Principles of Software Construction: Objects, Design, and Concurrency

The Last One: Locking Back & Looking Forward

Bogdan Vasilescu Jonathan Aldrich **Christian Kästner (surprise appearance)**

Looking Back at the Semester

Principles of Software Construction: Objects, Design, and Concurrency

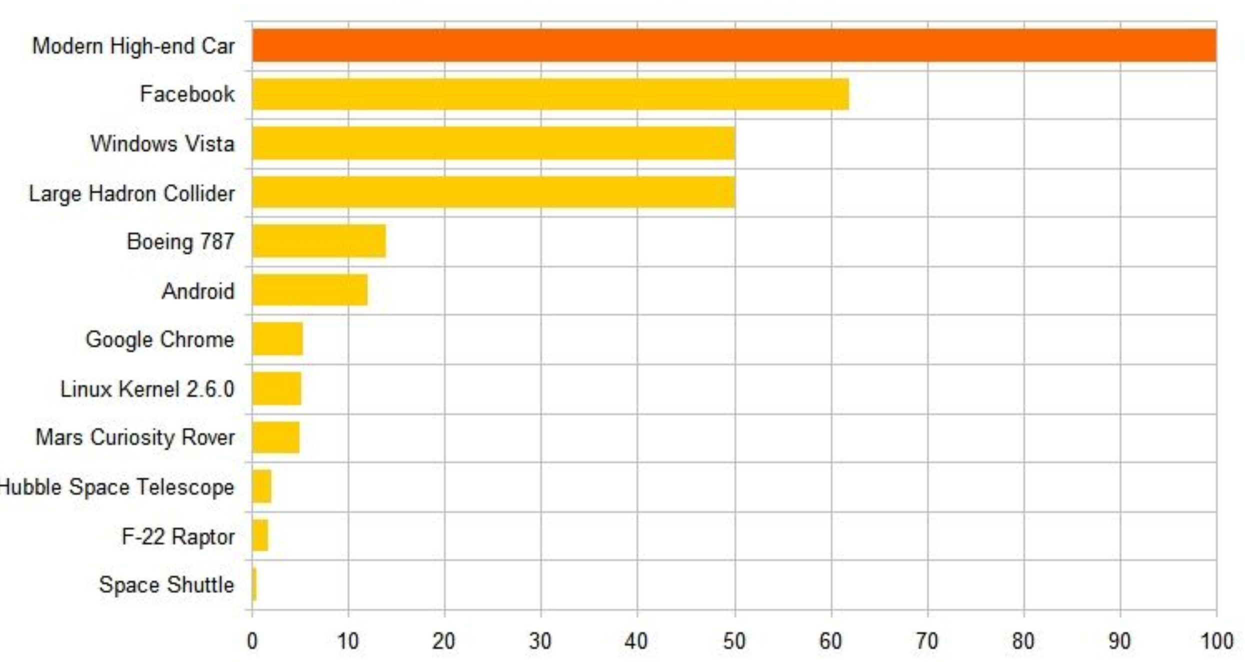
Introduction, Overview, and Syllabus

Bogdan Vasilescu **Jonathan Aldrich**

How Modern Software Gets Built

"Building software is like constructing a building. A construction company wouldn't build its hammers and drills from scratch, or source and chop all of the lumber themselves."

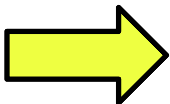
Welcome to the era of “big code”



(informal reports)

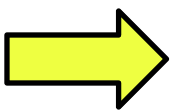
From Programs to Applications and Systems

