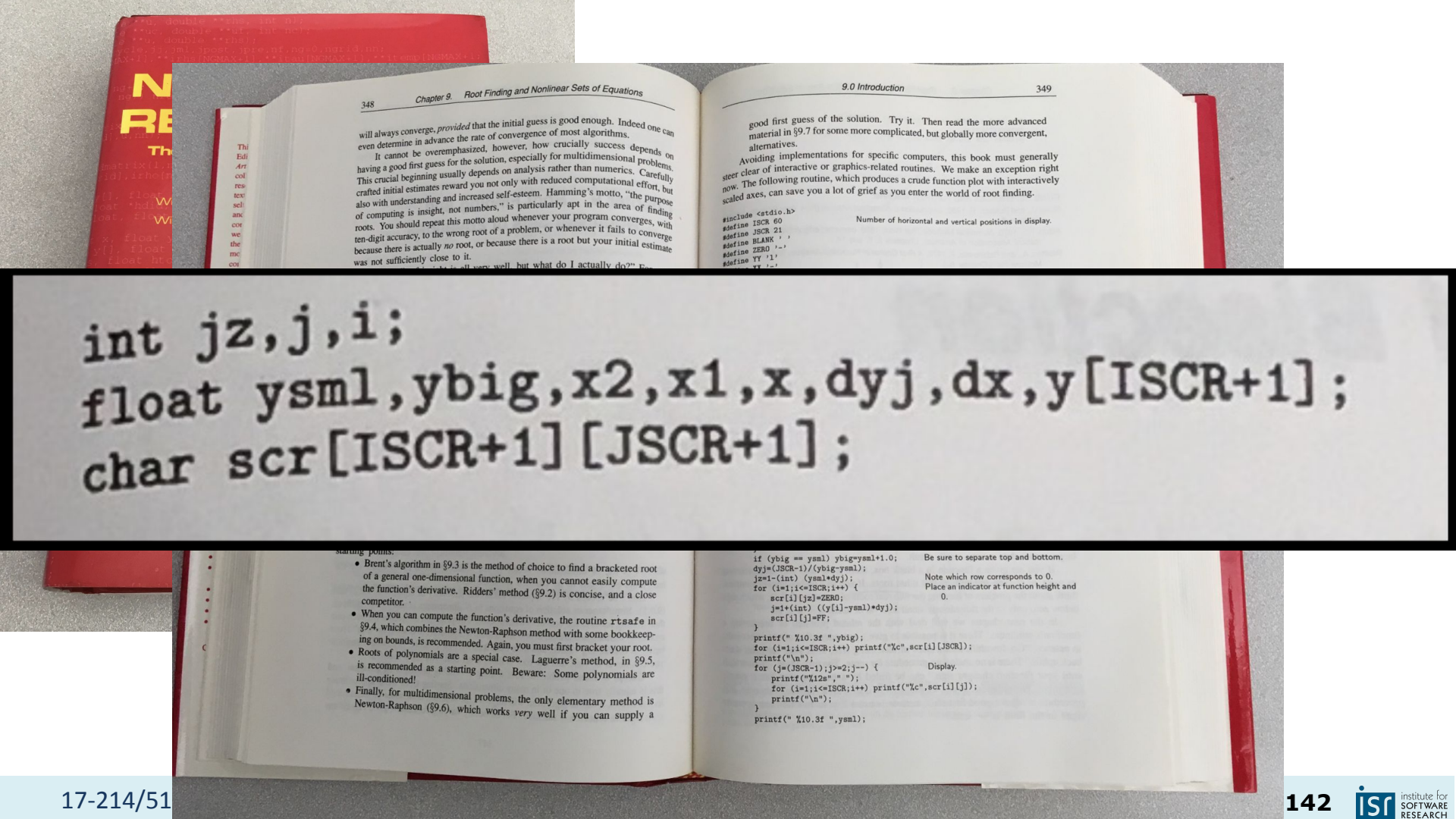
Principle: Minimize conceptual weight

- API should be as small as possible but no smaller
  - **When in doubt, leave it out**
- Conceptual weight: How many concepts must a programmer learn to use your API?
  - APIs should have a "high power-to-weight ratio"

Boilerplate Code

```
import org.w3c.dom.*;
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;

/** DOM code to write an XML document to a specified output stream. */
static final void writeDoc(Document doc, OutputStream out) throws IOException{
  try {
    Transformer t = TransformerFactory.newInstance().newTransformer();
    t.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM, doc.getDoctype().getSystemId());
    t.transform(new DOMSource(doc), new StreamResult(out)); // Does actual writing
  } catch(TransformerException e) {
    throw new AssertionError(e);  // Can't happen!
  }
}
```

- Generally done via cut-and-paste
- Ugly, annoying, and error-prone


348 Chapter 9. Root Finding and Nonlinear Sets of Equations
will always converge, *provided* that the initial guess is good enough. Indeed one can even determine in advance the rate of convergence of most algorithms.
It cannot be overemphasized, however, how crucially success depends on having a good first guess for the solution, especially for multidimensional problems. This crucial beginning usually depends on analysis rather than numerics. Carefully crafted initial estimates reward you not only with reduced computational effort, but also with understanding and increased self-esteem. Hamming's motto, "the purpose of computing is insight, not numbers," is particularly apt in the area of finding roots. You should repeat this motto aloud whenever your program converges, with ten-digit accuracy, to the wrong root of a problem, or whenever it fails to converge because there is actually *no* root, or because there is a root but your initial estimate was not sufficiently close to it.
9.0 Introduction 349
good first guess of the solution. Try it. Then read the more advanced material in §9.7 for some more complicated, but globally more convergent, alternatives.
Avoiding implementations for specific computers, this book must generally steer clear of interactive or graphics-related routines. We make an exception right now. The following routine, which produces a crude function plot with interactively scaled axes, can save you a lot of grief as you enter the world of root finding.
#include <stdio.h>
#define ISCR 60
#define JSCR 21
#define BLANK ' '
#define ZERO '-'
#define YY 'l'
Number of horizontal and vertical positions in display.
int jz,j,i;
float ysml,ybig,x2,x1,x,dyj,dx,y[ISCR+1];
char scr[ISCR+1][JSCR+1];
starting points.
• Brent's algorithm in §9.3 is the method of choice to find a bracketed root of a general one-dimensional function, when you cannot easily compute the function's derivative. Ridders' method (§9.2) is concise, and a close competitor.
• When you can compute the function's derivative, the routine rtsafe in §9.4, which combines the Newton-Raphson method with some bookkeeping on bounds, is recommended. Again, you must first bracket your root.
• Roots of polynomials are a special case. Laguerre's method, in §9.5, is recommended as a starting point. Beware: Some polynomials are ill-conditioned!
• Finally, for multidimensional problems, the only elementary method is Newton-Raphson (§9.6), which works *very* well if you can supply a
if (ybig == ysml) ybig=ysml+1.0; Be sure to separate top and bottom.
dyj=(JSCR-1)/(ybig-ysml);
jz=1-(int) (ysml*dyj); Note which row corresponds to 0.
for (i=1;i<=ISCR;i++) { Place an indicator at function height and 0.
scr[i][jz]=ZERO;
j=1+(int) ((y[i]-ysml)*dyj);
scr[i][j]=FF;
}
printf(" %10.3f ",ybig);
for (i=1;i<=ISCR;i++) printf("%c",scr[i][JSCR]);
printf("\n");
for (j=(JSCR-1);j>=2;j--) { Display.
printf("%12s"," ");
for (i=1;i<=ISCR;i++) printf("%c",scr[i][j]);
printf("\n");
}
printf(" %10.3f ",ysml);

Principle: Favor composition over inheritance

```
// A Properties instance maps Strings to Strings
public class Properties extends HashTable {
    public Object put(Object key, Object value);
    …
}
public class Properties {
    private final HashTable data = new HashTable();
    public String put(String key, String value) {
        data.put(key, value);
    }
    …
```

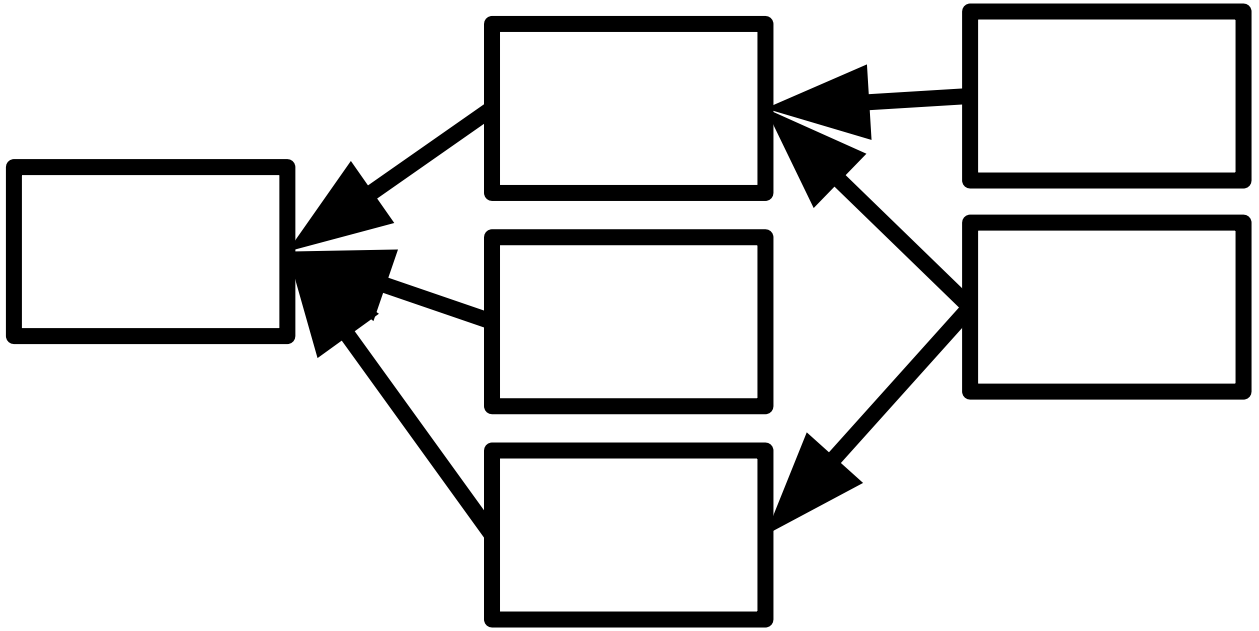
REST API

API of a web service

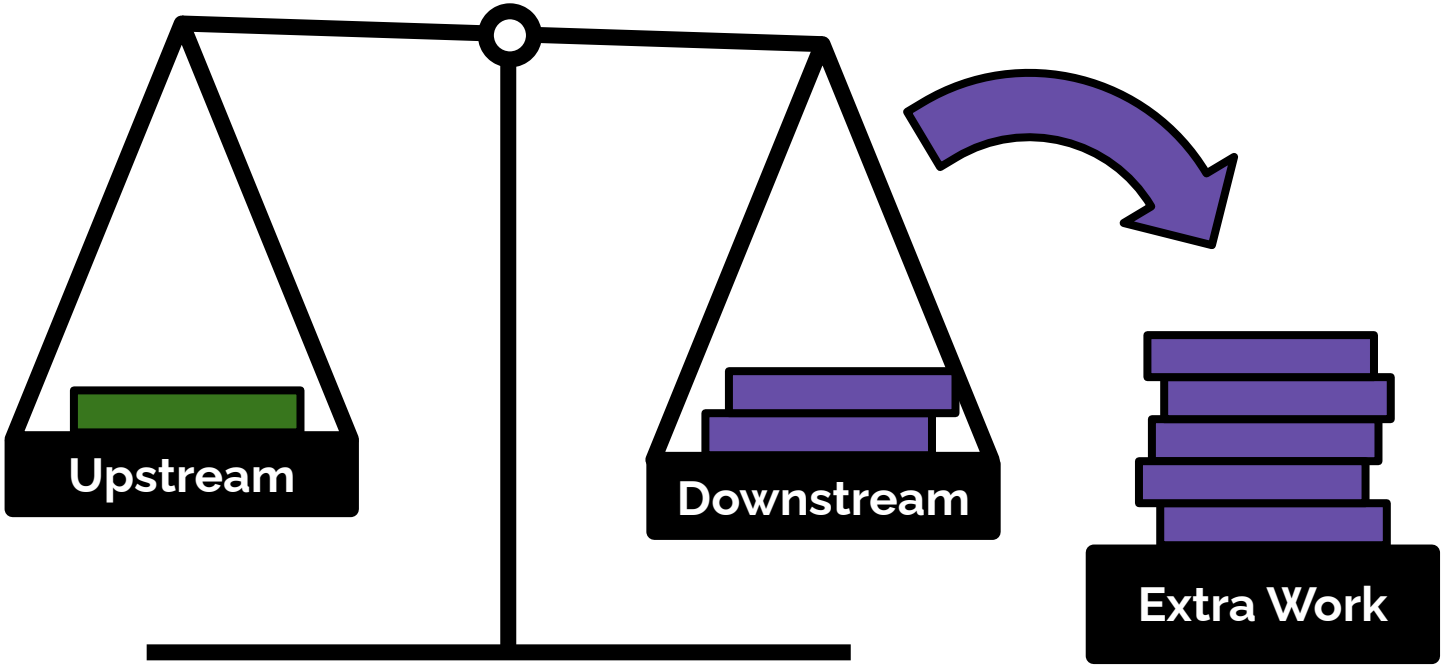
Uniform interface over HTTP requests

- Send parameters to URL, receive data (JSON, XML common)
- Stateless: Each request is self-contained

Language independent, distributed



Software Ecosystem



Avoiding dependencies
Encapsulating from change

HW6: Data Analytics Framework

Principles of Software Construction

Version Control with Git

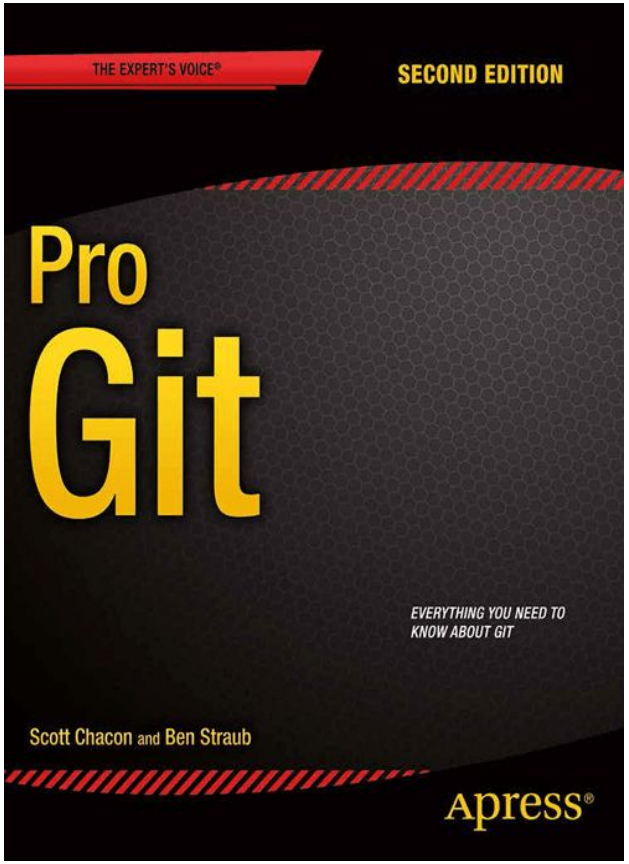
Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Highly recommended

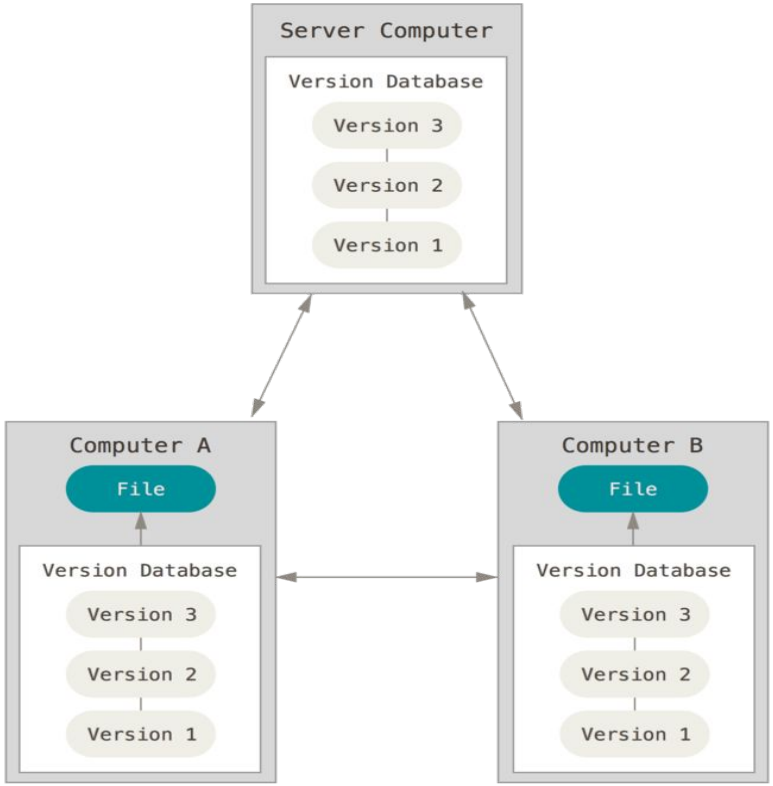
- (second) most useful life skill you will have learned in 214/514



https://git-scm.com/book/en/v2

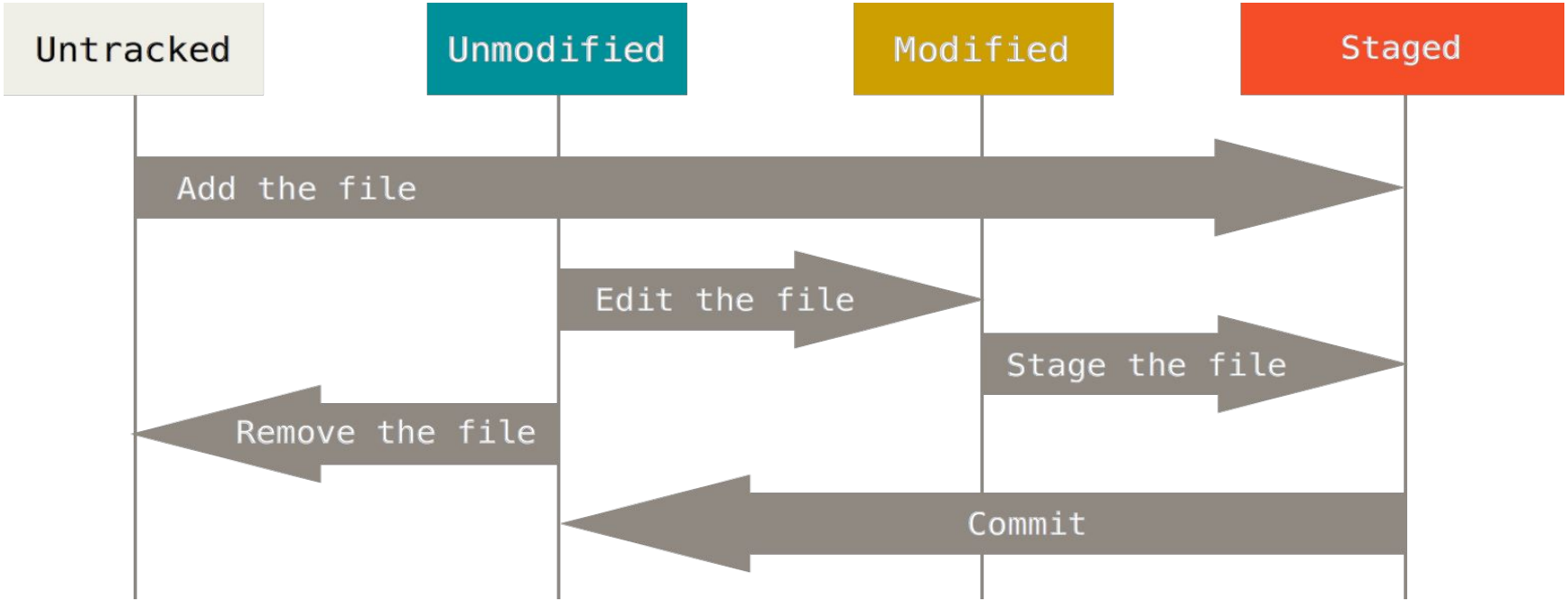
Distributed version control

- Clients fully mirror the repository
  - Every clone is a full backup of ***all*** the data
- E.g., Git, Mercurial, Bazaar



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Aside: Git process



© Scott Chacon “Pro Git”

Principles of Software Construction

Version Control in the Wild

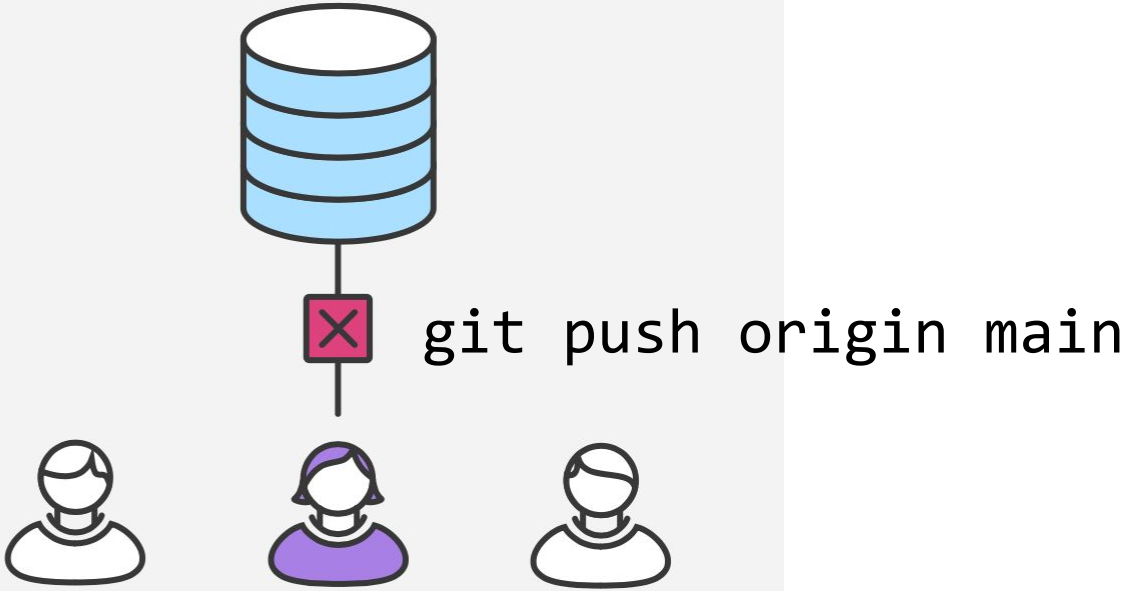
Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

```
error: failed to push some refs to '/path/to/repo.git'
hint: Updates were rejected because the tip of your current branch is behind its
remote counterpart. Merge the remote changes (e.g. 'git pull') before pushing again.
See the 'Note about fast-forwards' in 'git push --help' for details.
```

Mary tries to publish her feature

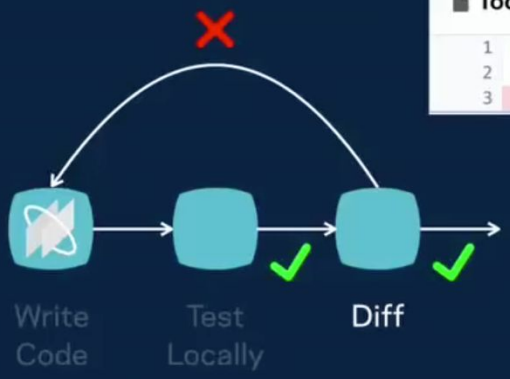


Semantic Versioning

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards compatible manner, and
3. PATCH version when you make backwards compatible bug fixes.

Diff lifecycle: local testing



Tools/xctool/xctool/xctool/Version.m

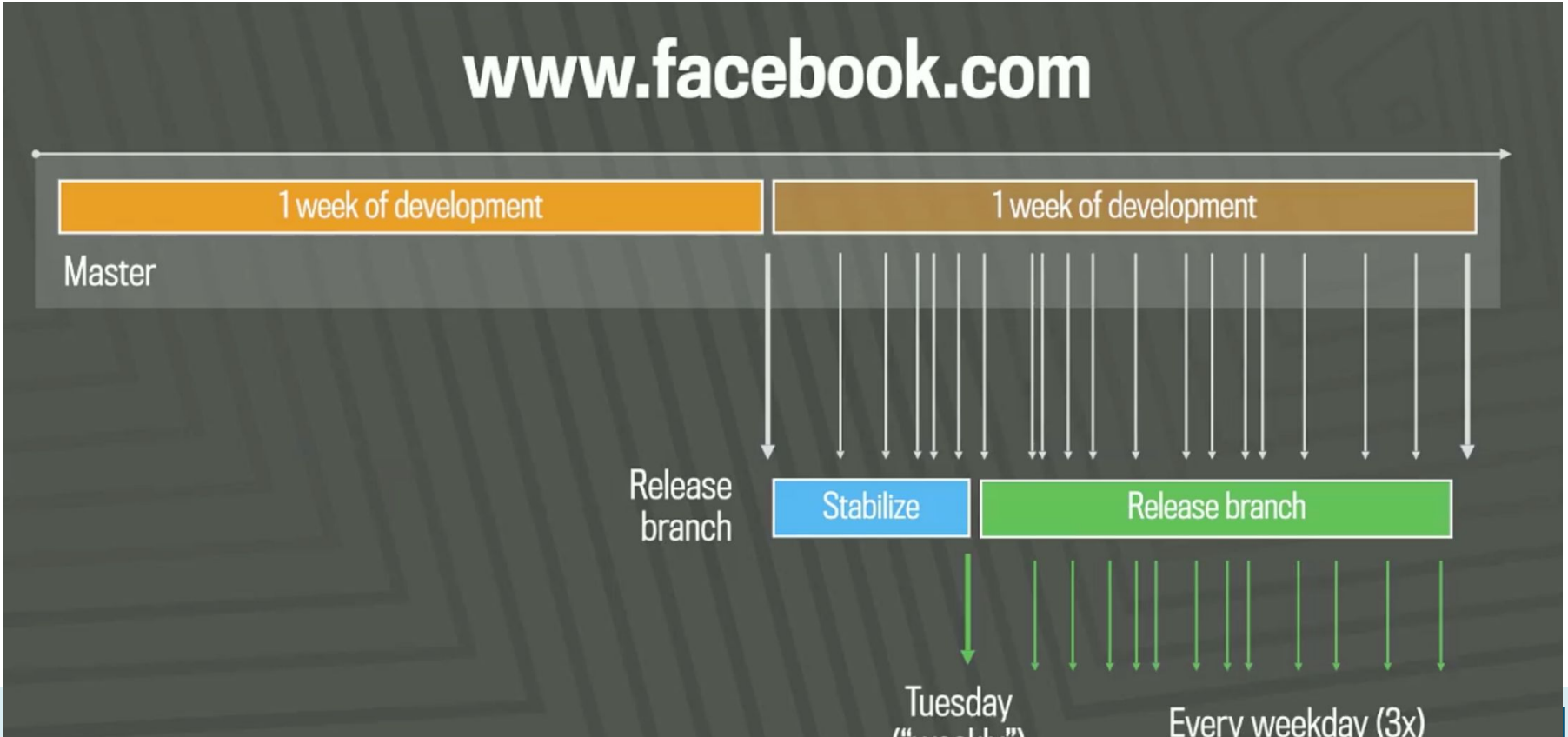
View Options

```
1  #import "Version.h"                          1  #import "Version.h"
2                                               2
3  NSString * const XCToolVersionString = @"0.2.1";   3  NSString * const XCToolVersionString = @"0.2.2";
```

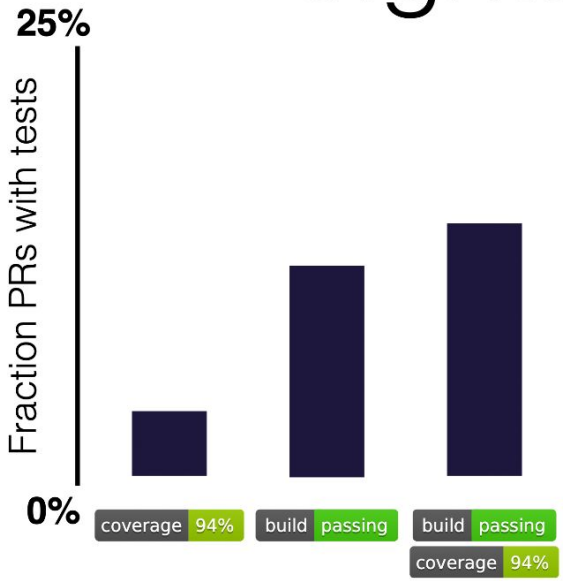
```
        PASS ExampleTest (0.050s)
.

OK (1 test, 4 assertions)

OK
(1 tests, 4 assertions, 0 incomplete, 0 failures)
```

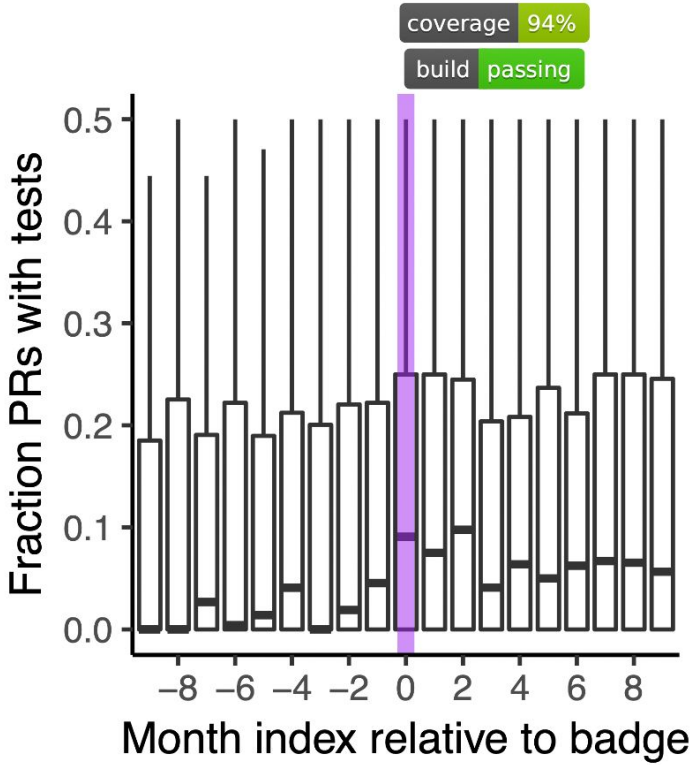
Test and lint locally

Release every two weeks



Signals of PR quality



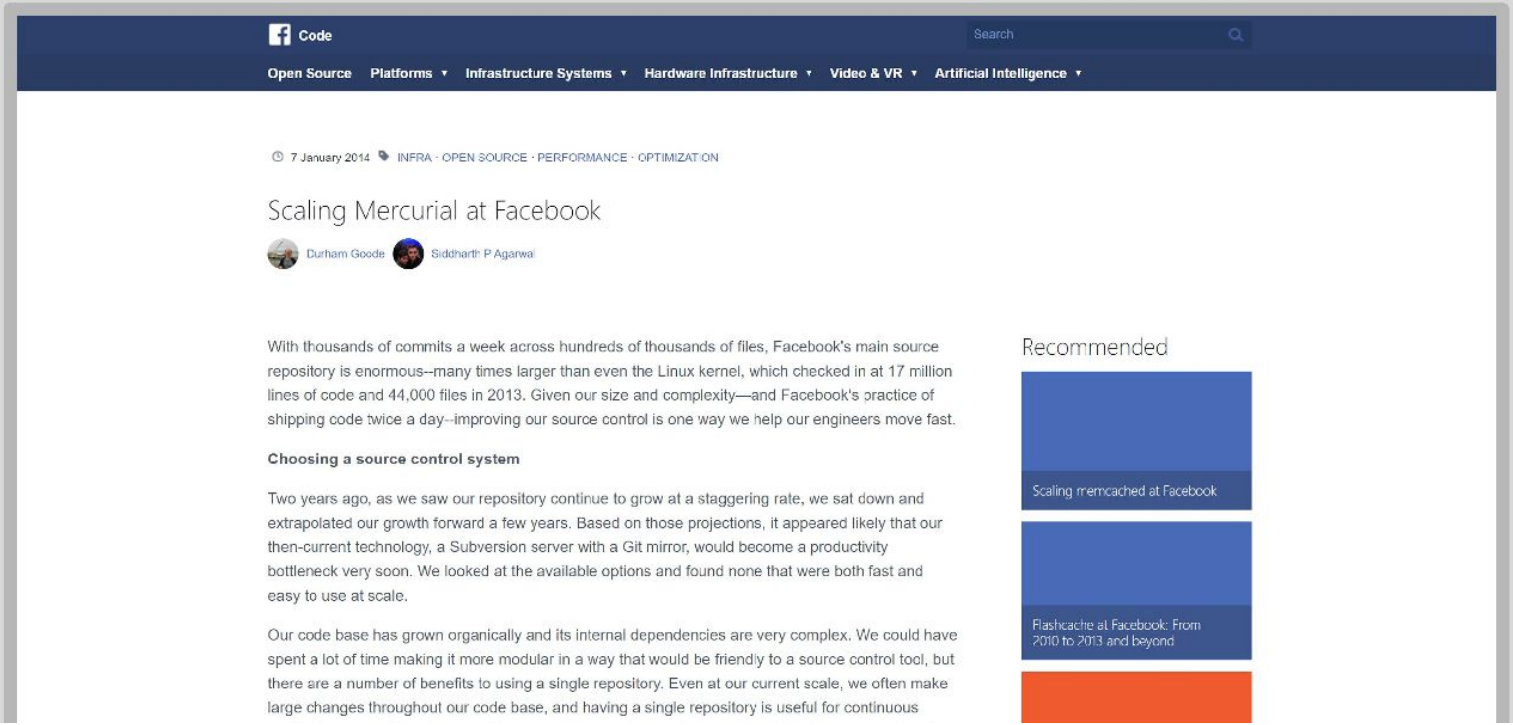


Result: Build status+code coverage badges indicate *more tests in PRs*

STRUDEL
Carnegie Mellon University

Monorepos in industry

Scaling Mercurial at Facebook



Code

Search

Open Source | Platforms | Infrastructure Systems | Hardware Infrastructure | Video & VR | Artificial Intelligence

7 January 2014 · INFRA · OPEN SOURCE · PERFORMANCE · OPTIMIZATION

Scaling Mercurial at Facebook

Durham Goode, Siddharth P Agarwal

With thousands of commits a week across hundreds of thousands of files, Facebook's main source repository is enormous--many times larger than even the Linux kernel, which checked in at 17 million lines of code and 44,000 files in 2013. Given our size and complexity—and Facebook's practice of shipping code twice a day--improving our source control is one way we help our engineers move fast.

Choosing a source control system

Two years ago, as we saw our repository continue to grow at a staggering rate, we sat down and extrapolated our growth forward a few years. Based on those projections, it appeared likely that our then-current technology, a Subversion server with a Git mirror, would become a productivity bottleneck very soon. We looked at the available options and found none that were both fast and easy to use at scale.

Our code base has grown organically and its internal dependencies are very complex. We could have spent a lot of time making it more modular in a way that would be friendly to a source control tool, but there are a number of benefits to using a single repository. Even at our current scale, we often make large changes throughout our code base, and having a single repository is useful for continuous

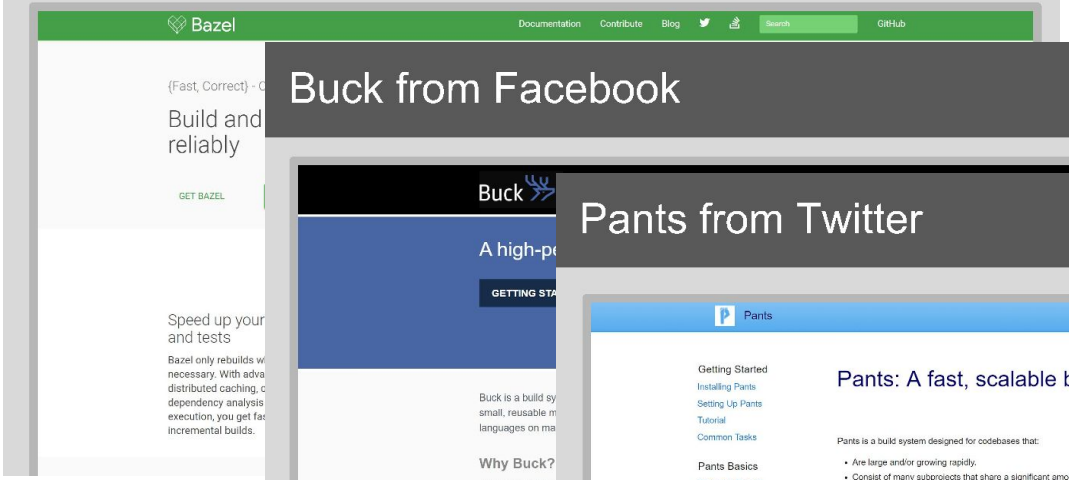
Recommended

Scaling memcached at Facebook

Flashcache at Facebook: From 2010 to 2013 and beyond

Common build system

Bazel from Google



Buck from Facebook

Pants from Twitter

Principles of Software Construction: Objects, Design, and Concurrency

{Static & Dynamic} x {Typing & Analysis}

Claire Le Goues Bogdan Vasilescu

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

How Do You Find Bugs?

- Run it?

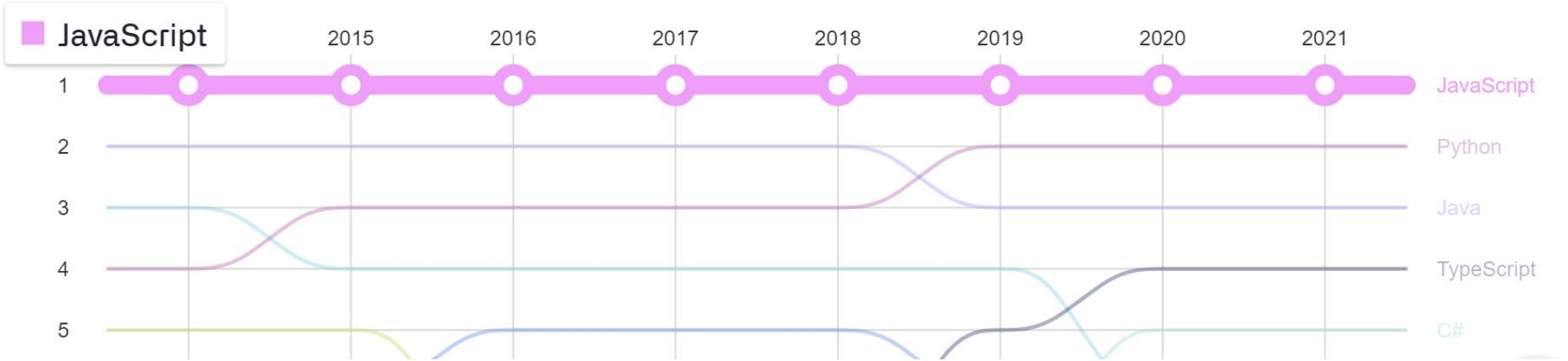
```
public class Fails {
    public static void main(String[] args) {
        getValue( i: null);
    }

    private static int getValue(Integer i) {
        return i.intValue();
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke "java.lang.Integer.intValue()" because "i" is null
    at misc.Fails.getValue(Fails.java:9)
    at misc.Fails.main(Fails.java:5)
```

Static vs. Dynamic Typing

Okay, but:

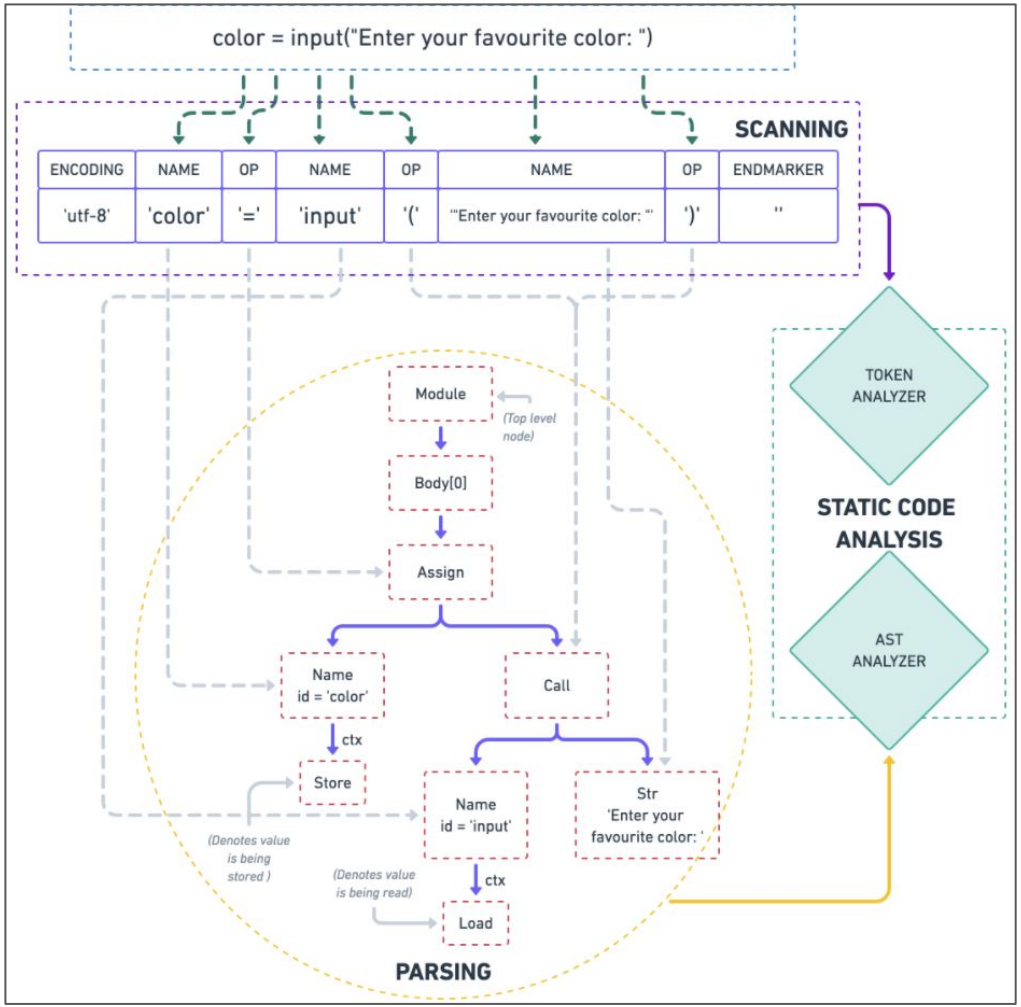
Top languages over the years



https://octoverse.github.com/#geographical-distribution-of-active-users

Static Analysis

- How?
  - Program analysis + Vocabulary of patterns



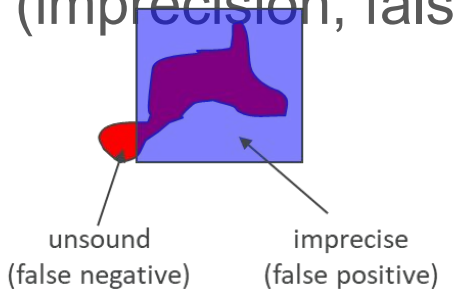
https://deepsource.io/blog/introduction-static-code-analysis/