Writing algorithms, data structures from scratch



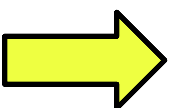
Reuse of libraries, frameworks

Functions with inputs and outputs



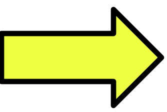
Asynchronous and reactive designs

Sequential and local computation



Parallel and distributed computation

Full functional specifications

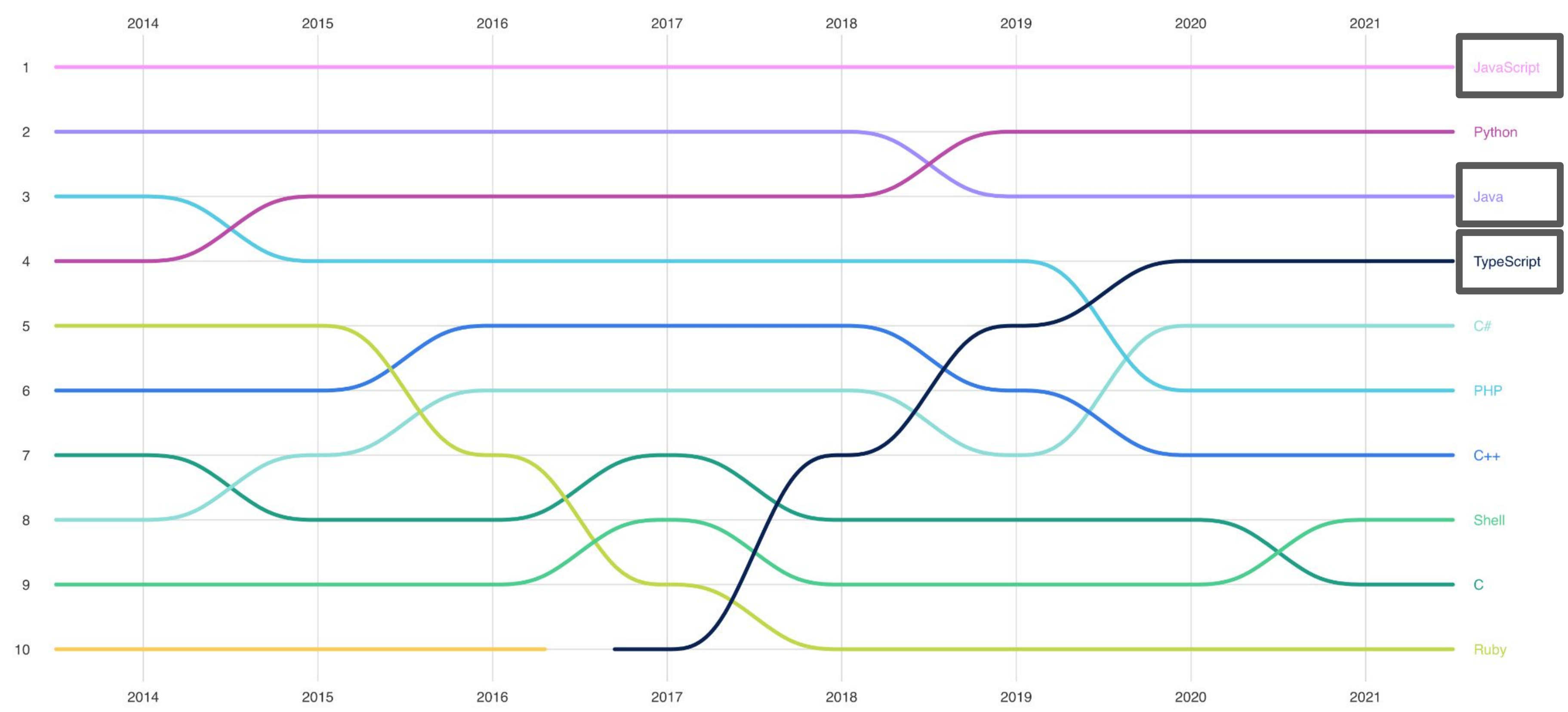


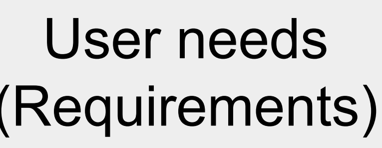
Partial, composable, targeted models

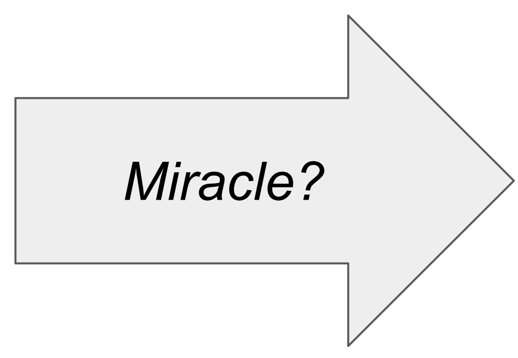
Our goal: understanding both the **building blocks** and also the **design principles** for construction of software systems **at scale**

Top languages over the years

2021 GitHub State of the Octoverse report







Code

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
    }
    …
}
```

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

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

Version B':

it depends

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

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

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

Jonathan Aldrich Bogdan Vasilescu

Where we are

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

Interfaces and Objects i

```java
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
```

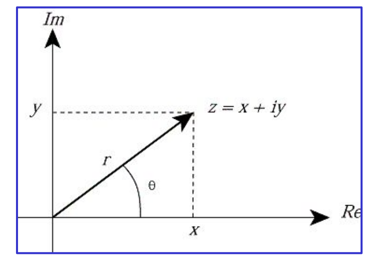
```typescript
interface Counter {
    v: number;
    inc(): void;
    get(): number;
    add(y: number): number;
}
const obj: Counter = {
    v: 1,
    inc: function() { this.v++; },
    get: function() { return this.v; },
    add: function(y) { return this.v + y; }
}
```

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

Multiple Implementations of Interface

This is Java code!

```
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);
```



Check your Understanding

```
interface Animal {
    void makeSound();
}
class Dog implements Animal {
    public void makeSound() { System.out.println("bark!"); } }
class Cow implements Animal {
    public void makeSound() { moo(); }
    public void moo() {System.out.println("moo!"); } }
Animal x = new Animal() {
    public void makeSound() { System.out.println("chirp!"); }}
x.makeSound(); // “chirp”

Animal d = new Dog();
d.makeSound(); // “bark!”
Animal b = new Cow();
b.makeSound(); // “moo!”
b.moo(); // compile-time error
```

```
Animal a = new Animal();
a.makeSound(); // compile-time error
```

JavaScript: Closures for Hiding

All methods and fields are public, no language constructs for access control

TypeScript added them, so it's quite similar to Java!

In JS: Encoding hiding with closures

```
function createPolarPoint(len, angle) {
    let xcache = -1;
    let internalLen=len;
    function computeX() {…}
    return {
        getX: function() {
            computeX(); return xcache; },
        getY: function() {
            return len * sin(angle); }
    };
}
const pp = createPolarPoint(1, 0);
pp.getX(); // works
pp.computeX(); // runtime error
pp.xcache // undefined
pp.len // undefined
```

How to hide information?