Soundness & Precision

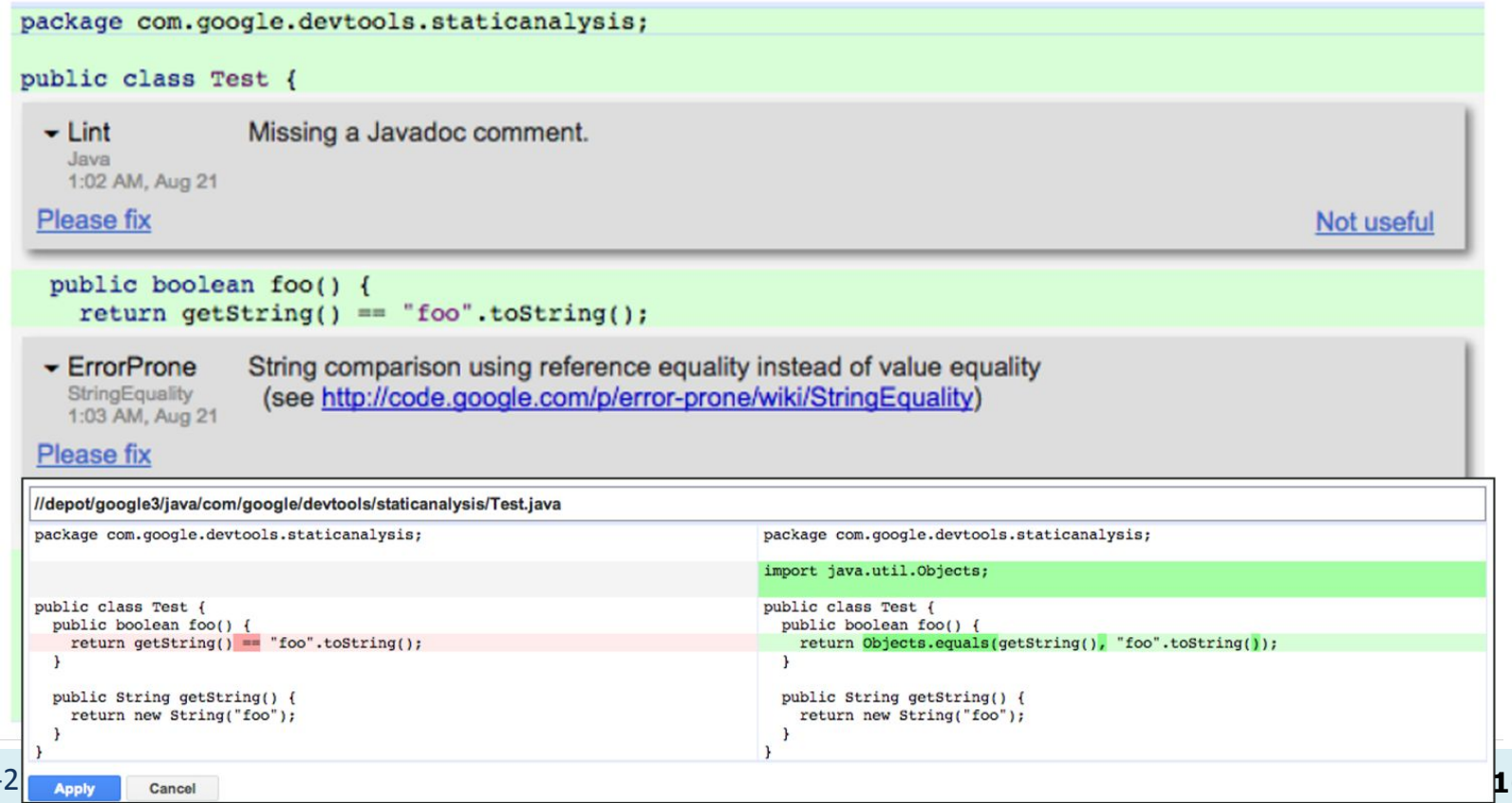
- Since we can't perfectly analyze behavior statically
  - We may miss things by being cautious (unsound; false negative)
  - We might identify non-problems (imprecision, false positive)

Program state covered in actual execution

Program state covered by abstract execution with analysis

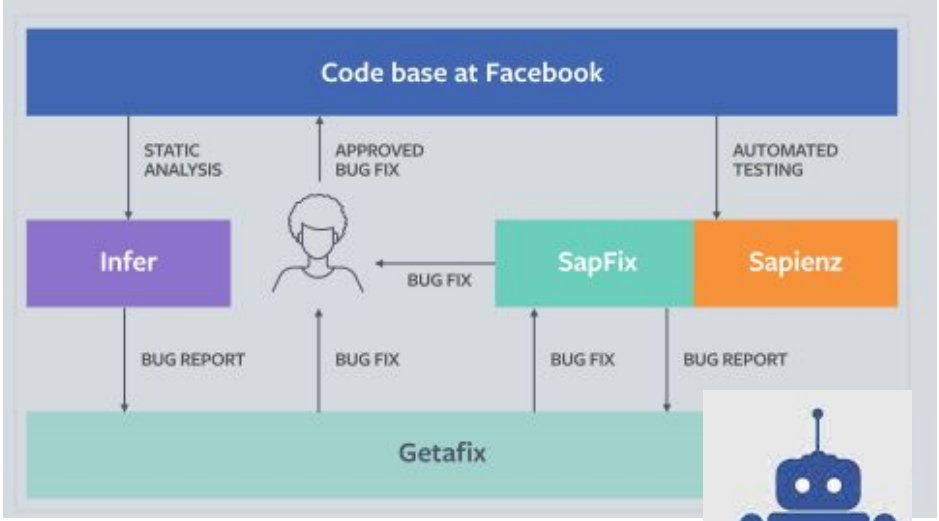


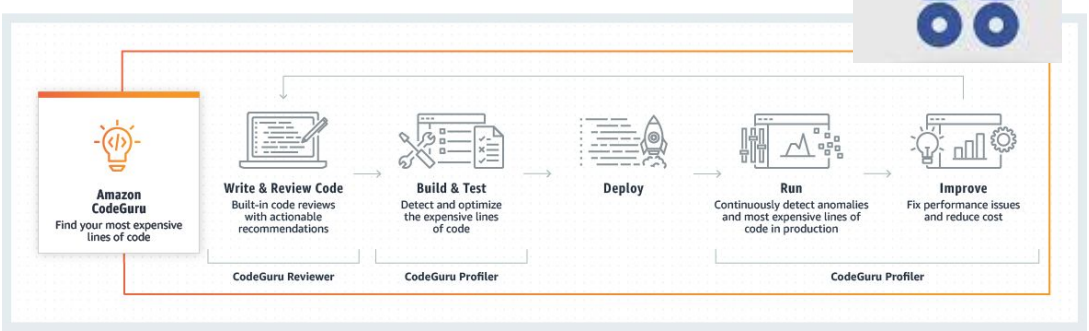
TriCorder


package com.google.devtools.staticanalysis;
public class Test {
Lint
Java
1:02 AM, Aug 21
Missing a Javadoc comment.
Please fix
Not useful
public boolean foo() {
return getString() == "foo".toString();
ErrorProne
StringEquality
1:03 AM, Aug 21
String comparison using reference equality instead of value equality
(see http://code.google.com/p/error-prone/wiki/StringEquality)
Please fix
//depot/google3/java/com/google/devtools/staticanalysis/Test.java
package com.google.devtools.staticanalysis;
public class Test {
public boolean foo() {
return getString() == "foo".toString();
}
public String getString() {
return new String("foo");
}
}
package com.google.devtools.staticanalysis;
import java.util.Objects;
public class Test {
public boolean foo() {
return Objects.equals(getString(), "foo".toString());
}
public String getString() {
return new String("foo");
}
}
Apply
Cancel

What else could we do?

- Use more complicated logic
  - One example: Infer, at Facebook (Google claims this won't (easily) scale to their mono-repo.)
- Use AI?
  - Facebook: Getafix, also integrates with SapFix
  - Amazon: CodeGuru
  - Microsoft: IntelliSense in VSCode, mostly refactoring/code completion, trained on large volumes of code
  - Mostly fairly simple ML (details limited)





Principles of Software Construction: Objects, Design, and Concurrency

A Quick Tour of all 23 GoF Design Patterns

Claire Le Goues **Bogdan Vasilescu**

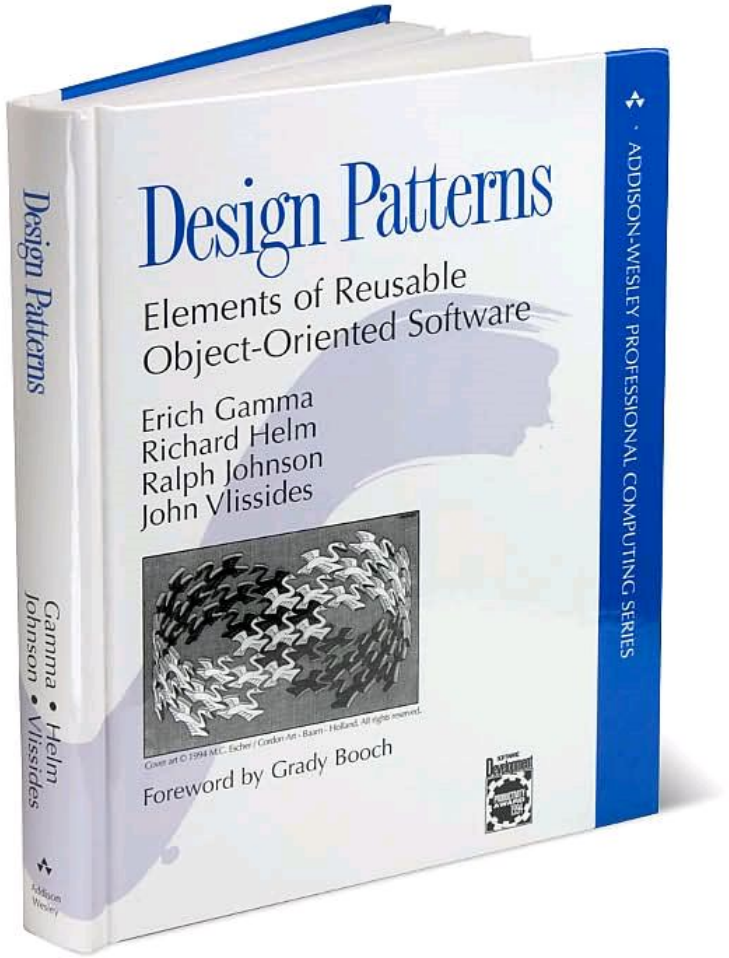
Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, **Design Patterns**, Antipattern ✓	Module systems, microservices ✓
robustness	Types	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps, Teams

- Published 1994
- 23 Patterns
- Widely known



I. Creational Patterns

1. Abstract factory
2. Builder
3. Factory method
4. Prototype
5. Singleton

Singleton Illustration

```
public class Elvis {
    private static final Elvis ELVIS = new Elvis();
    public static Elvis getInstance() { return ELVIS; }
    private Elvis() { }
    ...
}
```

```
const elvis = { … }
function getElvis() {

export { getElvis }
```

II. Structural Patterns

1. Adapter
2. Bridge
3. Composite
4. Decorator
5. Façade
6. Flyweight
7. Proxy

Decorator vs Strategy?

```
interface GameLogic {
     isValidMove(w, x, y)
     move(w, x, y)
}

class BasicGameLogic
     implements GameLogic { … }

class AbstractGodCardDecorator
     implements GameLogic { … }

class PanDecorator
     extends AbstractGodCardDecorator
     implements GameLogic { … }
```

```
interface GameLogic {
     isValidMove(w, x, y)
     move(w, x, y)
}
class BasicGameLogic
          implements GameLogic {
     constructor(board) { … }
     isValidMove(w, x, y) { … }
     move(w, x, y) { … }
}
class PanDecorator
          extends BasicGameLogic {
     move(w, x, y} { /* super.move(w,
x, y) + checkWinner */ }
}
```

III. Behavioral Patterns

1. Chain of Responsibility
2. Command
3. Interpreter
4. Iterator
5. Mediator
6. Memento
7. Observer
8. State
9. Strategy
10. Template method
11. Visitor

Principles of Software Construction: Objects, Design, and Concurrency

Design for Robustness: Distributed Systems

Bogdan Vasilescu **Claire Le Goues**

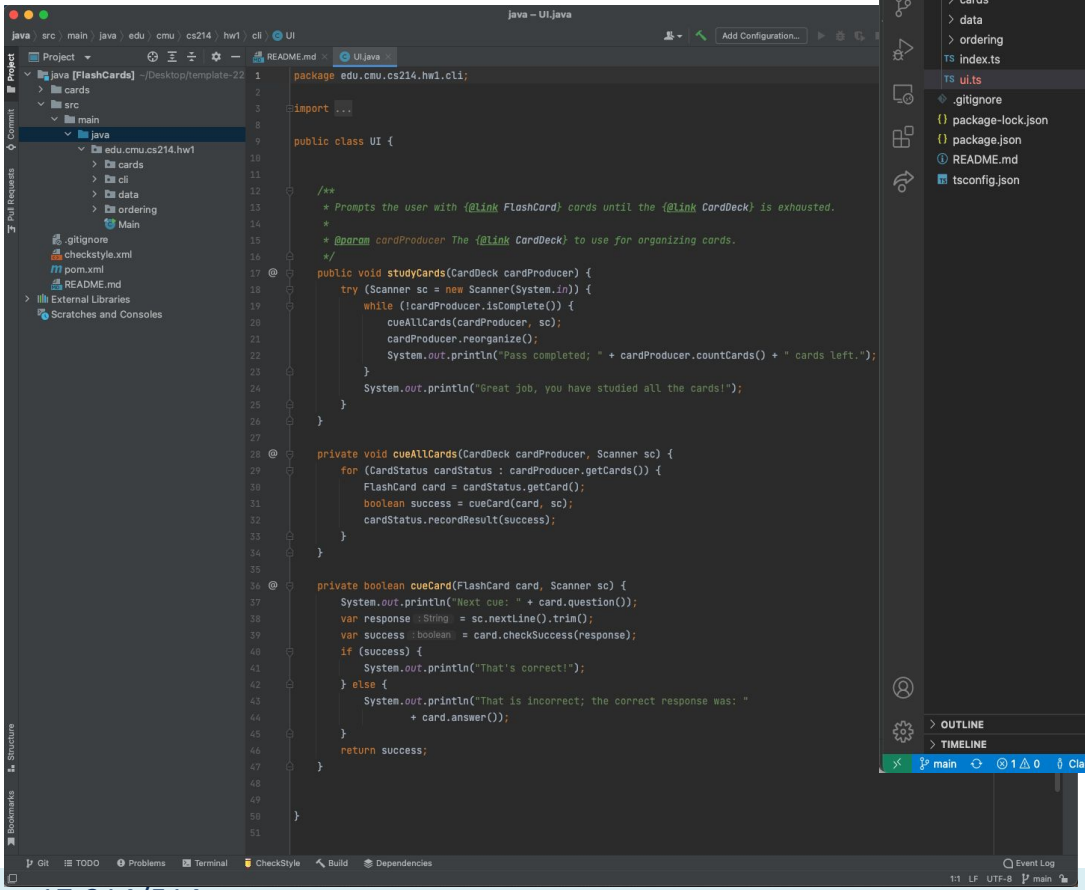
Carnegie Mellon University
School of Computer Science

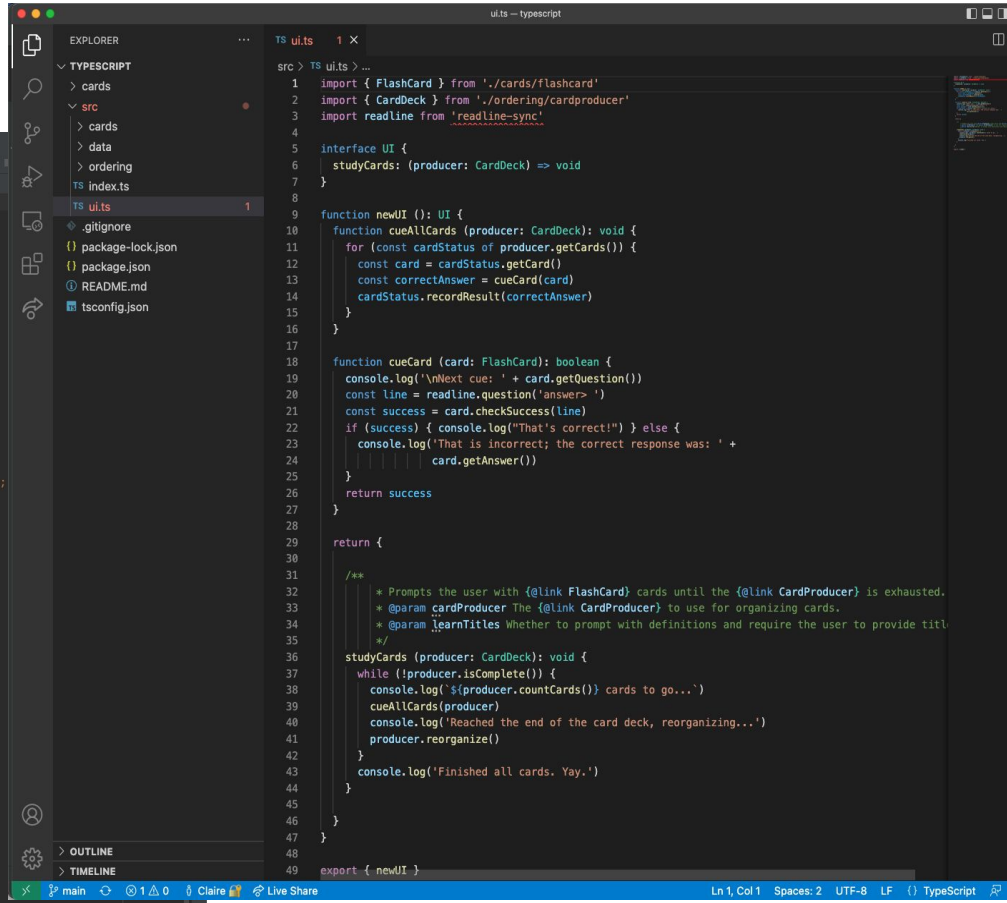
institute for SOFTWARE RESEARCH

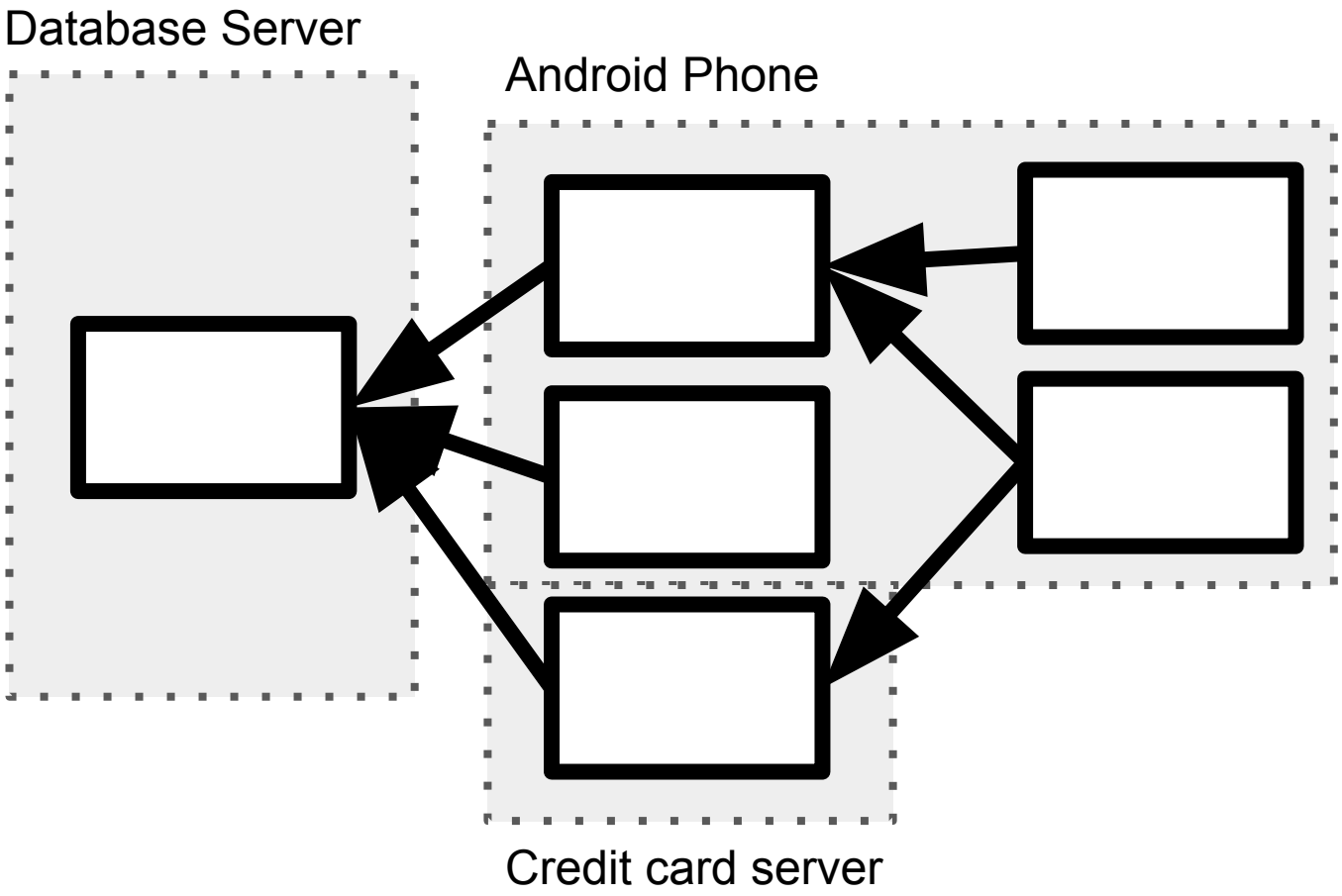
Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, Design Patterns, Antipattern ✓	**Module systems, microservices**
robustness	Types ✓	Promises/ Reactive P. ✓	**(Testing for) Robustness**
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, Teams

Where did we start?



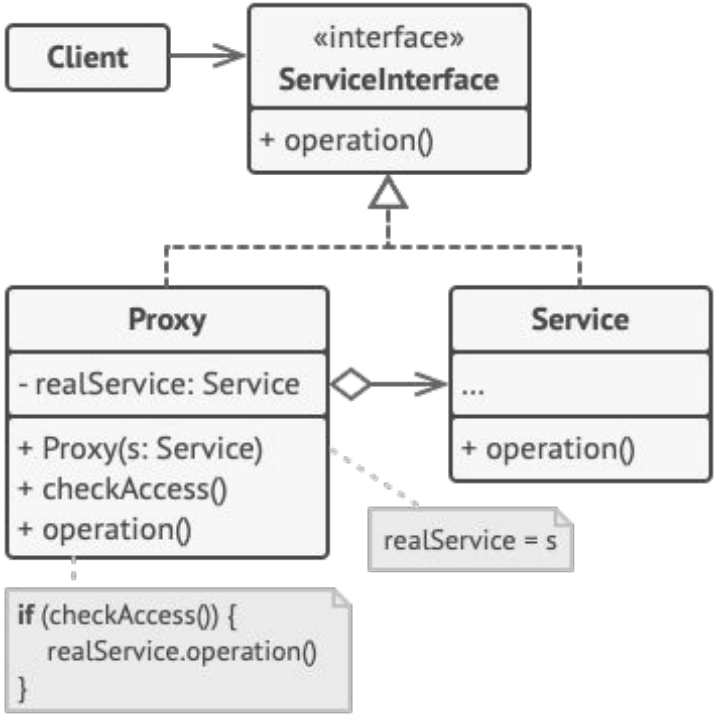



Database Server
Android Phone
Credit card server


Microservices
Microservices Everywhere

Proxy Design Pattern

- Local representative for remote object
  - Create expensive obj on-demand
  - Control access to an object
- Hides extra “work” from client
  - Add extra error handling, caching
  - Uses *indirection*

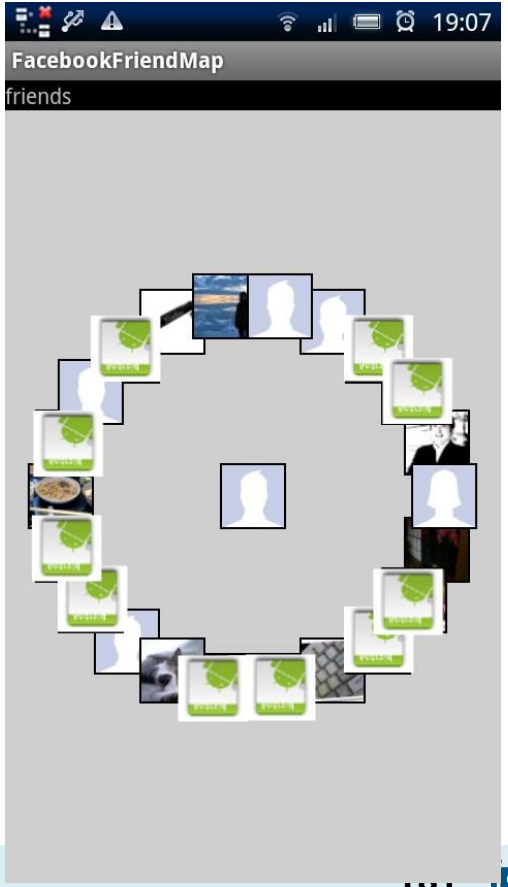


https://refactoring.guru/design-patterns/proxy

For example:

- 3rd party Facebook apps
  - **Android user interface**
  - Backend uses **Facebook data**

How do we test this?



Test Doubles

- Stand in for a real object under test
- Elements on which the unit testing depends (i.e. collaborators), but need to be approximated because they are
  - Unavailable
  - Expensive
  - Opaque
  - Non-deterministic
- Not just for distributed systems!

http://www.kickvick.com/celebrities-stunt-doubles

Design: Testability

- Single responsibility principle
- Dependency Inversion Principle (DIP)
  - High-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details. Details should depend upon abstractions.
- Law of Demeter: Don’t acquire dependencies through dependencies.
  - avoid: `this.getA().getB().doSomething()`
- Use factory pattern to instantiate new objects, rather than new.
- Use appropriate tools, e.g., dependency injection or mocking frameworks

Principles of Software Construction: Objects, Design, and Concurrency

DevOps

Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, Design Patterns, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, **DevOps**, Teams

Jenkins

search

Jenkins › Suisse › Stop-tabac dev

ENABLE AUTO REFRESH

- Back to Dashboard
- **Status**
- Changes
- Workspace
- Build Now
- Delete Project
- Configure
- Set Next Build Number
- Duplicate Code
- Coverage Report
- SLOCCount
- Git Polling Log

Build History (trend)

#977	Aug 27, 2012 4:37:27 PM
#438	Jun 28, 2012 8:47:42 AM
#426	Jun 26, 2012 1:39:39 PM
#345	Jun 19, 2012 9:02:20 AM
#263	Jun 6, 2012 9:14:42 PM
#210	May 31, 2012 8:42:29 AM
#171	May 23, 2012 9:58:18 PM
#90	May 15, 2012 11:49:41 AM

RSS for all RSS for failures

Project Stop-tabac dev

CI build

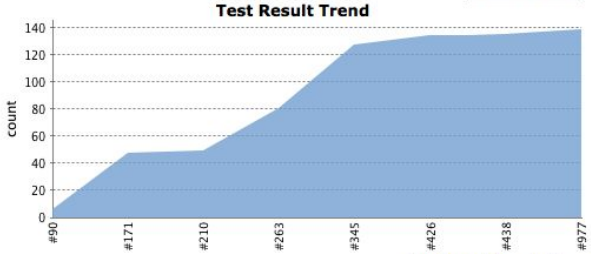
edit description

Disable Project

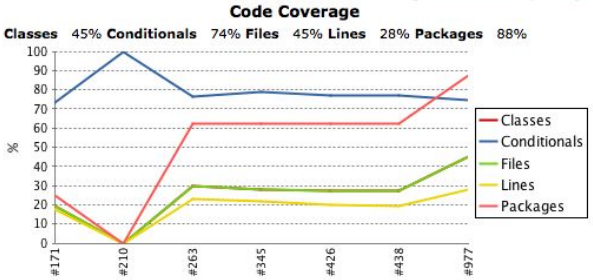
- Coverage Report
- Workspace
- Recent Changes
- Latest Test Result (no failures)

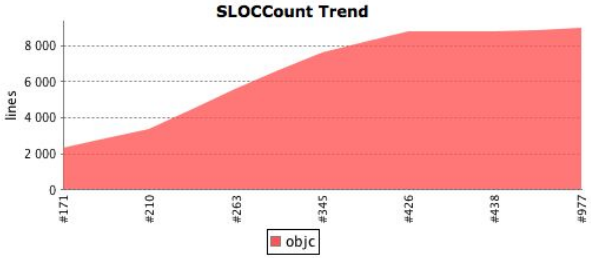
Permalinks

- Last build (#977), 3 min 17 sec ago
- Last stable build (#977), 3 min 17 sec ago
- Last successful build (#977), 3 min 17 sec ago



(just show failures) enlarge





Help us localize this page

Page generated: Aug 27, 2012 4:40:45 PM Jenkins ver. 1.470

institute for SOFTWARE RESEARCH

Aside: The role of signaling

Status

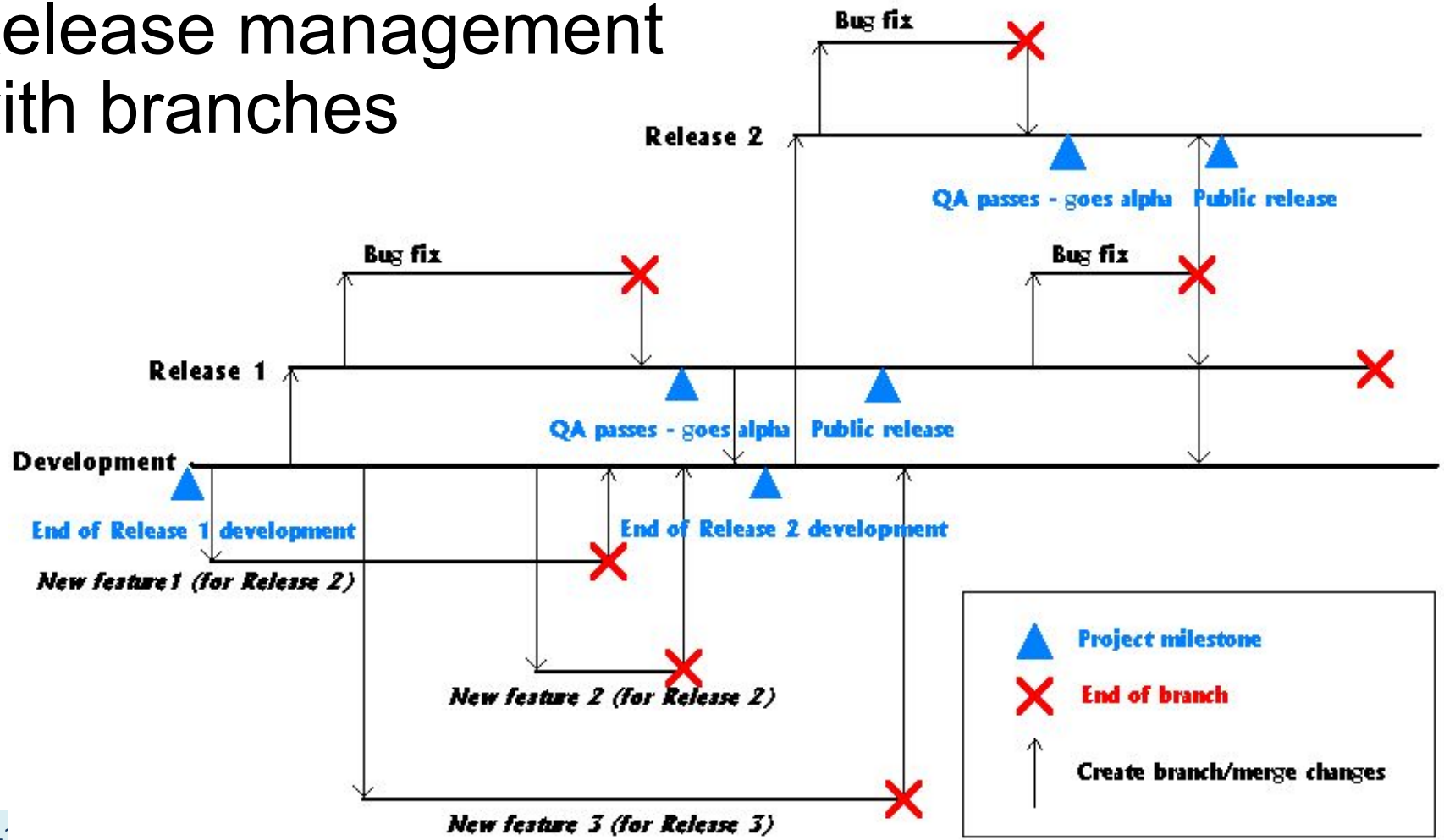
Build Pipeline

Release Pipeline

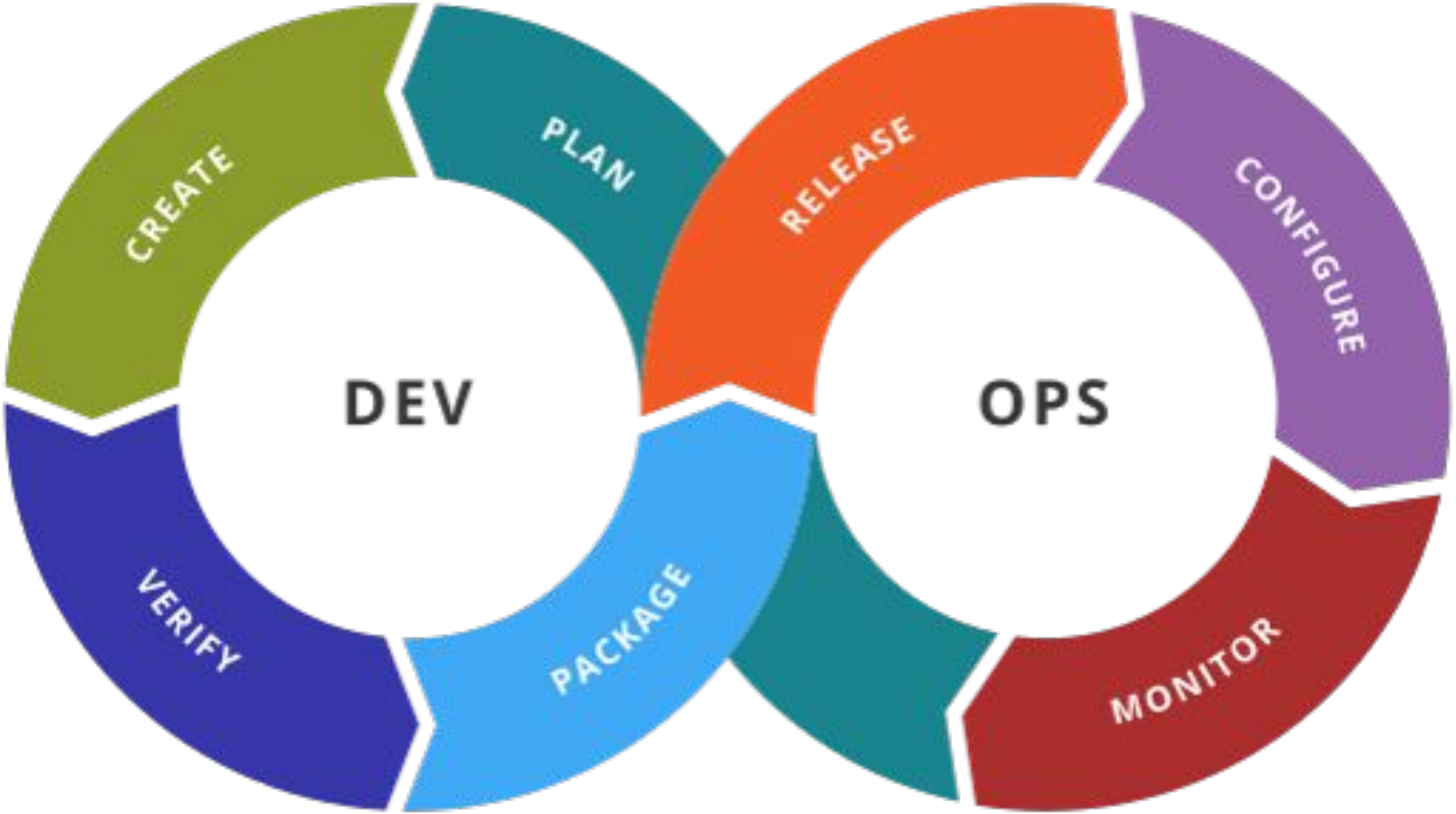
Dev	Test	Prod
		
		

https://blog.devops4me.com/status-badges-in-azure-devops-pipelines/

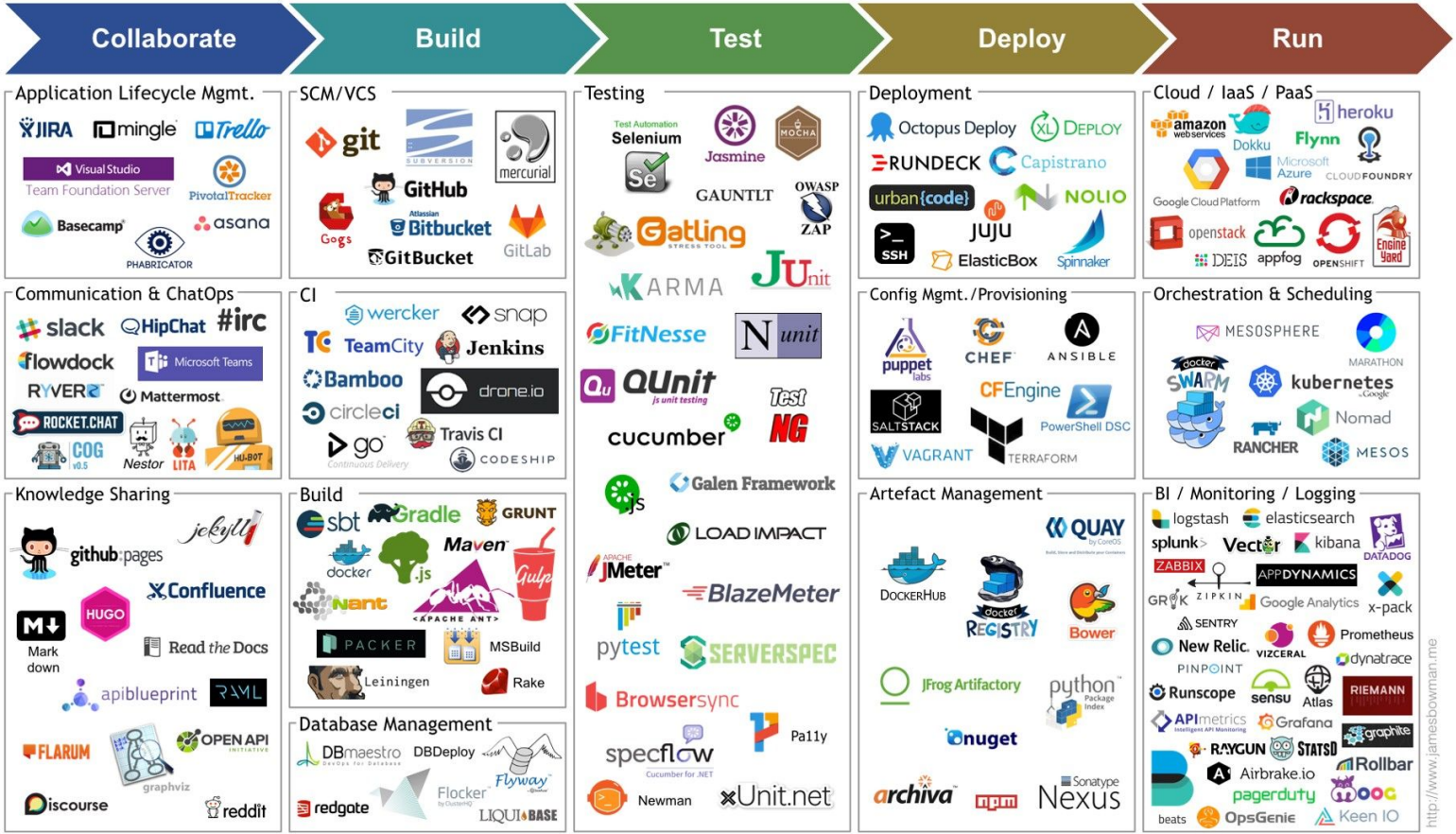
Release management with branches




WORKED FINE IN DEV
OPS PROBLEM NOW
memegenerator.net


CREATE
PLAN
RELEASE
CONFIGURE
DEV
OPS
VERIFY
PACKAGE
MONITOR

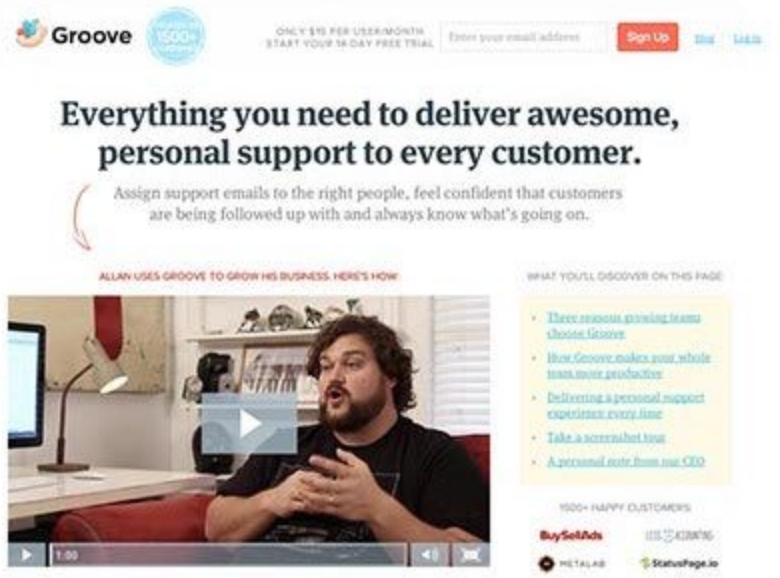
Heavy Tooling and Automation



A/B Testing

Original: 2.3%

Long Form: 4.3%



Looking Forward:
Beyond Code-Level Concerns

Where we are

Design for	*Small scale:* One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
understanding	Subtype Polymorphism ✓	Domain Analysis ✓	GUI vs Core ✓
change/ext.	Information Hiding, Contracts ✓	Inheritance & Del. ✓	Frameworks and Libraries ✓, APIs ✓
reuse	Immutability ✓	Responsibility Assignment, Design Patterns, Antipattern ✓	Module systems, microservices ✓
robustness	Types ✓ Static Analysis ✓	Promises/ Reactive P. ✓	Testing for Robustness ✓
...	Unit Testing ✓	Integration Testing ✓	CI ✓, DevOps ✓, **Teams**

This Course

We focused on code-level concerns

Writing maintainable, extensible, robust, and correct code

- Design from classes to subsystems
- Testing, concurrency, basic user interfaces

Toyota Case: Single Bit Flip That Killed

Junko Yoshida

10/25/2013 03:35 PM EDT

During the trial, embedded systems experts who reviewed Toyota's electronic throttle source code testified that they found Toyota's source code defective, and that it contains bugs -- including bugs that can cause unintended acceleration.

"We did a few things that NASA apparently did not have time to do," Barr said. For one thing, by looking within the real-time operating system, the experts identified "unprotected critical variables." They obtained and reviewed the source code for the "sub-CPU," and they "uncovered gaps and defects in the throttle fail safes."