```
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();
}
```

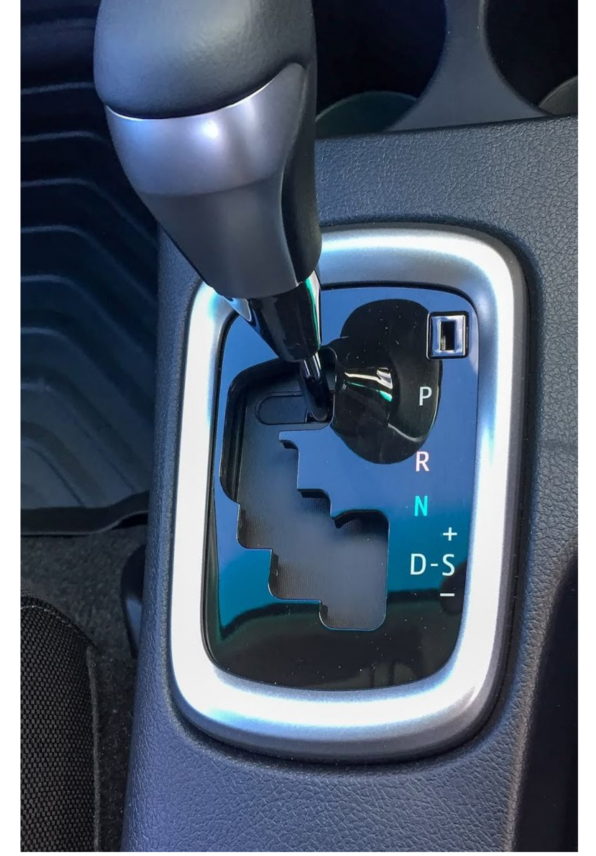
```
const point = {
    x: 1, y: 0,
    getX: function() {…}
    helper_getAngle:
        function() {…}
}
```

Principles of Software Construction: Objects, Design, and Concurrency

IDEs, Build system, Continuous Integration, Libraries

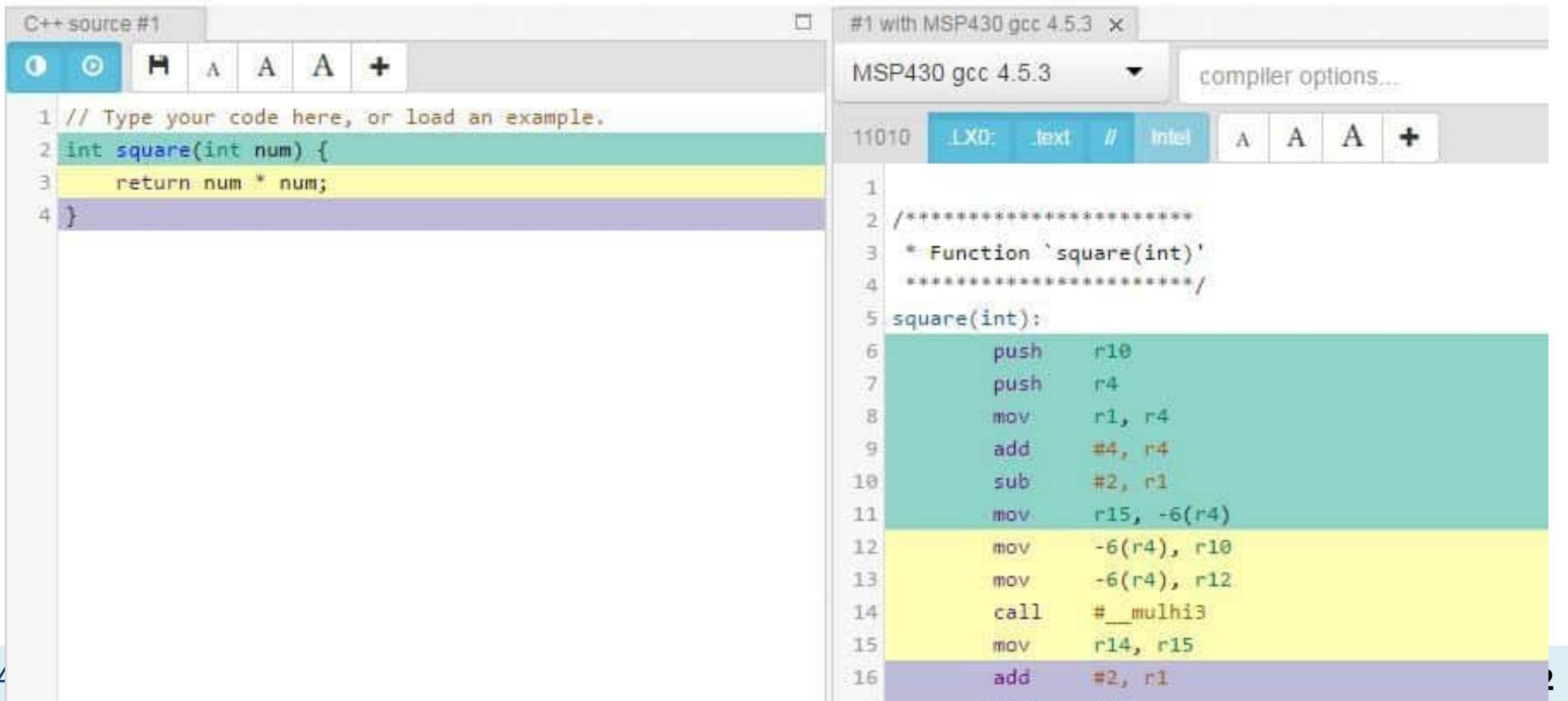
Bogdan Vasilescu **Jonathan Aldrich**

Productivity Requires Automation Requires Abstraction



Quick overview of today's toolchain: Build Systems

How does this happen?



S3D

This is Java code!

Starting a program: Java

All Java code is in classes, so how to create an object and call a method?