The experts demonstrated that "the defects we found were linked to unintended acceleration through vehicle testing," Barr said. "We also obtained and reviewed the source code for the black box and found that it can record false information about the driver's actions in the final seconds before a crash."

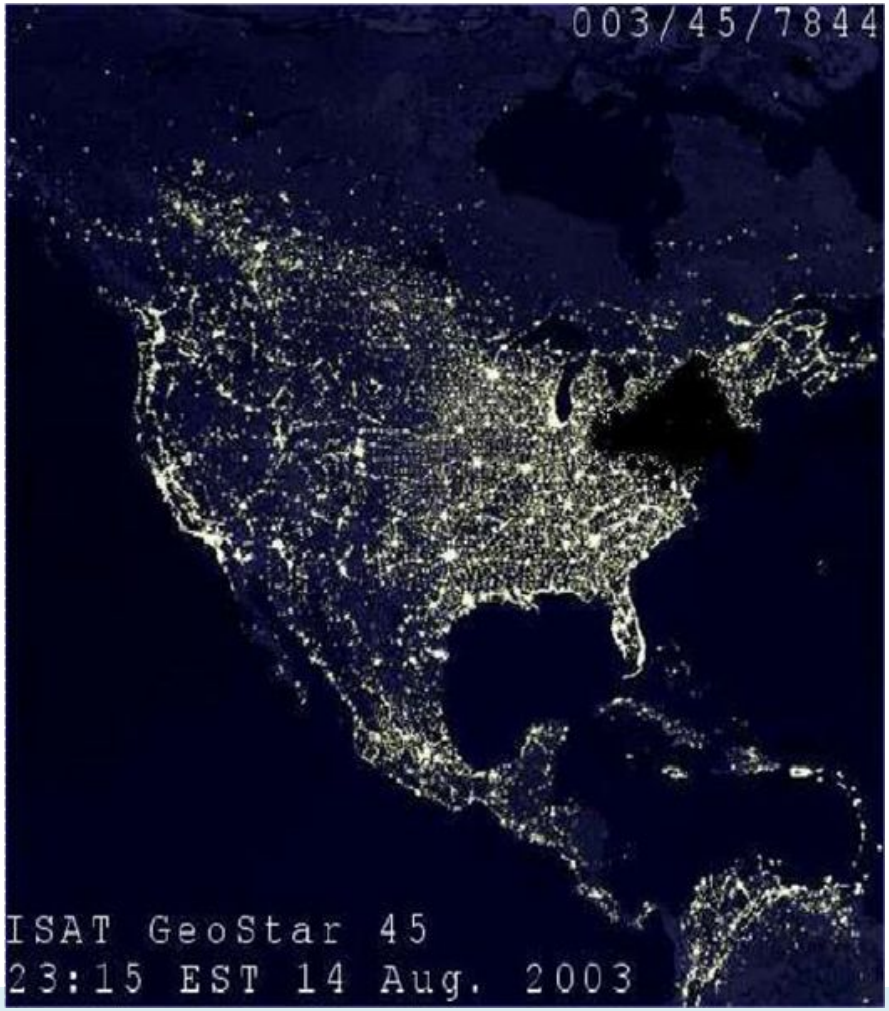
Stack overflow and software bugs led to memory corruption, he said. And it turns out that the crux of the issue was these memory corruptions, which acted "like ricocheting bullets."

Barr also said more than half the dozens of tasks' deaths studied by the experts in their experiments "were not detected by any fail safe."

Bookout Trial Reporting

http://www.eetimes.com/document.asp?doc_id=1319903&page_number=1 (excerpts)

"Task X death in combination with other task deaths"


003/45/7844
ISAT GeoStar 45
23:15 EST 14 Aug. 2003

Healthcare.gov: Government IT Project Failure at its Finest

Posted: 10/18/2013 6:33 pm

Read more > Project Management, Government, Healthcare, It Projects, Open Source, Business News

The *BusinessWeek* article on the Healthcare.gov failure is nothing if not instructive. From the piece:

> Healthcare.gov isn't just a website; it's more like a platform for building health-care marketplaces. Visiting the site is like visiting a restaurant. You sit in the dining room, read the menu, and tell the waiter what you want, and off he goes to the kitchen with your order. The dining room is the front end, with all the buttons to click and forms to fill out. The kitchen is the back end, with all the databases and services. The contractor most responsible for the back end is CGI Federal. Apparently it's this company's part of the system that's burning up under the load of thousands of simultaneous users.

The restaurant analogy is a good one. Projects with scopes like these fail for all sorts of reasons. *Why New Systems Fail* details a bunch of culprits, most of which are people-related.

As I read the article, a few other things jumped out at me, as they virtually guarantee failure:

- The sheer number of vendors involved
- The unwillingness of key parties involved with the back-end to embrace

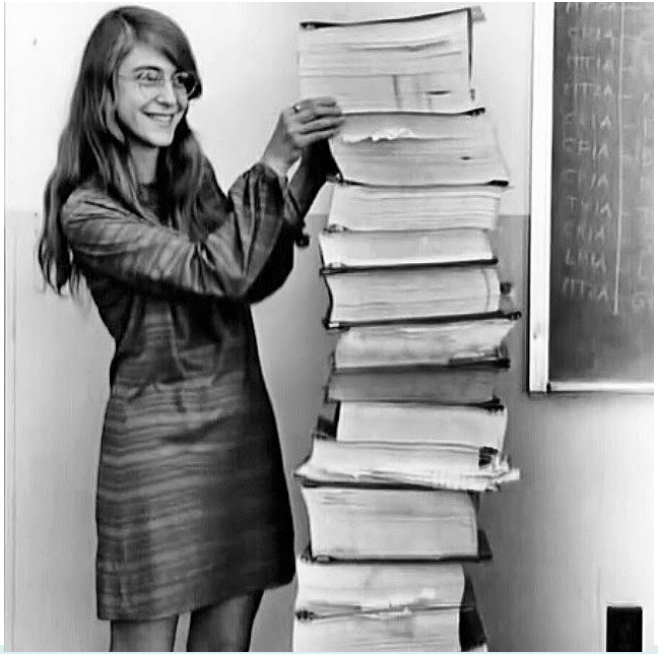
“But we’re CMU students and we are really, really smart!”

What is engineering? And how is it different from hacking/programming?

Software *Engineering*?

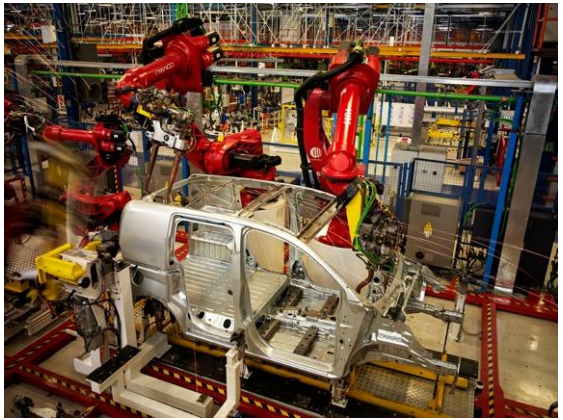
1968 NATO Conference on Software Engineering

"Software Engineering" was a provocative term



Compare to other forms of engineering

- e.g., Producing a car or bridge
  - Estimable costs and risks
  - Well-defined expected results
  - High quality
- Separation between plan and production
- Simulation before construction
- Quality assurance through measurement
- Potential for automation



From Programming to Software Engineering

Healthcare.gov: Government IT Project Failure at its Finest

Posted: 10/18/2013 6:33 pm

Read more > Project Management, Government, Healthcare, It Projects, Open Source, Business News

The *BusinessWeek* article on the Healthcare.gov failure is nothing if not instructive. From the piece:

> Healthcare.gov isn't just a website; it's more like a platform for building health-care marketplaces. Visiting the site is like visiting a restaurant. You sit in the dining room, read the menu, and tell the waiter what you want, and off he goes to the kitchen with your order. The dining room is the front end, with all the buttons to click and forms to fill out. The kitchen is the back end, with all the databases and services. The contractor most responsible for the back end is CGI Federal. Apparently it's this company's part of the system that's burning up under the load of thousands of simultaneous users.

The restaurant analogy is a good one. Projects with scopes like these fail for all sorts of reasons. *Why New Systems Fail* details a bunch of culprits, most of which are people-related.

As I read the article, a few other things jumped out at me, as they virtually guarantee failure:

- The sheer number of vendors involved
- The unwillingness of key parties involved with the back-end to embrace

What happened with HealthCare.gov?

- Poor team and process coordination.
- Changing requirements.
- Inadequate quality assurance infrastructure.
- Architecture unsuited to the ultimate system load.

But….*why??*

Boeing 737 MAX

Software is written by humans

Sociotechnical system: interlinked system of people, technology, and their environment

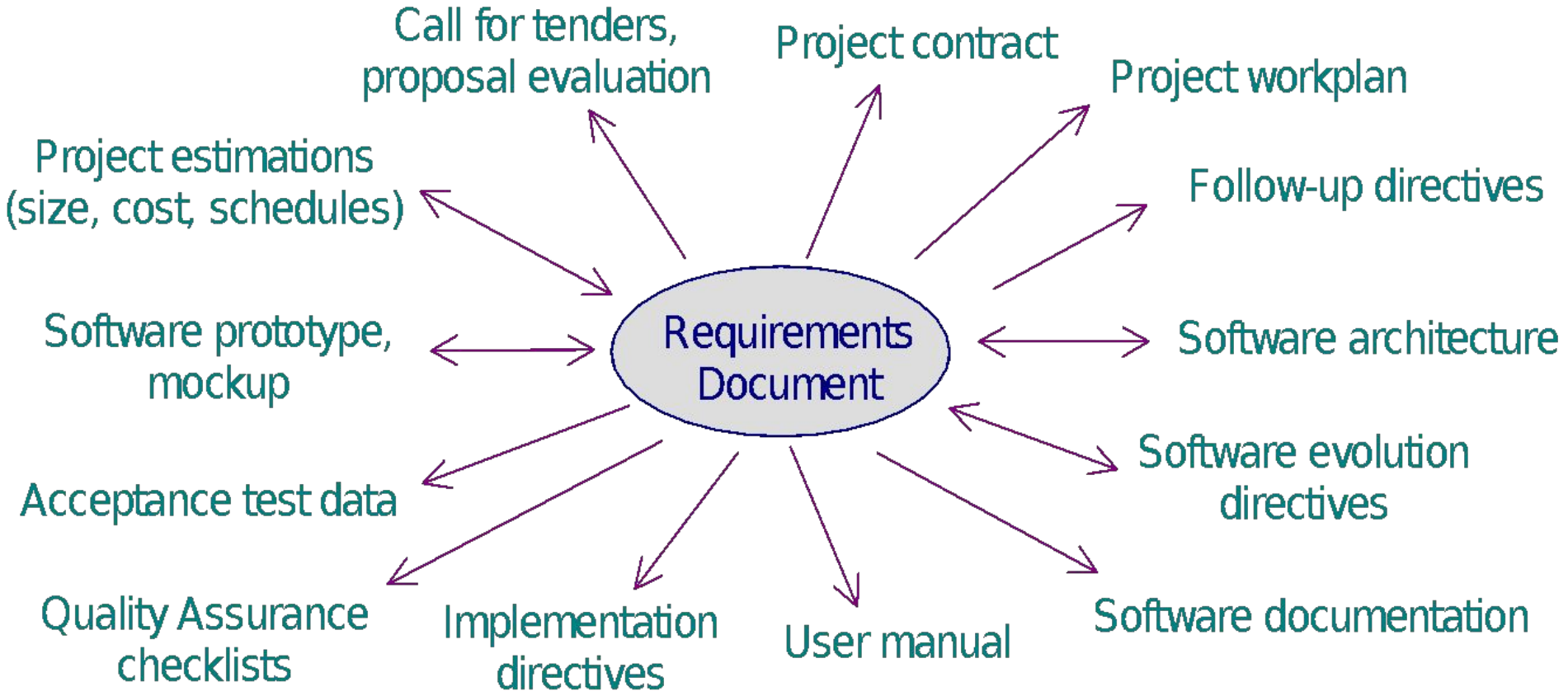
Key challenges in how to

- identify what to build (requirements)
- coordinate people building it (process)
- assure quality (speed, safety, fairness)
- contain risk, time and budget (management)
- sustain a community (open source, economics)

Requirements

Requirements

- What does the customer want?
- What is required, desired, not necessary? Legal, policy constraints?
- Customers often do not know what they really want; vague, biased by what they see; change their mind; get new ideas…
- Difficult to define requirements precisely
- (Are we building the right thing? Not: Are we building the thing right?)


Call for tenders, proposal evaluation
Project contract
Project workplan
Project estimations (size, cost, schedules)
Follow-up directives
Software prototype, mockup
Requirements Document
Software architecture
Software evolution directives
Acceptance test data
Quality Assurance checklists
Implementation directives
User manual
Software documentation

Interviews

Abby Jones[1]

You can edit anything in blue print

- 28 years old
- Employed as an Accountant
- Lives in Cardiff, Wales

Abby has always liked music. When she is on her way to work in the morning, she listens to music that spans a wide variety of styles. But when she arrives at work, she turns it off, and begins her day by scanning all her emails first to get an overall picture before answering any of them. (This extra pass takes time but seems worth it.) Some nights she exercises or stretches, and sometimes she likes to play computer puzzle games like Sudoku

Background and skills

Abby works as an accountant. She is comfortable with the technologies she uses regularly, but she just moved to this employer 1 week ago, and their software systems are new to her.

Abby says she's a "numbers person", but she has never taken any computer programming or IT systems classes. She likes Math and knows how to think with numbers She writes and edits spreadsheet formulas in her work.

In her free time, she also enjoys working with numbers and logic. She especially likes working out puzzles and puzzle games, either on paper or on the computer

Motivations and Attitudes

- ***Motivations***: Abby uses technologies to accomplish her tasks. She learns new technologies if and when she needs to, but prefers to use methods she is already familiar and comfortable with, to keep her focus on the tasks she cares about.
- ***Computer Self-Efficacy***: Abby has low confidence about doing unfamiliar computing tasks. If problems arise with her technology, she often blames herself for these problems. This affects whether and how she will persevere with a task if technology problems have arisen.
- ***Attitude toward Risk:*** Abby's life is a little complicated and she rarely has spare time. So she is risk averse about using unfamiliar technologies that might need her to spend extra time on them, even if the new features might be relevant. She instead performs tasks using familiar features, because they're more predictable about what she will get from them and how much time they will take.

How Abby Works with Information and Learns:

- ***Information Processing Style:*** Abby tends towards a *comprehensive*
- ***Learning: by Process vs. by Tinkering***: When learning new technology,

Process

How to develop software?

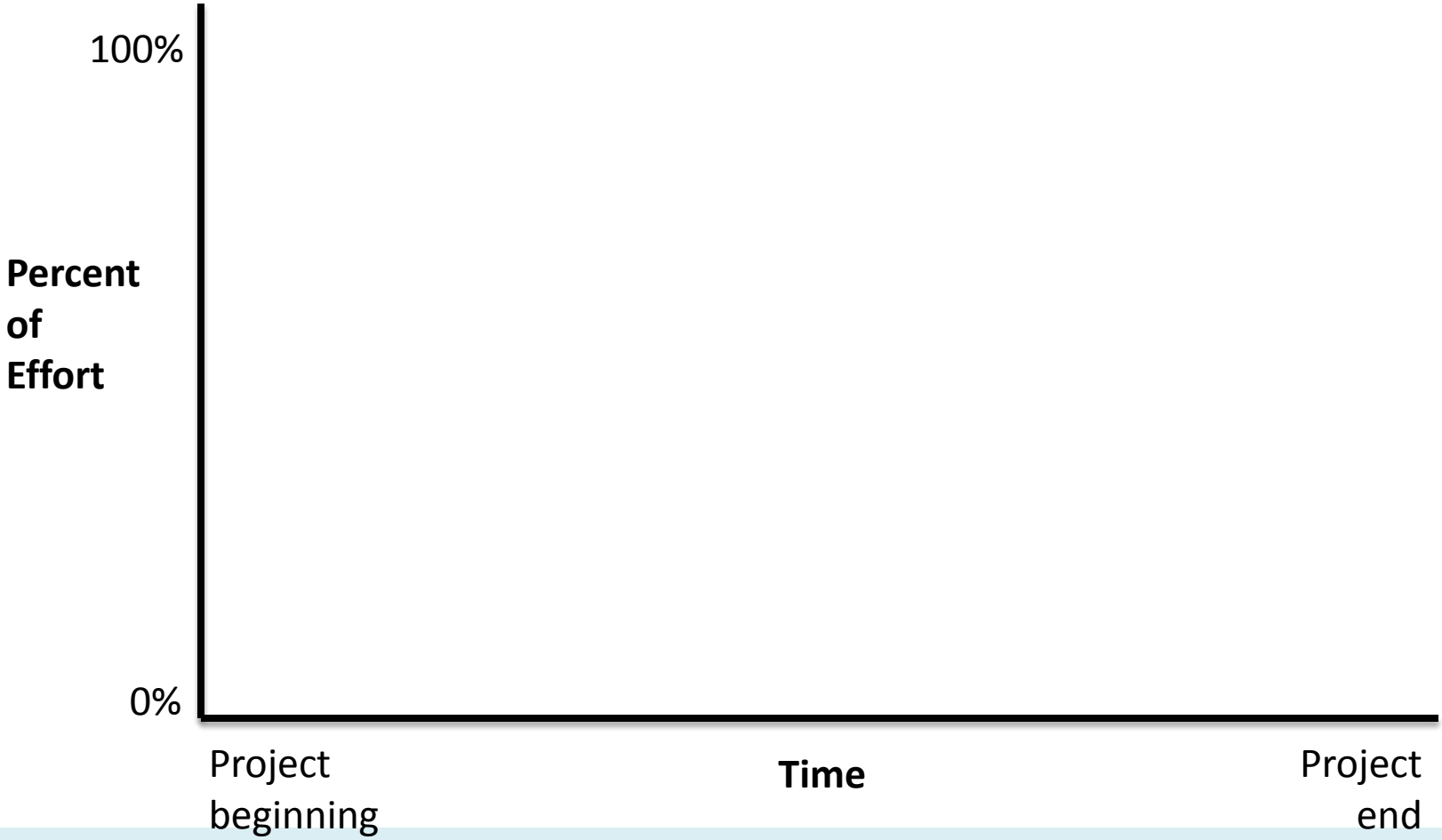
1. Discuss the software that needs to be written
2. Write some code
3. Test the code to identify the defects
4. Debug to find causes of defects
5. Fix the defects
6. If not done, return to step 1