```
// start with: java Printer
class Printer {
    void print() {
        System.out.println("hi");
    }
    public static void main(String[] args) {
        Printer obj = new Printer();
        obj.print();
    }
}
```

in Java, everything is a class

main must be public and static

s (`java X` calls

Main method to be executed, here used to create object and invoke method

Static methods belong to class not the object, generally avoid them

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

...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

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

```
31 <version>RELEASE</version>
32 <scope>test</scope>
33 </dependency>
```

project › dependencies › dependency

Build Dependencies

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

- Node.js is a JS runtime. npm is its package manager.

package.json — claire-hw1-js

package.json 1, M

package.json > { } dependencies

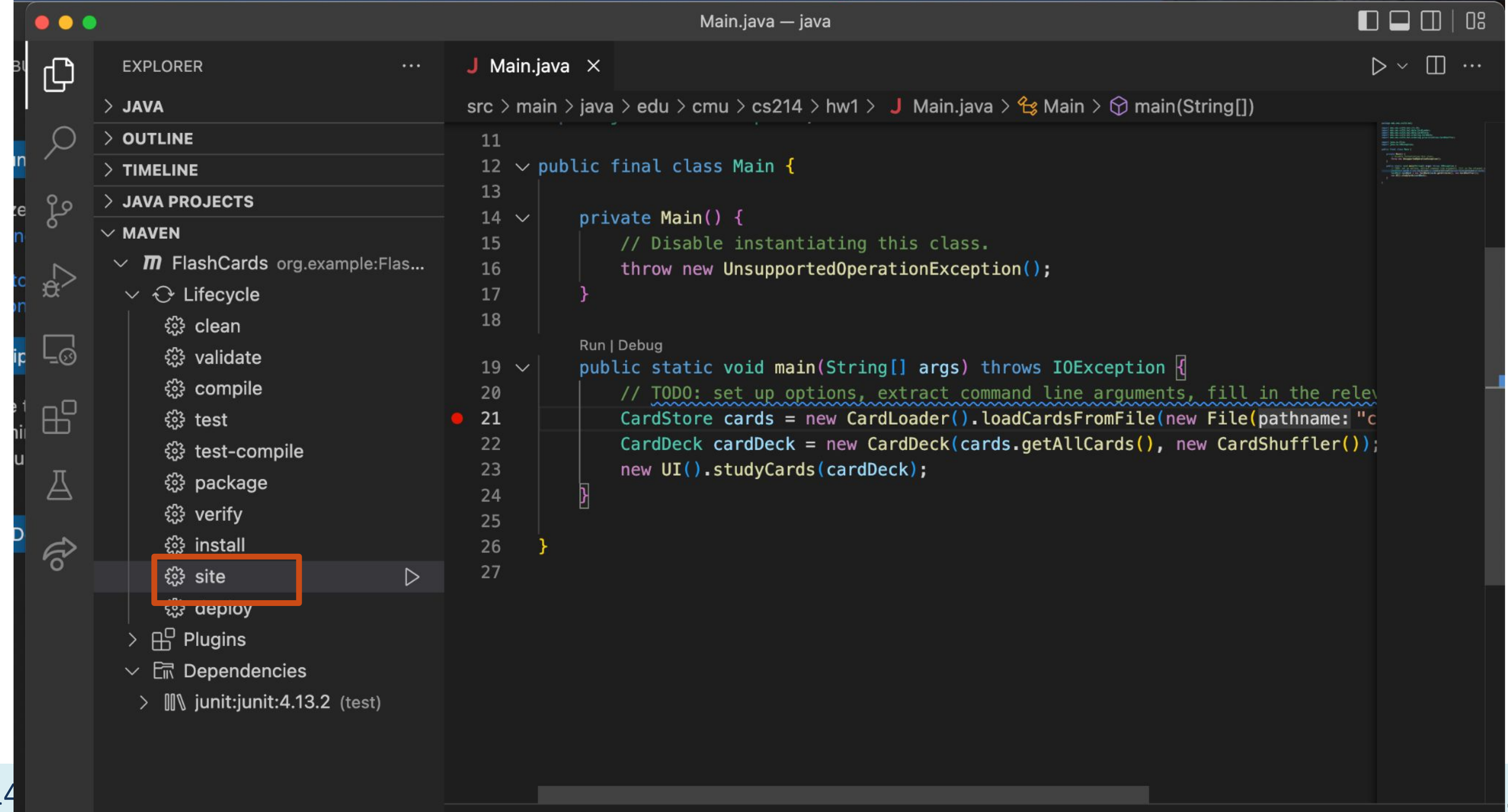
```json
{
  "name": "hw1-flashcards",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "compile": "tsc",
    "lint": "ts-standard",
    "start": "node dist/index.js"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@types/node": "^17.0.8",
    "@types/readline-sync": "^1.4.4",
    "ts-standard": "^10.0.0",
    "typescript": "^4.4.2"
  },
  "dependencies": {
    "readline-sync": "^1.4.10",
  }
}
```

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

Combine build systems + IDEs + plugins (checkstyle example/demo!)

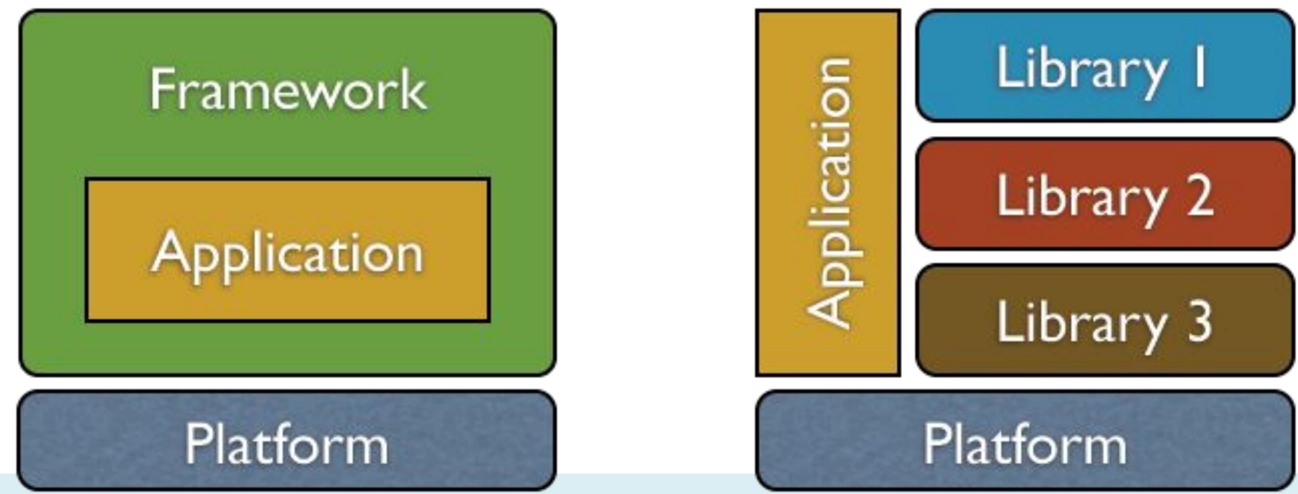


Under the Hood: Libraries & Frameworks

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`

Under the Hood: Continuous Integration

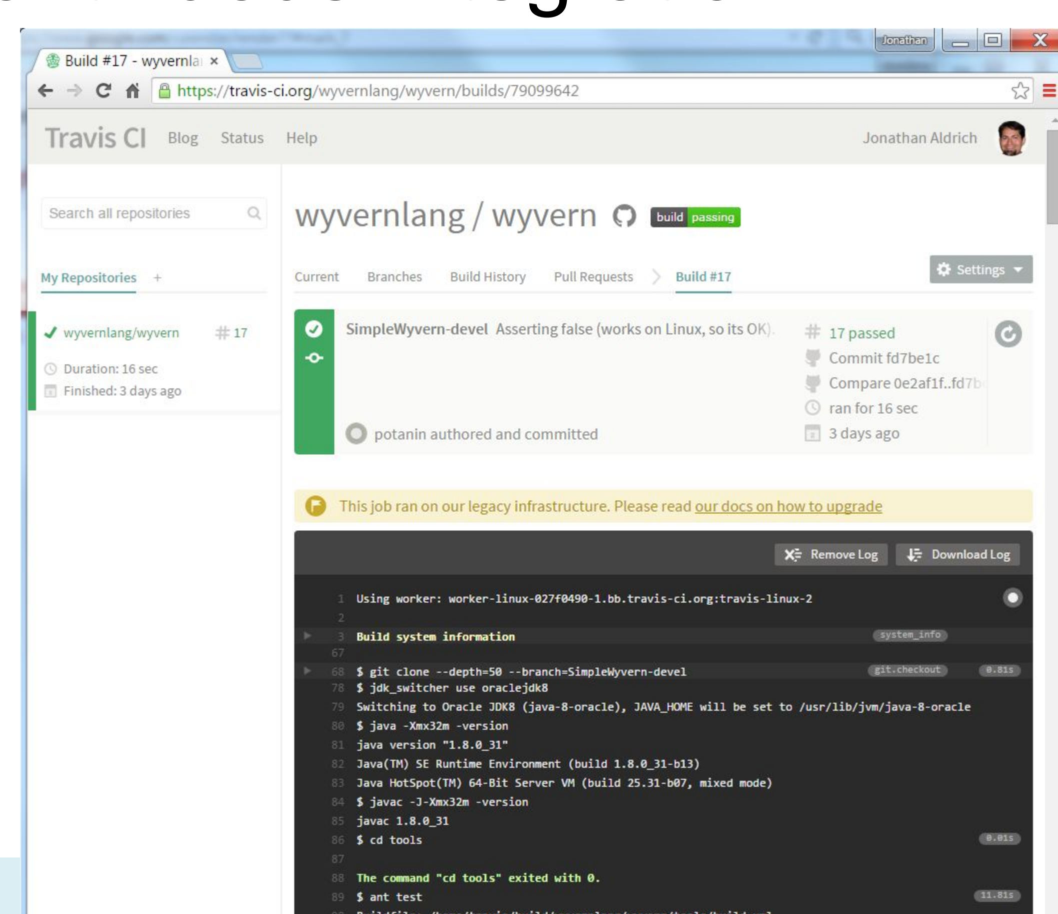
Automatically builds, tests, and displays the result

We – and everyone else – used to use Travis CI.

- Until they randomly stopped supporting OSS.

GitHub has native CI support, and it's pretty good: GitHub Actions.

- Sidebar on how our GH Actions are configured for HW1



HW1: Extending the Flash Card System

Principles of Software Construction: Objects, Design, and Concurrency

Specifications and unit testing, exceptions

Bogdan Vasilescu **Jonathan Aldrich**

Where we are

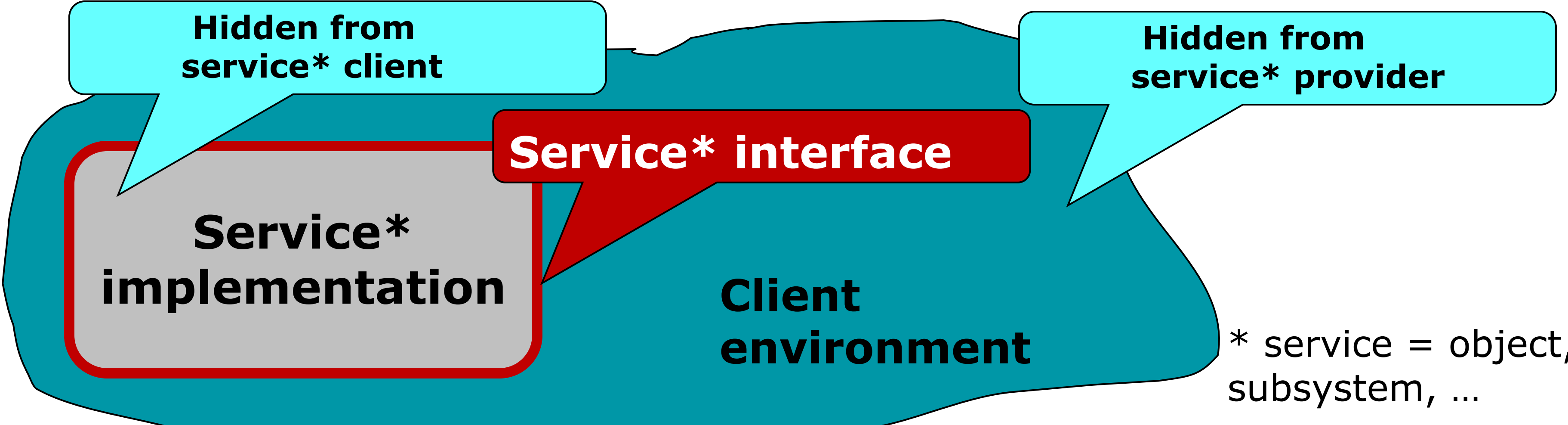
	Small scale: One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
Design for understanding change/ext. reuse robustness ...	Subtype Polymorphism ✓ Information Hiding, **Contracts ✓** Immutability ✓ Types ✓ Static Analysis ✓ **Unit Testing ✓**	Domain Analysis ✓ Inheritance & Del. ✓ Responsibility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

Who’s to blame?

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

> ArrayOutOfBoundsException
```