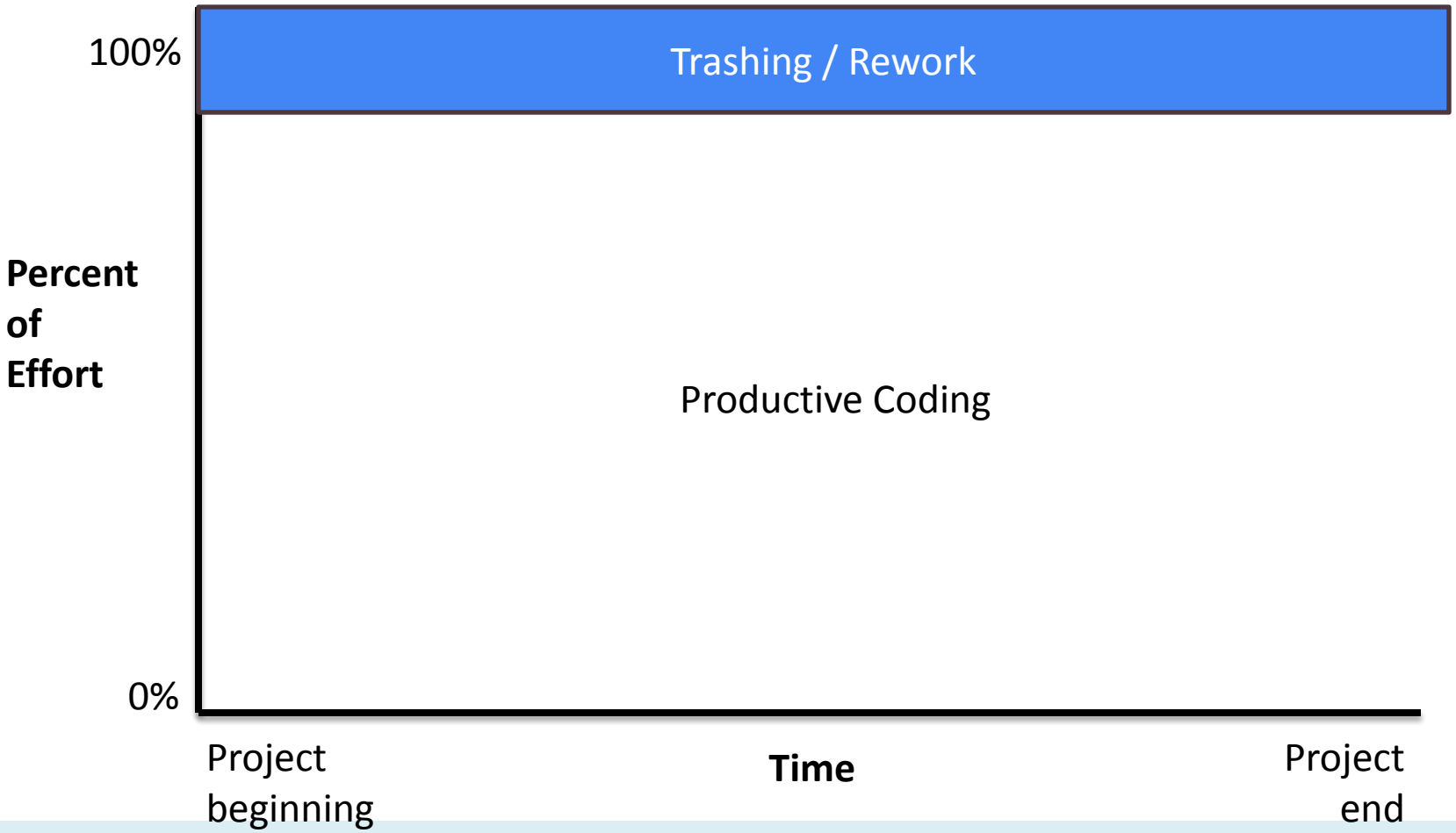
Software Process

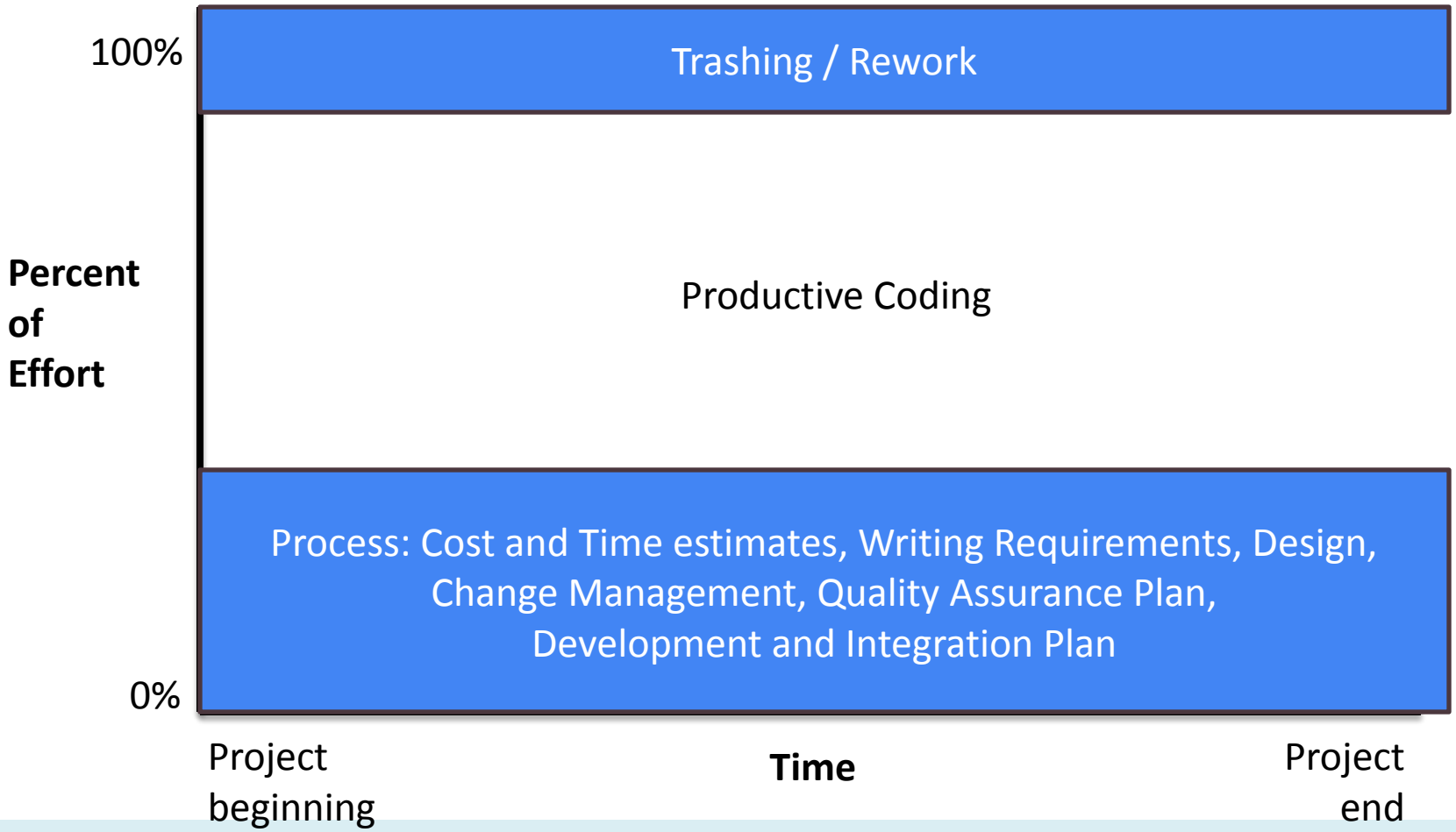
“The set of activities and associated results that produce a software product”

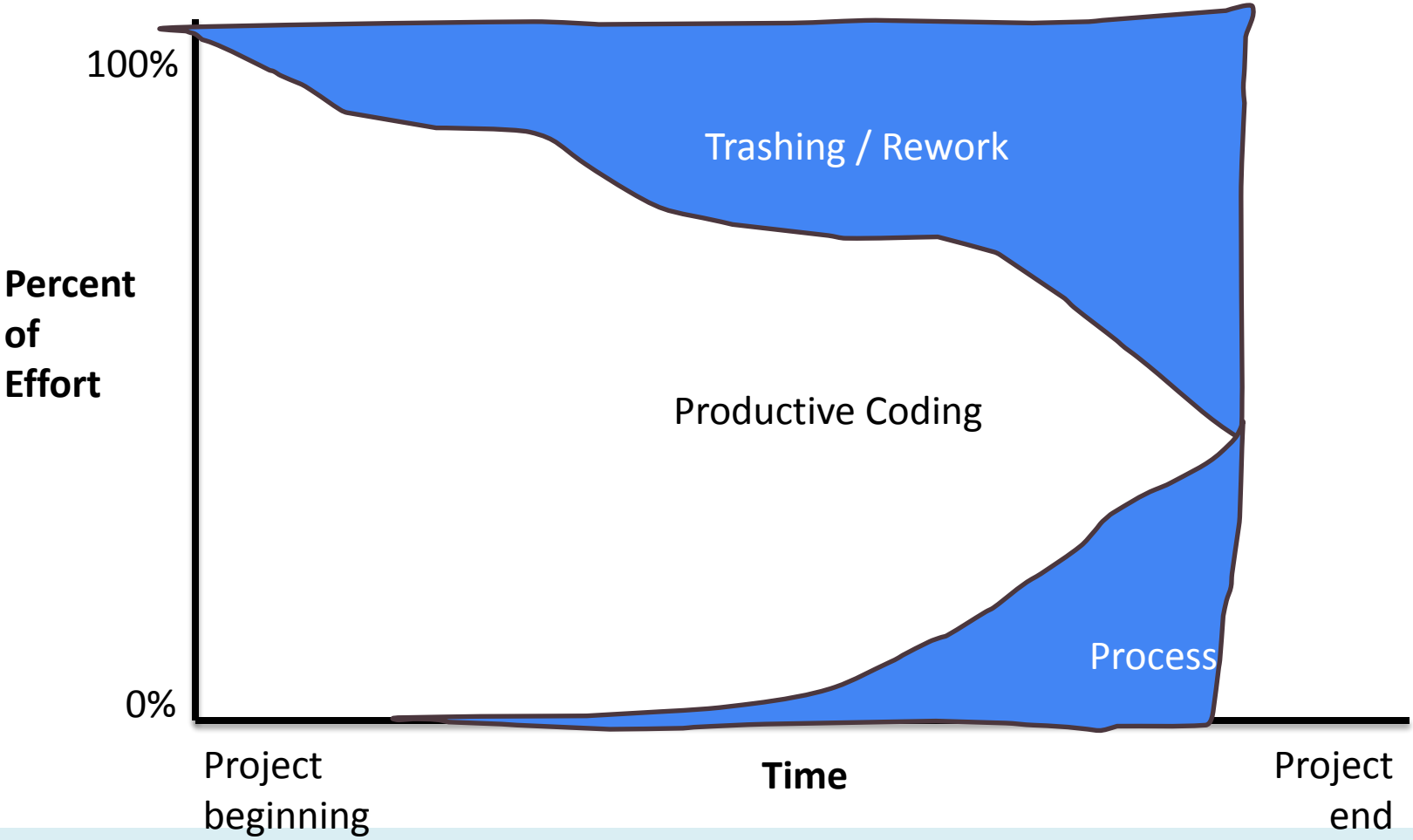
What makes a good process?

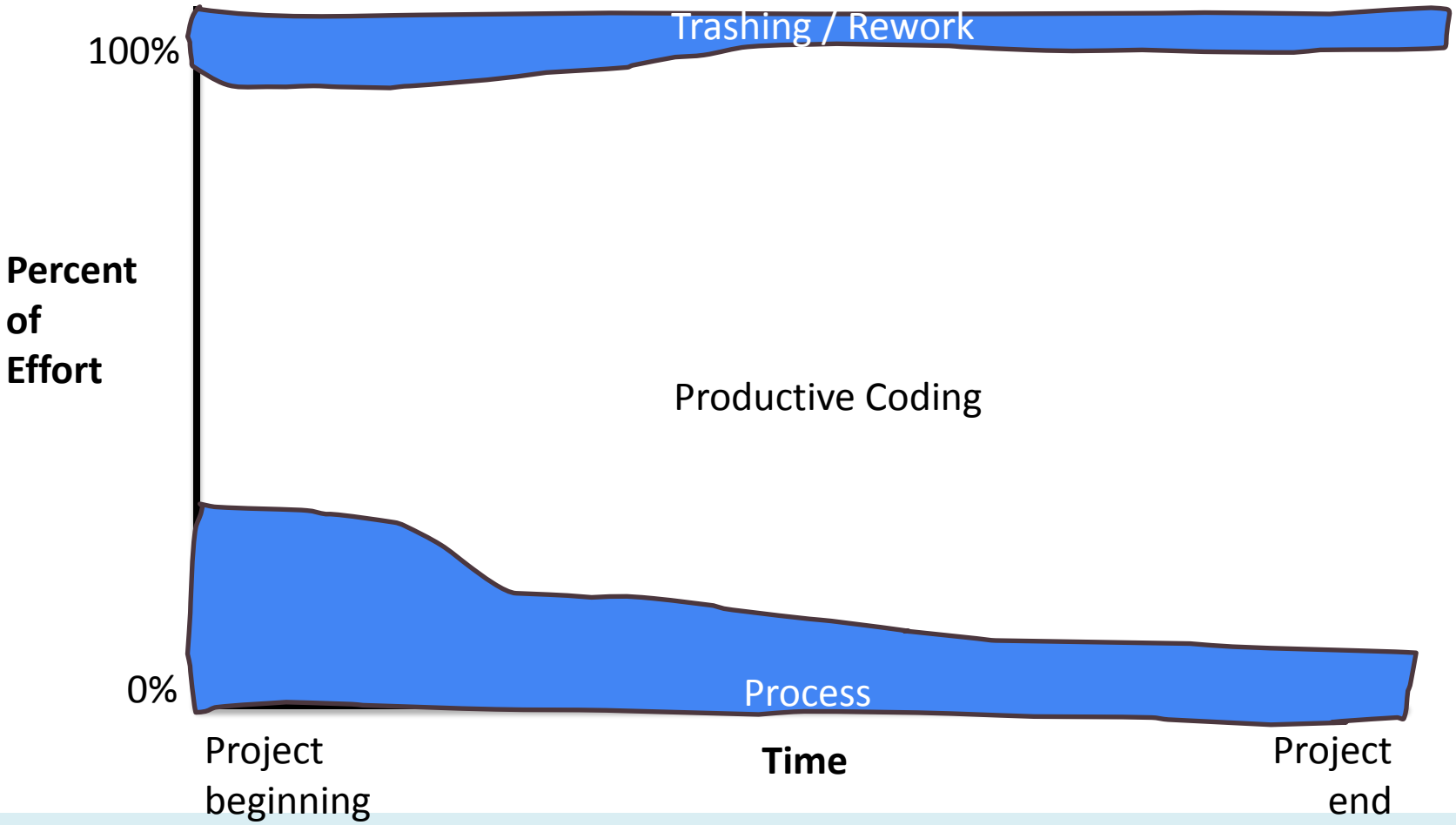
Sommerville, SE, ed. 8


100%
Percent of Effort
0%
Project beginning
Time
Project end


100%
Trashing / Rework
Percent of Effort
Productive Coding
0%
Project beginning
Time
Project end


100%
Trashing / Rework
Percent of Effort
Productive Coding
Process: Cost and Time estimates, Writing Requirements, Design, Change Management, Quality Assurance Plan, Development and Integration Plan
0%
Project beginning
Time
Project end

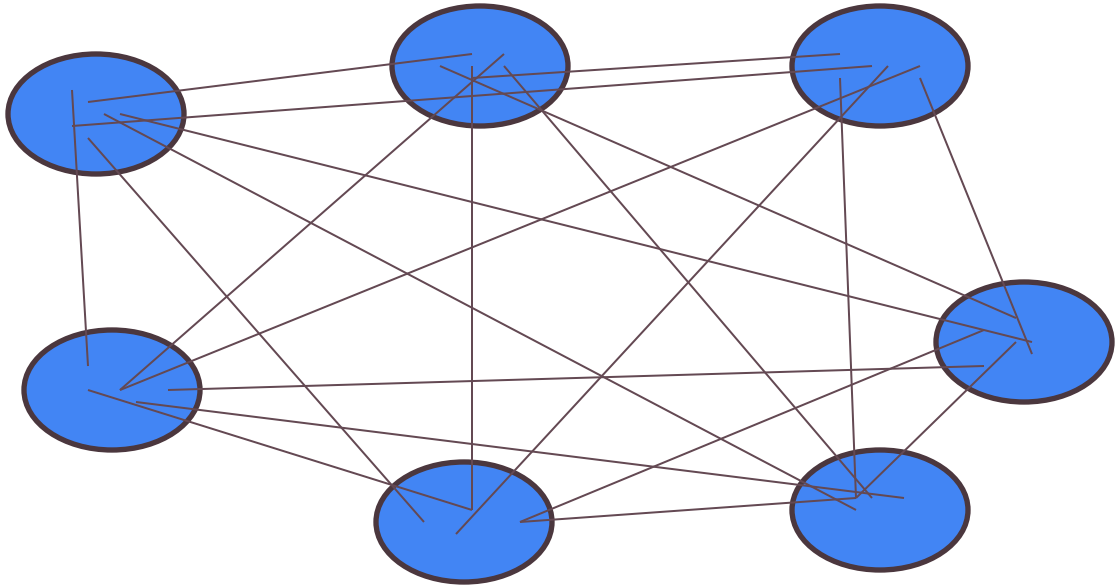

100%
Trashing / Rework
Percent of Effort
Productive Coding
Process
0%
Project beginning
Time
Project end


Trashing / Rework
100%
Percent of Effort
Productive Coding
0%
Process
Project beginning
Time
Project end

Example process issues

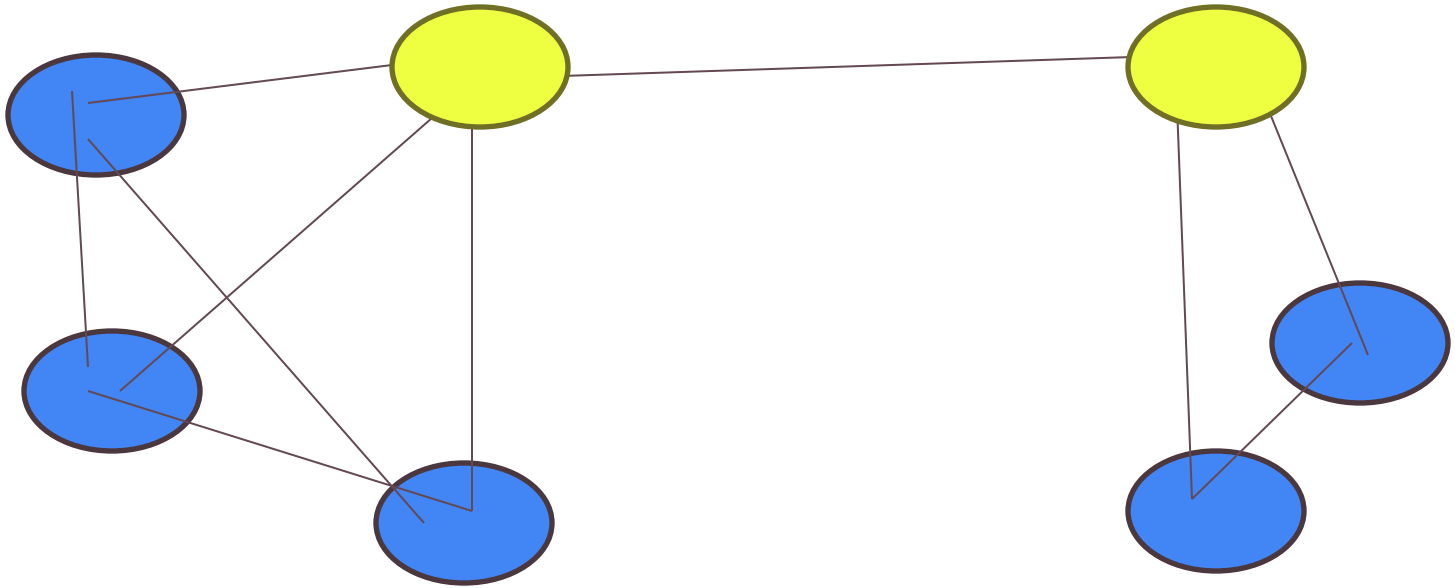
- Change Control: Mid-project informal agreement to changes suggested by customer or manager. Project scope expands 25-50%
- Quality Assurance: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features. Release with known defects.
- Defect Tracking: Bug reports collected informally, forgotten
- System Integration: Integration of independently developed components at the very end of the project. Interfaces out of sync.
- Source Code Control: Accidentally overwritten changes, lost work.
- Scheduling: When project is behind, developers are asked weekly for new estimates.

Process Costs



$n(n-1)/2$
communication links

Process Costs



Large teams (29 people) create around six times as many defects as small teams (3 people) and obviously burn through a lot more money. Yet, the large team appears to produce about the same mount of output in only an average of 12 days' less time. This is a truly astonishing finding, through it fits with my personal experience on projects over 35 years.

- Phillip Amour, 2006, CACM 49:9

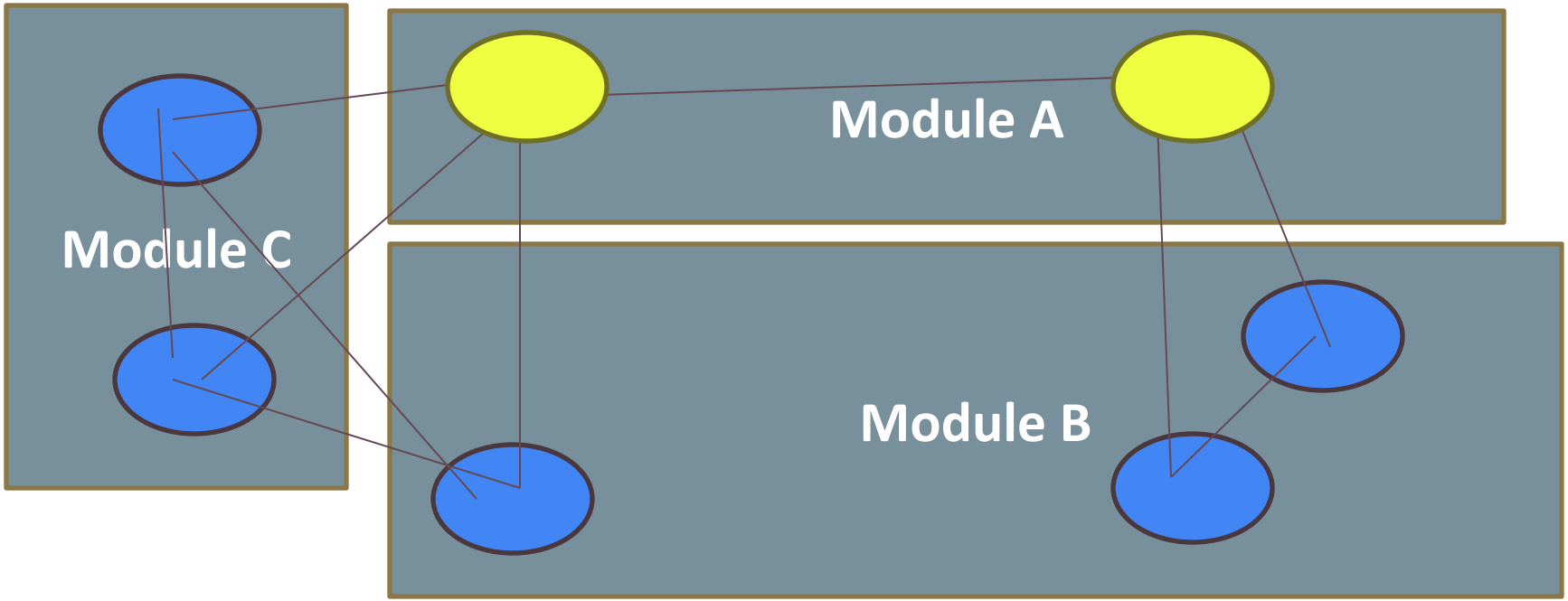
Conway's Law

"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure."

— *Mel Conway, 1967*

"If you have four groups working on a compiler, you'll get a 4-pass compiler."

Congruence



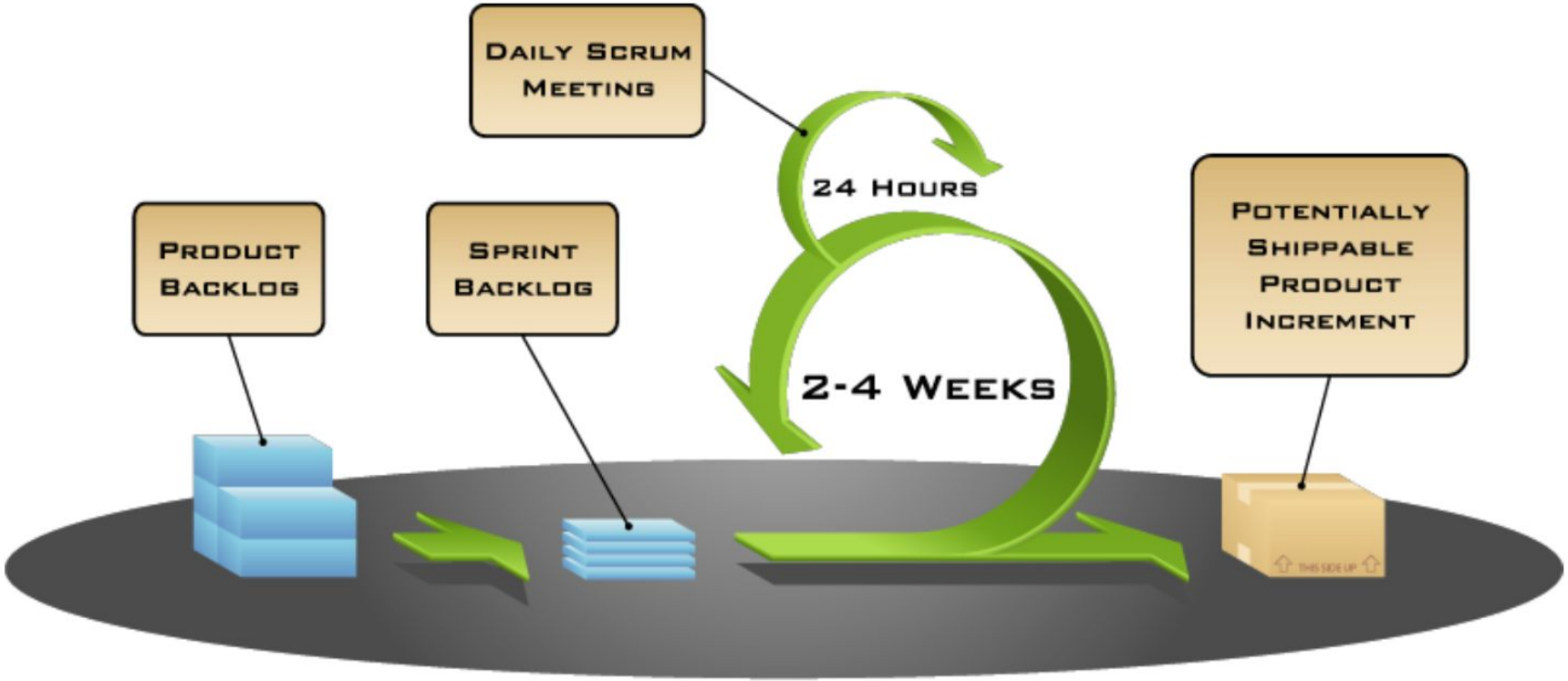
The Manifesto for Agile Software Development (2001)

Value

Individuals and interactions	*over*	Processes and tools
Working software	*over*	Comprehensive documentation
Customer collaboration	*over*	Contract negotiation
Responding to change	*over*	Following a plan

Pair Programming

Scrum Process



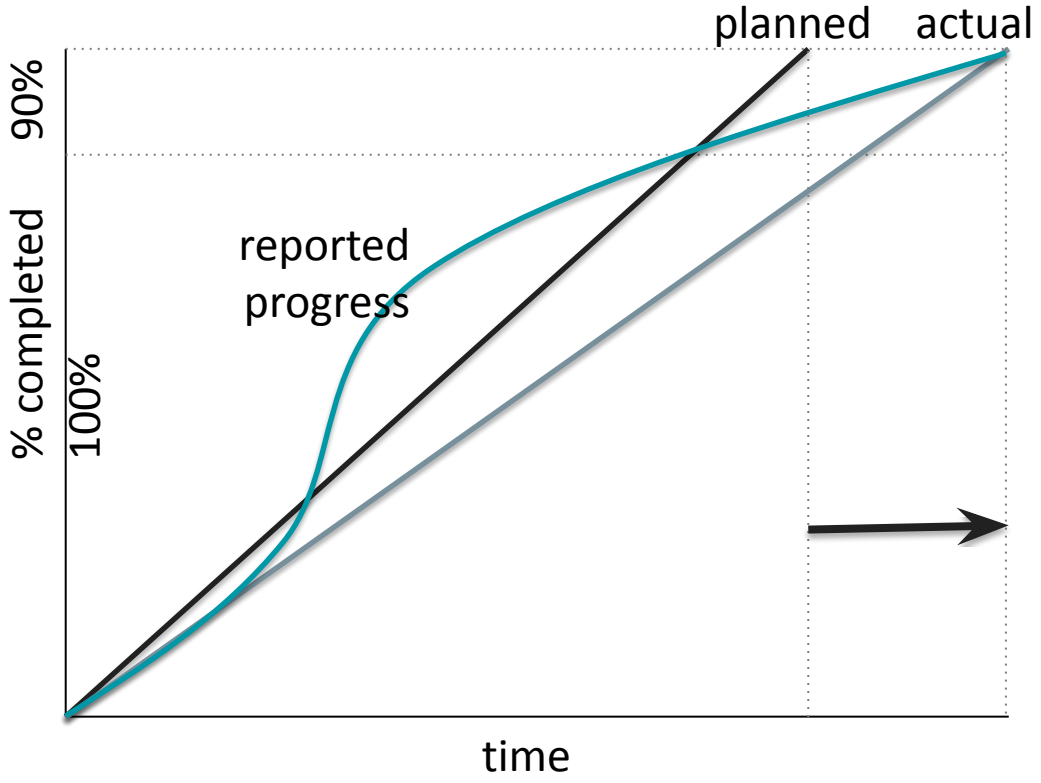
Planning

Measuring Progress?

"I'm almost done with the X. Component A is almost fully implemented. Component B is finished except for the one stupid bug that sometimes crashes the server. I only need to find the one stupid bug, but that can probably be done in an afternoon?"

Almost Done Problem

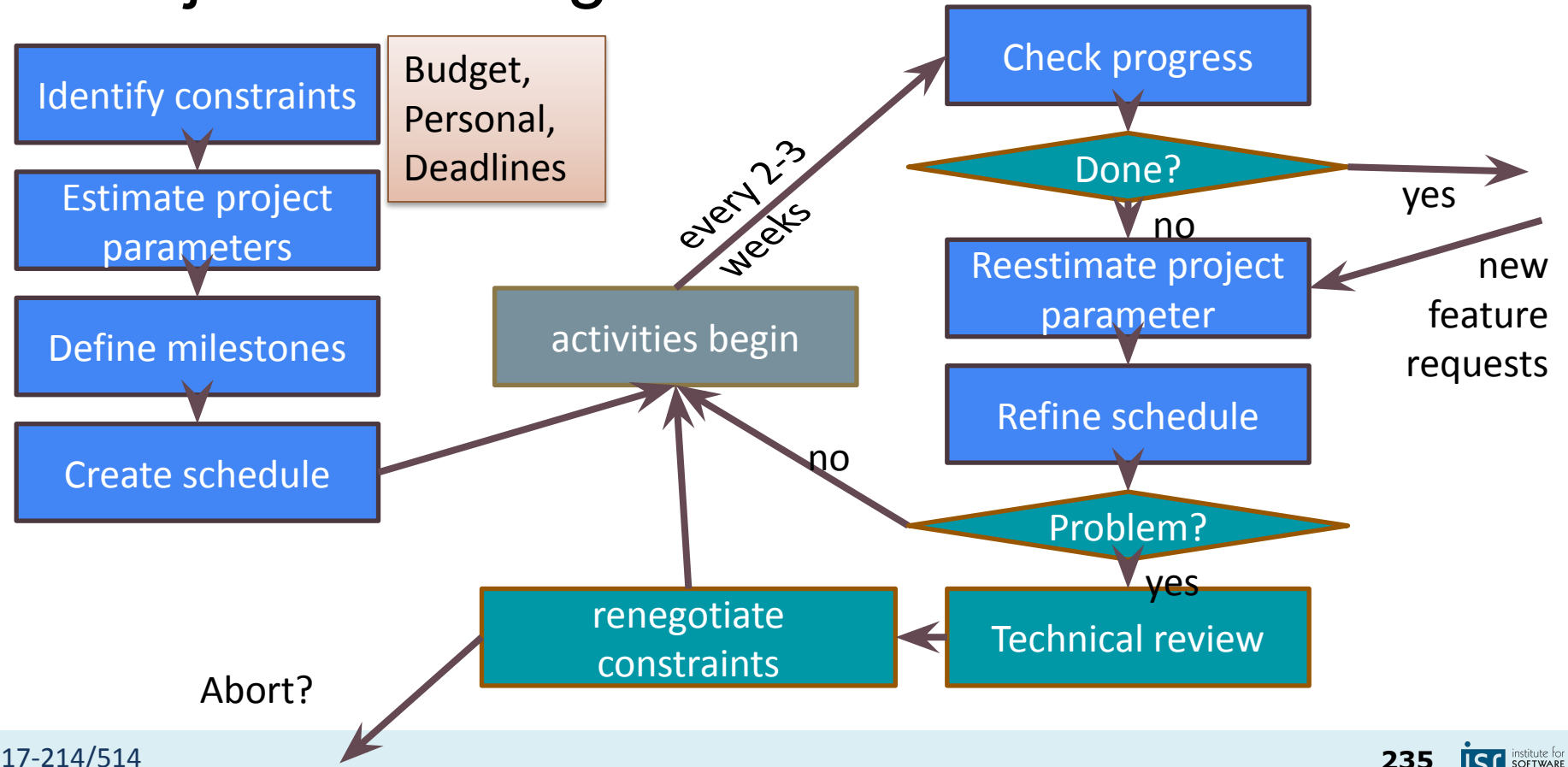
- Last 10% of work -> 40% of time
  (or 20/80)
- Make progress measureable
- Avoid depending entirely on developer estimations



Measuring Progress?

- Developer judgment: x% done
- Lines of code?
- Functionality?
- Quality?

Project Planning

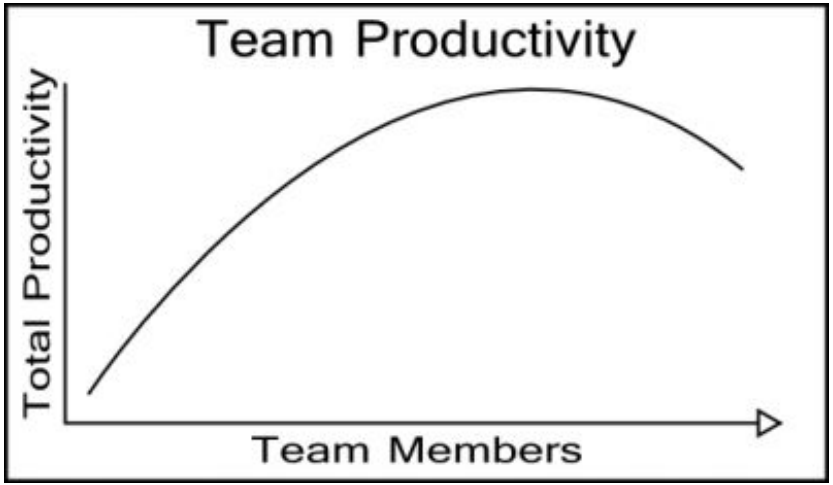


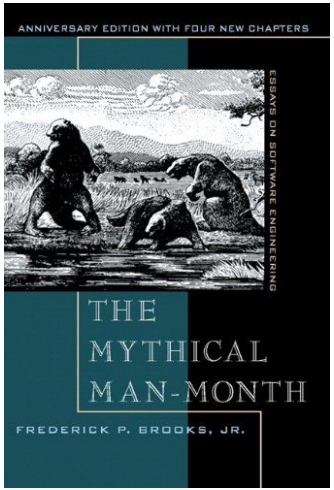
Reasons for Missed Deadlines

- Insufficient staff (illnesses, staff turnover, ...)
- Insufficient qualitication
- Unanticipated difficulties
- Unrealistic time estimations
- Unanticipated dependencies
- Changing requirements, additional requirements
- Especially in student projects
  - Underestimated time for learning technologies
  - Uneven work distribution
  - Last-minute panic.

Team productivity

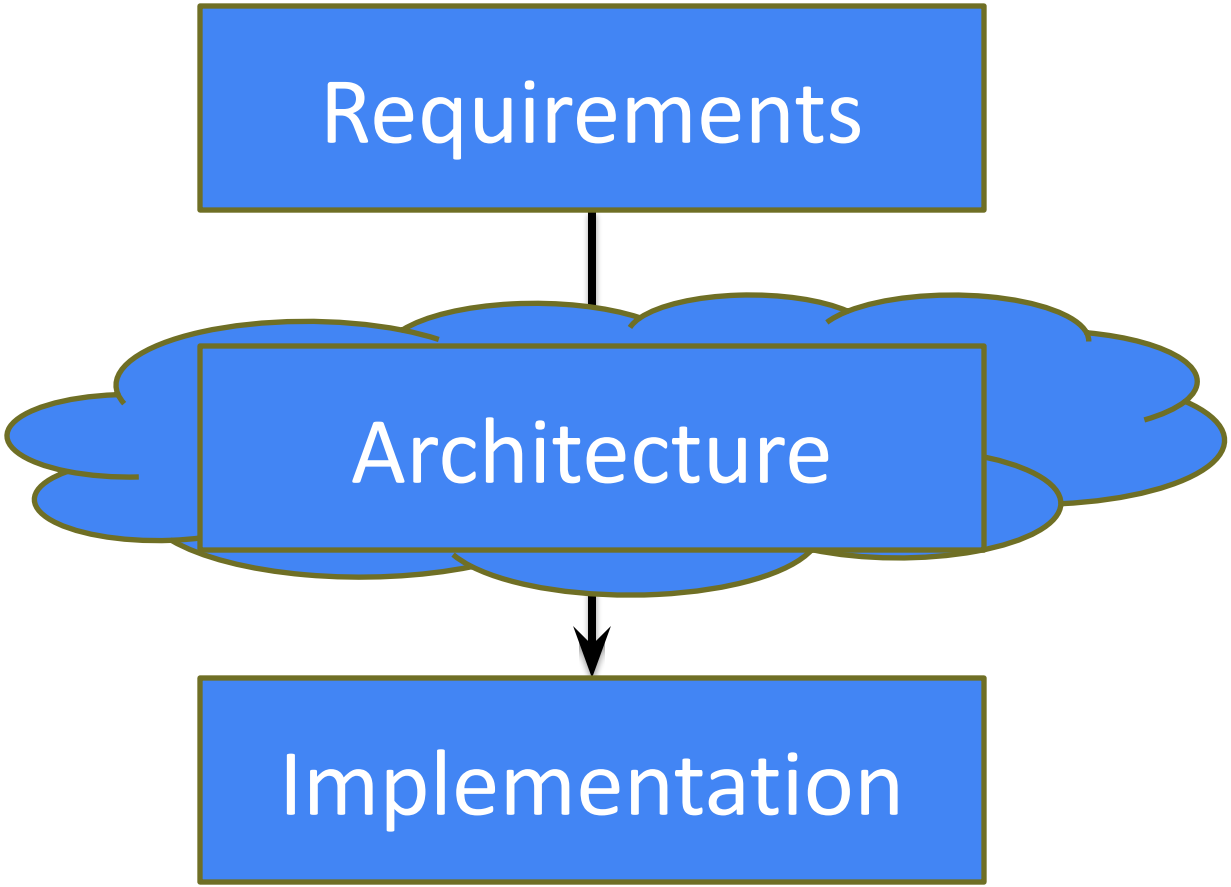
- Brook's law: Adding people to a late software project makes it later.





Estimating effort

Software Architecture


Requirements
Architecture
Implementation

Software Architecture

"The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both."

[Clements et al. 2010]

Design vs. Architecture

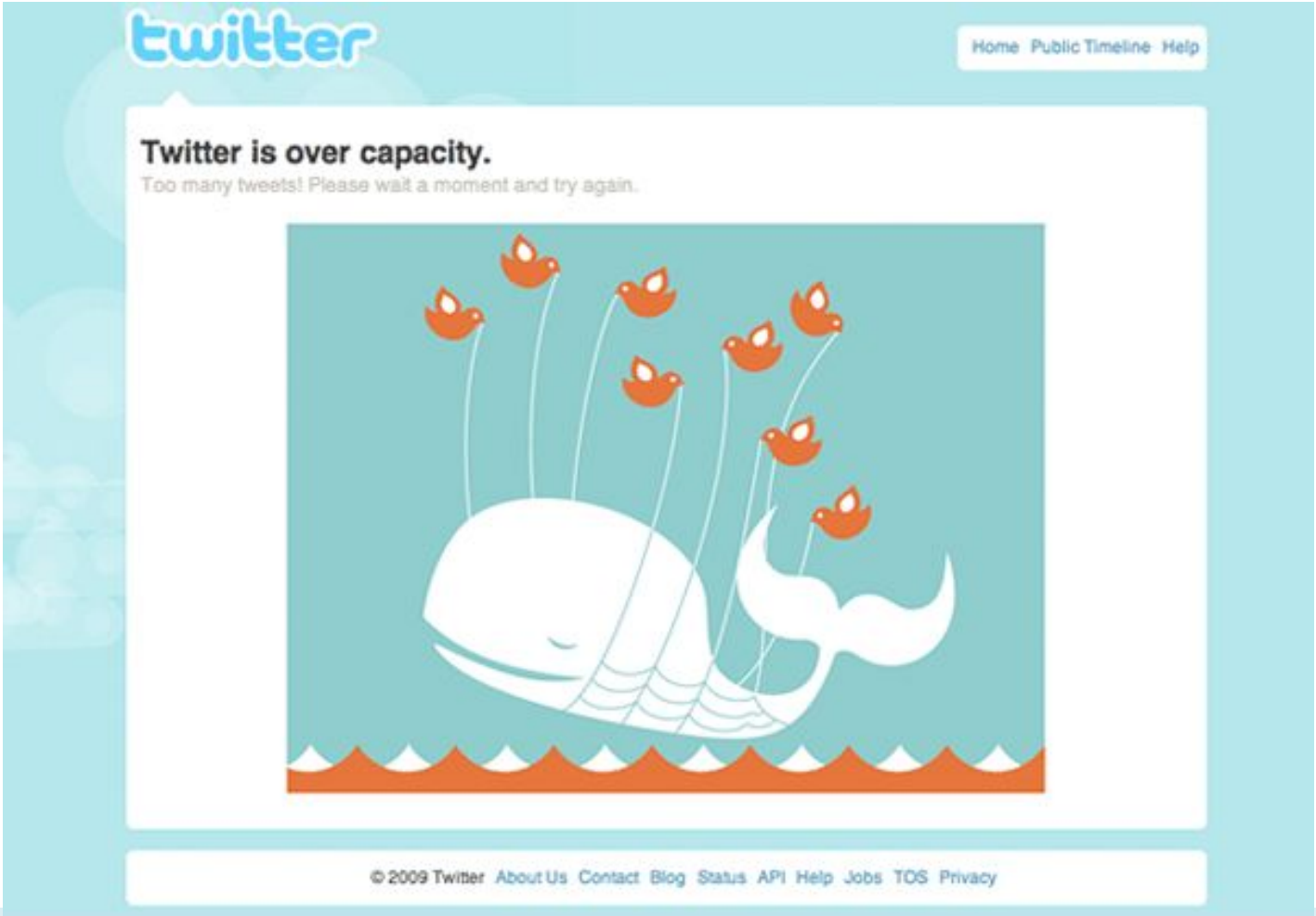
Design Questions

- How do I add a menu item in Eclipse?
- How can I make it easy to add menu items in Eclipse?
- What lock protects this data?
- How does Google rank pages?
- What encoder should I use for secure communication?
- What is the interface between objects?

Architectural Questions

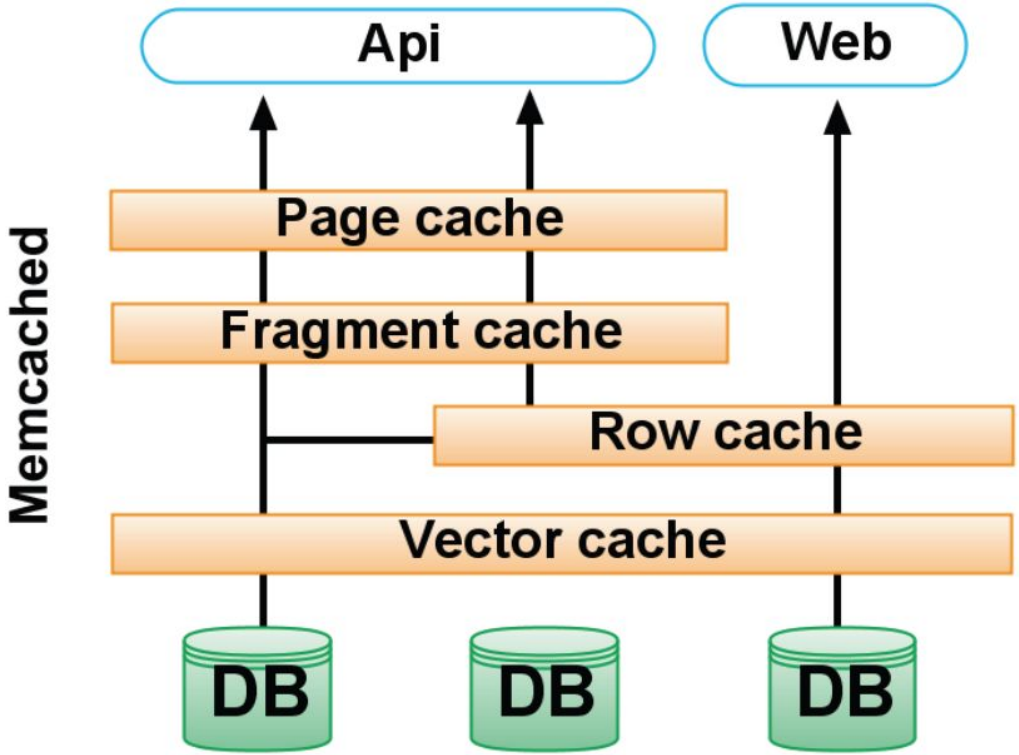
- How do I extend Eclipse with a plugin?
- What threads exist and how do they coordinate?
- How does Google scale to billions of hits per day?
- Where should I put my firewalls?
- What is the interface between subsystems?