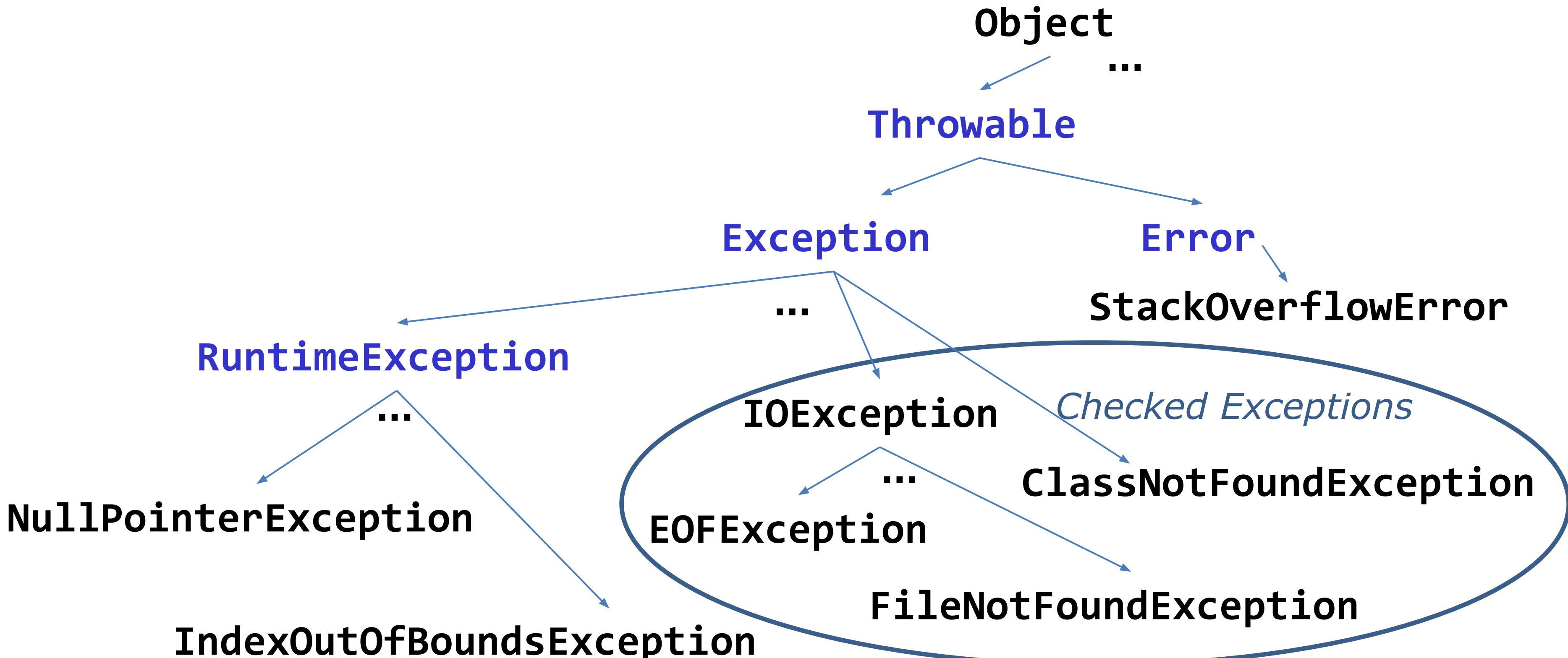
Most real-world code has a contract

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



* service = object, subsystem, …

Java's exception hierarchy (messy)



Testing

This is Java code

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!
  }
}
```

Specification vs. Structural Testing

This is Java code

```java
/**
 * 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) {
  return card.getSuccesses.get(0);  // <-- Bad, but passes both tests
}
```

Principles of Software Construction: Objects, Design, and Concurrency

Test case design

Bogdan Vasilescu Jonathan Aldrich

CreditWallet.pay()

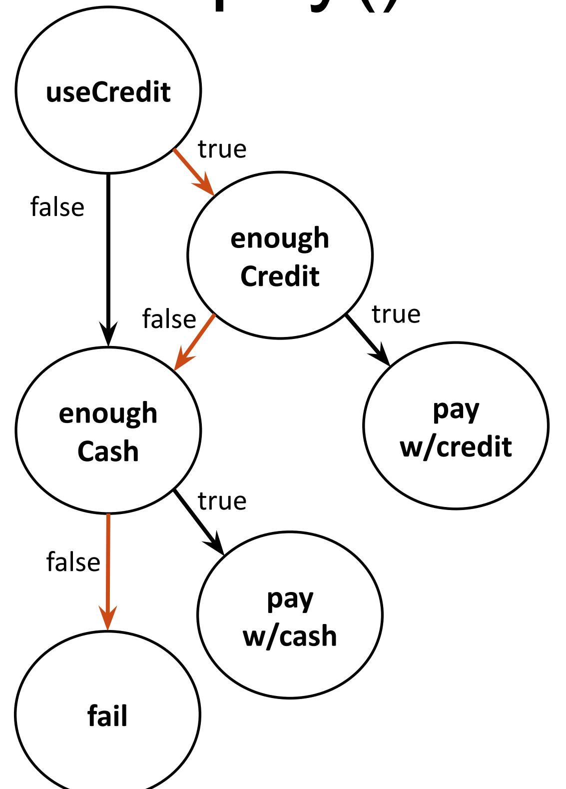
```
public boolean pay(int cost, boolean useCredit) {
    if (useCredit) {
        if (enoughCredit) {
            return true;
        }
    }
    if (enoughCash) {
        return true;
    }
    return false;
}
```

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



S3D

Writing Testable Code

What is the problem with this?