Case Study: Architecture Changes at Twitter


twitter
Home Public Timeline Help
Twitter is over capacity.
Too many tweets! Please wait a moment and try again.
© 2009 Twitter About Us Contact Blog Status API Help Jobs TOS Privacy


SOUTH
AFRICA
2010
FIFA
WORLD CUP

Caching


Api
Web
Memcached
Page cache
Fragment cache
Row cache
Vector cache
DB
DB
DB

Redesign Goals

- Improve median latency; lower outliers
- Reduce number of machines 10x
- Isolate failures
- "We wanted cleaner boundaries with “related” logic being in one place"
  - encapsulation and modularity at the systems level (rather than at the class, module, or package level)
- Quicker release of new features
  - "run small and empowered engineering teams that could make local decisions and ship user-facing changes, independent of other teams"

performance

reliability

maintainability

modifiability

Outcome: Rearchitecting Twitter

"This re-architecture has not only made the service more **resilient when traffic spikes** to record highs, but also provides a more **flexible** platform on which to **build more features faster**, including synchronizing direct messages across devices, Twitter cards that allow Tweets to become richer and contain more content, and a rich search experience that includes stories and users."

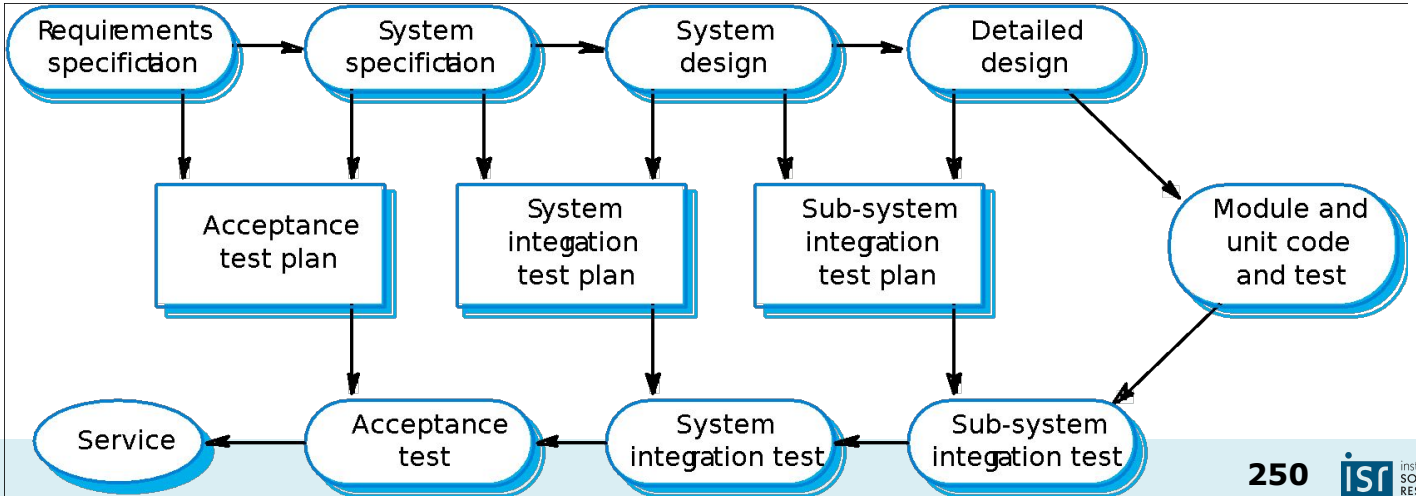
Was the original architect wrong?

Beyond testing: QA and Process

Many QA approaches

Code review, static analysis, formal verification, …

Which to use when, how much?



How to get students to write tests?

“We had initially scheduled time to write tests for both front and back end systems, although this never happened.”

“Due to the lack of time, we could only conduct individual pages’ unit testing. Limited testing was done using use cases. Our team felt that this testing process was rushed and more time and effort should be allocated.”

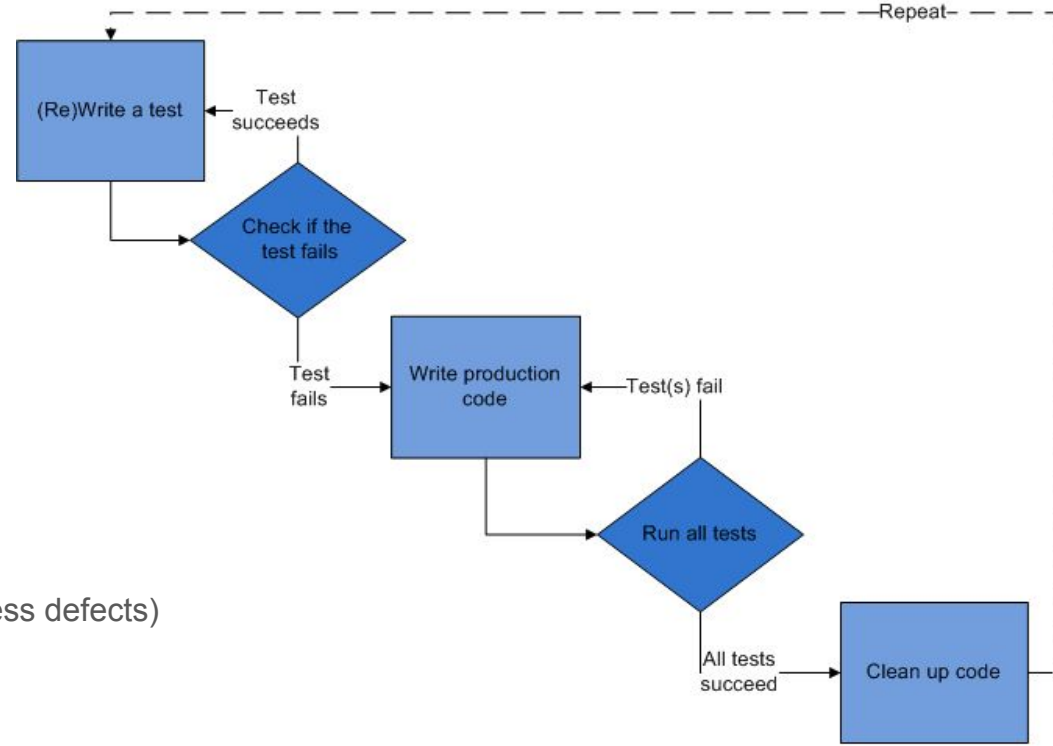
Time estimates (in hours):

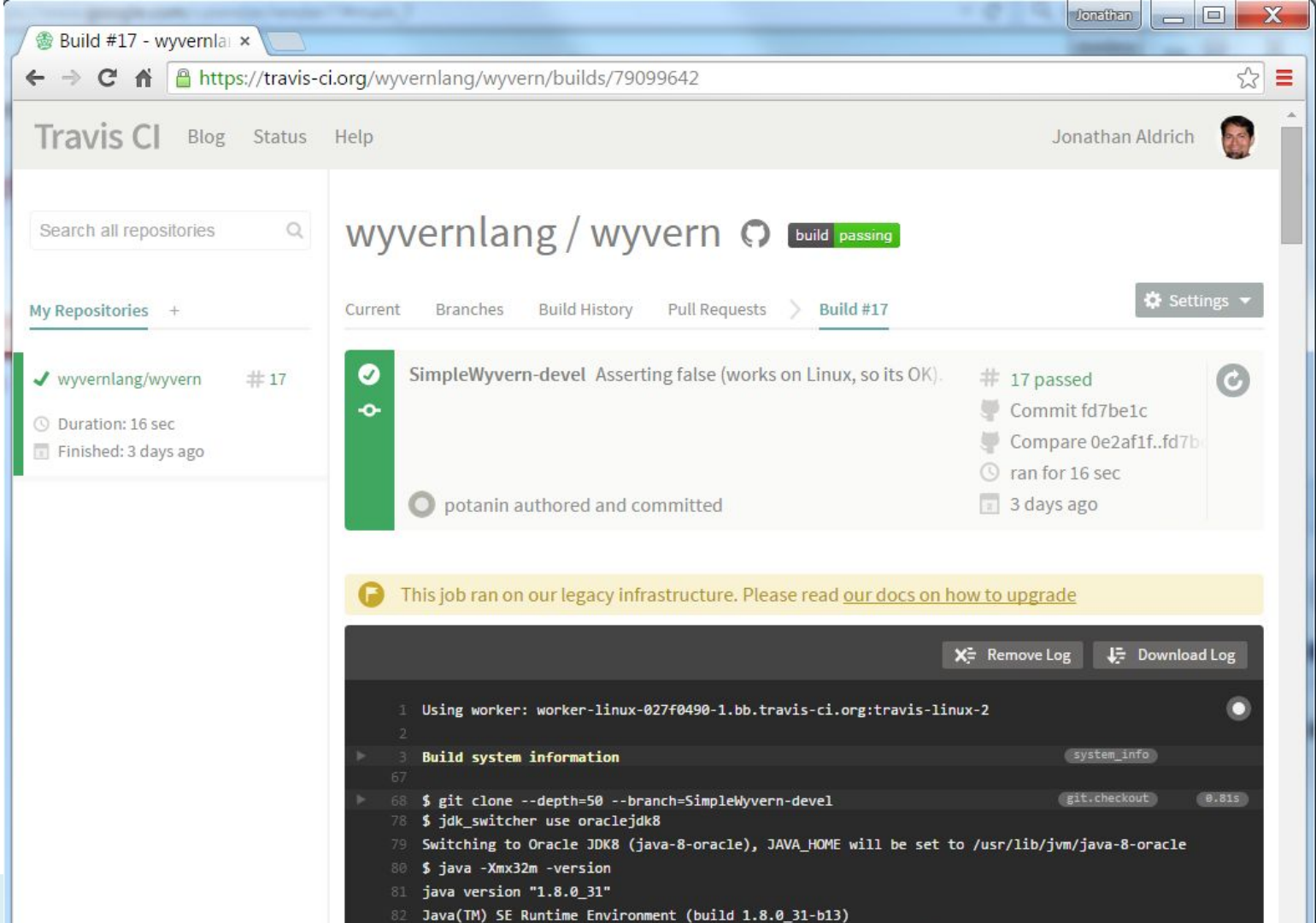
Activity	Estimated	Actual
testing plans	3	0
unit testing	3	1
validation testing	4	2
test data	1	1

How to get developers to write tests?

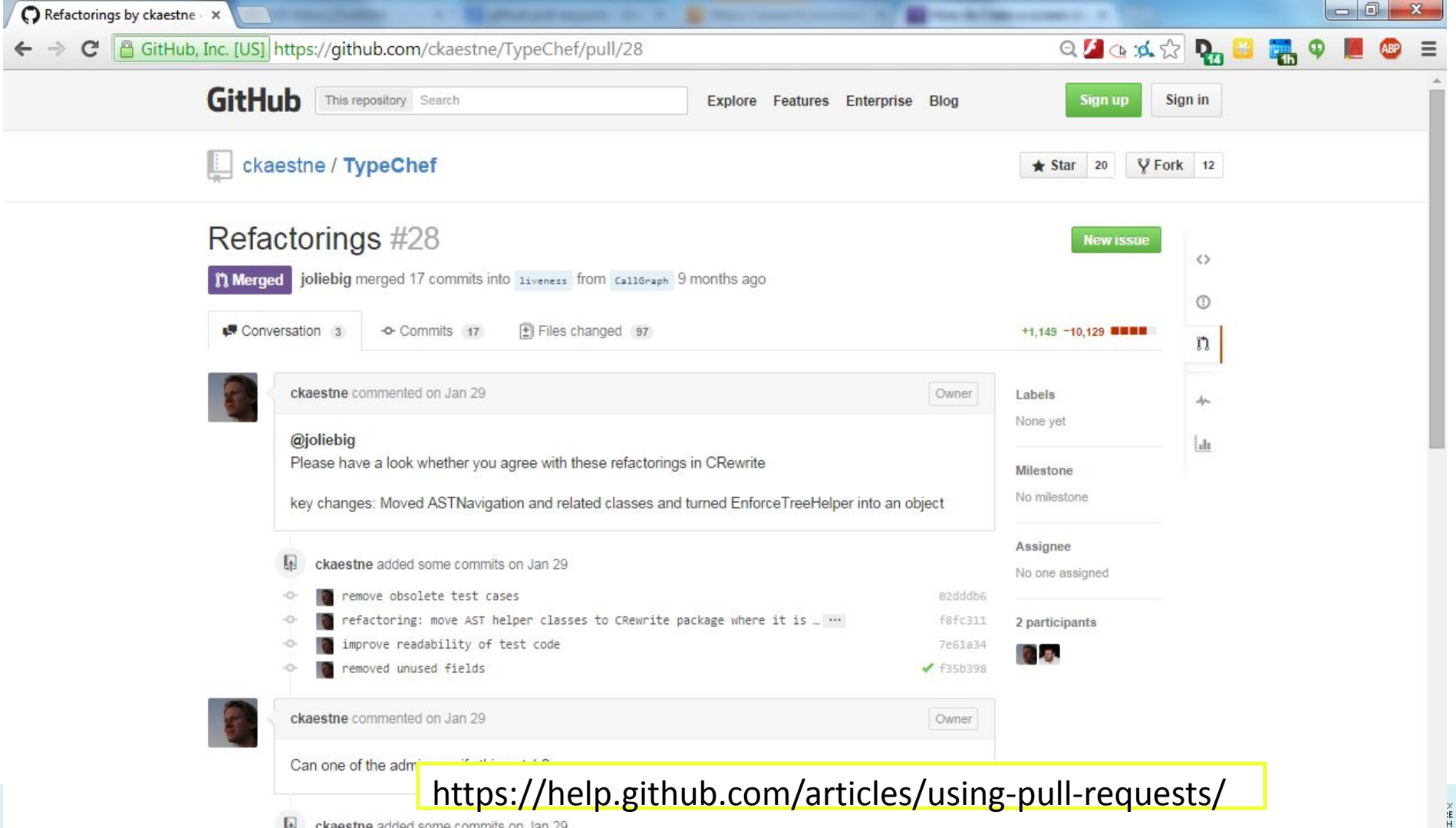
Test Driven Development

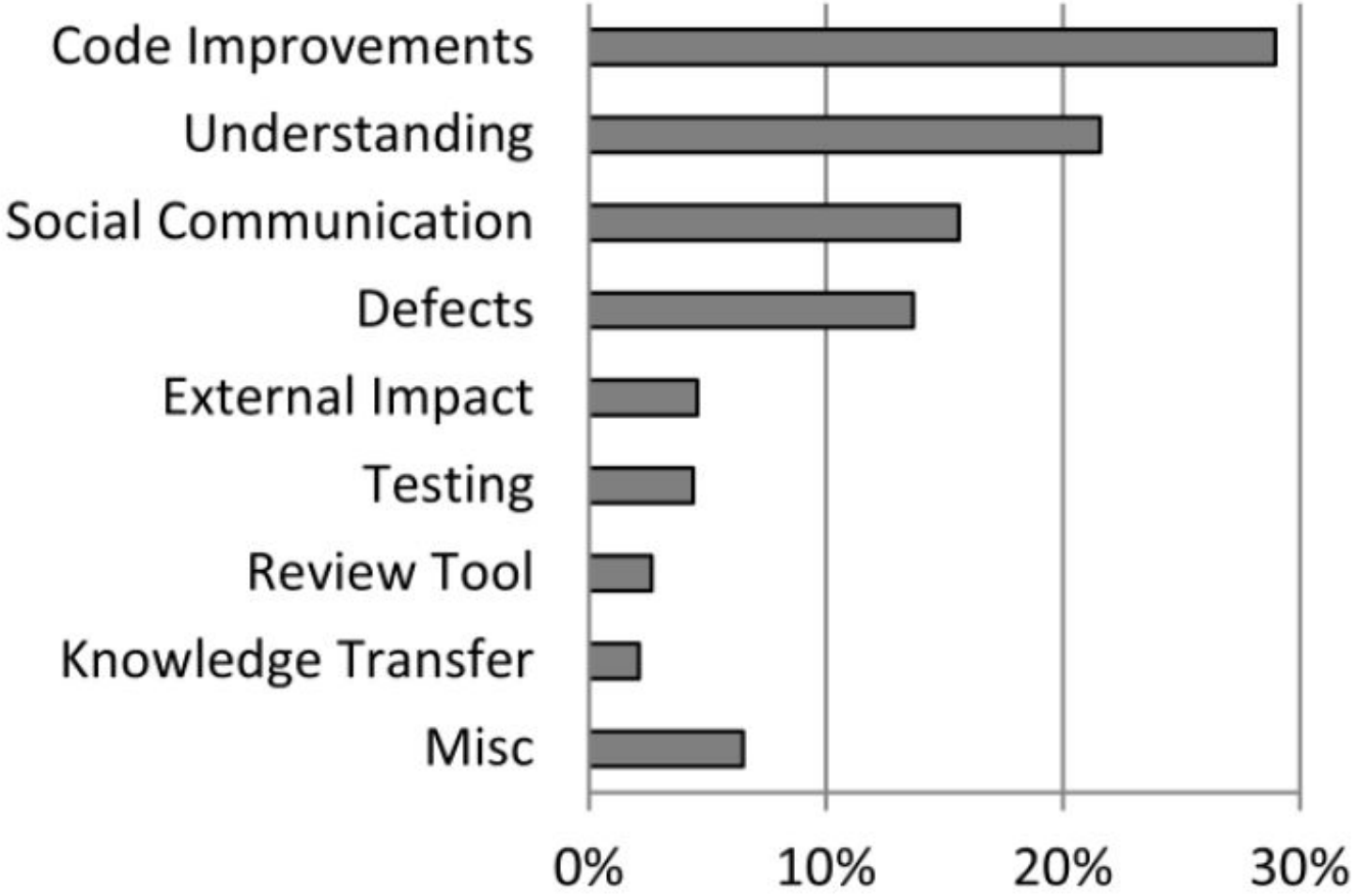
- Tests first!
- Popular agile technique
- Write tests as specifications before code
- Never write code without a failing test
- Claims:
  - Design approach toward testable design
  - Think about interfaces first
  - Avoid writing unneeded code
  - Higher product quality (e.g. better code, less defects)
  - Higher test suite quality
  - Higher overall productivity




Build #17 - wyvernla
https://travis-ci.org/wyvernlang/wyvern/builds/79099642
Travis CI
Blog
Status
Help
Jonathan Aldrich
Search all repositories
wyvernlang / wyvern
build passing
My Repositories
Current
Branches
Build History
Pull Requests
Build #17
Settings
wyvernlang/wyvern
17
Duration: 16 sec
Finished: 3 days ago
SimpleWyvern-devel Asserting false (works on Linux, so its OK).
potanin authored and committed
17 passed
Commit fd7be1c
Compare 0e2af1f..fd7b
ran for 16 sec
3 days ago
This job ran on our legacy infrastructure. Please read our docs on how to upgrade
Remove Log
Download Log
1 Using worker: worker-linux-027f0490-1.bb.travis-ci.org:travis-linux-2
3 Build system information
system_info
68 $ git clone --depth=50 --branch=SimpleWyvern-devel
git.checkout
0.81s
78 $ jdk_switcher use oraclejdk8
79 Switching to Oracle JDK8 (java-8-oracle), JAVA_HOME will be set to /usr/lib/jvm/java-8-oracle
80 $ java -Xmx32m -version
81 java version "1.8.0_31"
82 Java(TM) SE Runtime Environment (build 1.8.0_31-b13)

How to get developers to use static analysis?


Refactorings by ckaestne
GitHub, Inc. [US] https://github.com/ckaestne/TypeChef/pull/28
GitHub
This repository Search
Explore Features Enterprise Blog
Sign up Sign in
ckaestne / TypeChef
Star 20 Fork 12
Refactorings #28
New issue
Merged joliebig merged 17 commits into liveness from CallGraph 9 months ago
Conversation 3 Commits 17 Files changed 97 +1,149 −10,129
ckaestne commented on Jan 29 Owner
@joliebig
Please have a look whether you agree with these refactorings in CRewrite
key changes: Moved ASTNavigation and related classes and turned EnforceTreeHelper into an object
ckaestne added some commits on Jan 29
remove obsolete test cases 02dddb6
refactoring: move AST helper classes to CRewrite package where it is … f8fc311
improve readability of test code 7e61a34
removed unused fields f35b398
ckaestne commented on Jan 29 Owner
Can one of the adm
Labels
None yet
Milestone
No milestone
Assignee
No one assigned
2 participants
https://help.github.com/articles/using-pull-requests/


Code Improvements
Understanding
Social Communication
Defects
External Impact
Testing
Review Tool
Knowledge Transfer
Misc
0%
10%
20%
30%

How to get developers to use static analysis?

```
package com.google.devtools.staticanalysis;

public class Test {
```

▾ Lint — Missing a Javadoc comment.
Java
1:02 AM, Aug 21

Please fix — Not useful

```
  public boolean foo() {
    return getString() == "foo".toString();
```

▾ ErrorProne — String comparison using reference equality instead of value equality (see http://code.google.com/p/error-prone/wiki/StringEquality)
StringEquality
1:03 AM, Aug 21

Please fix

Suggested fix attached: show — Not useful

```
  }

  public String getString() {
```

Are code reviews worth it?

Advertisement: SE @ CMU

Many courses

Spring: SE for Startups, ML in Production, Program Analysis, WebApps, Foundations of SE
Fall: Foundations of SE, (sometimes) API Design

Master level: Formal methods, Requirements, Architecture, Agile, QA, DevOps, Software Project Mgmt, Scalable Systems, Embedded Sys., …
Technical foundations: ML, Distributed Systems

Many research opportunities -- contact us for pointers

https://www.cmu.edu/scs/isr/reuse/
https://se-phd.isri.cmu.edu/

Software Engineering Concentration / Minor

Summary

Looking back at one semester of code-level design, testing, and concurrency

Looking forward to human aspects of software engineering, including process and requirements