```
public boolean hasHeader(String path) throws IOException {
   List<String> lines = Files.readAllLines(Path.of(path));
   return !lines.get(0).isEmpty()
}

// to achieve a ‘false’ output without having a test input file:
try {
   Path tempFile = Files.createTempFile(null, null);
   Files.write(tempFile,"\n".getBytes(StandardCharsets.UTF_8));
   hasHeader(tempFile.toFile().getAbsolutePath()); // false
} catch (IOException e) {
   e.printStackTrace();
}
```

Back to Specification Testing

What would you test differently in this situation?

- "if useCredit is set and enough credit is available":
  - Test both true, either/both false
- "pays with cash if enough cash is available; otherwise":
  - Test true, false
- Could to this with as few as three test cases

```
/** Pays with credit if useCredit is set and enough
  * credit is available; otherwise, pays with cash if
  * enough cash is available; otherwise, returns false.
  */
public boolean pay(int cost, boolean useCredit);
```

Structural Testing vs. Specification Testing

You will ***typically have both*** code & (prose) specification

- Test specification, but know that it can be underspecified
- Test implementation, but not to the point that it cannot change
- Use testing strategies that leverage both
  - There is a fair bit of overlap; e.g., BVA yields useful branch coverage

HW 2: Testing the Flash Card System

Principles of Software Construction: Objects, Design, and Concurrency

Object-oriented Analysis

Bogdan Vasilescu Jonathan Aldrich

Where we are

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

Problem Space

(Domain Model)

Solution Space

(Object Model)

- 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**

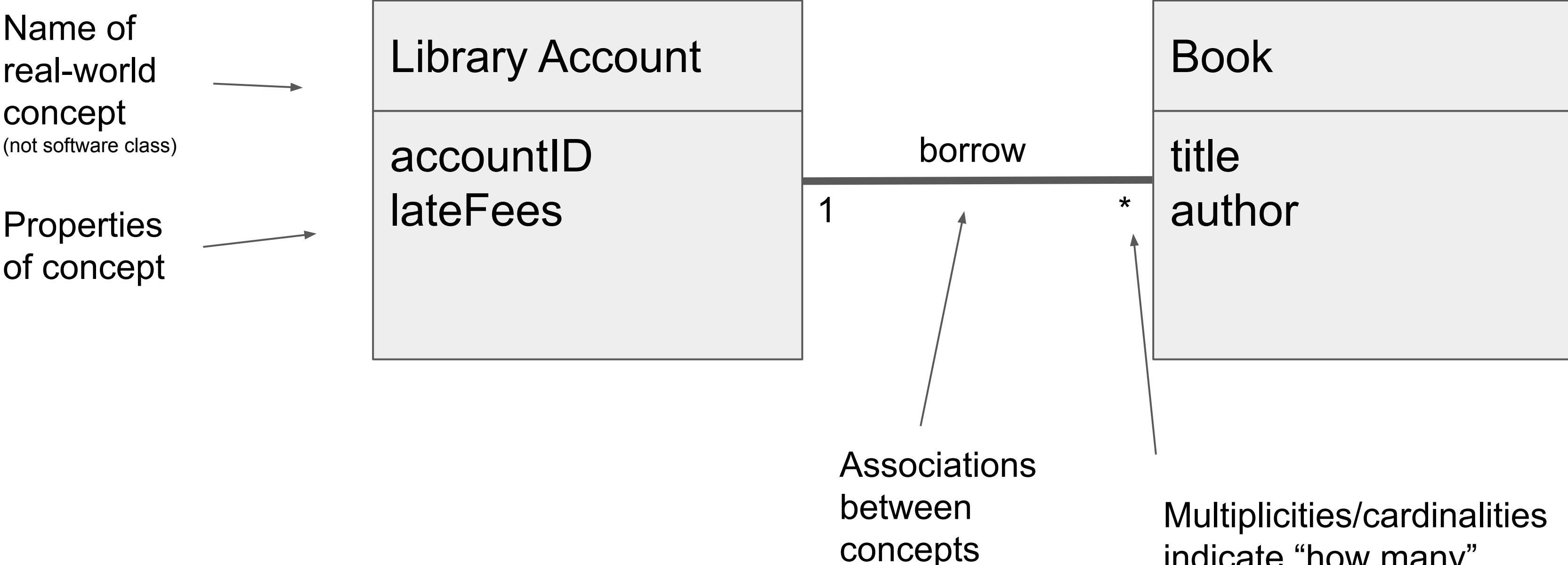
Model / diagram a potential solution

- **Object model**

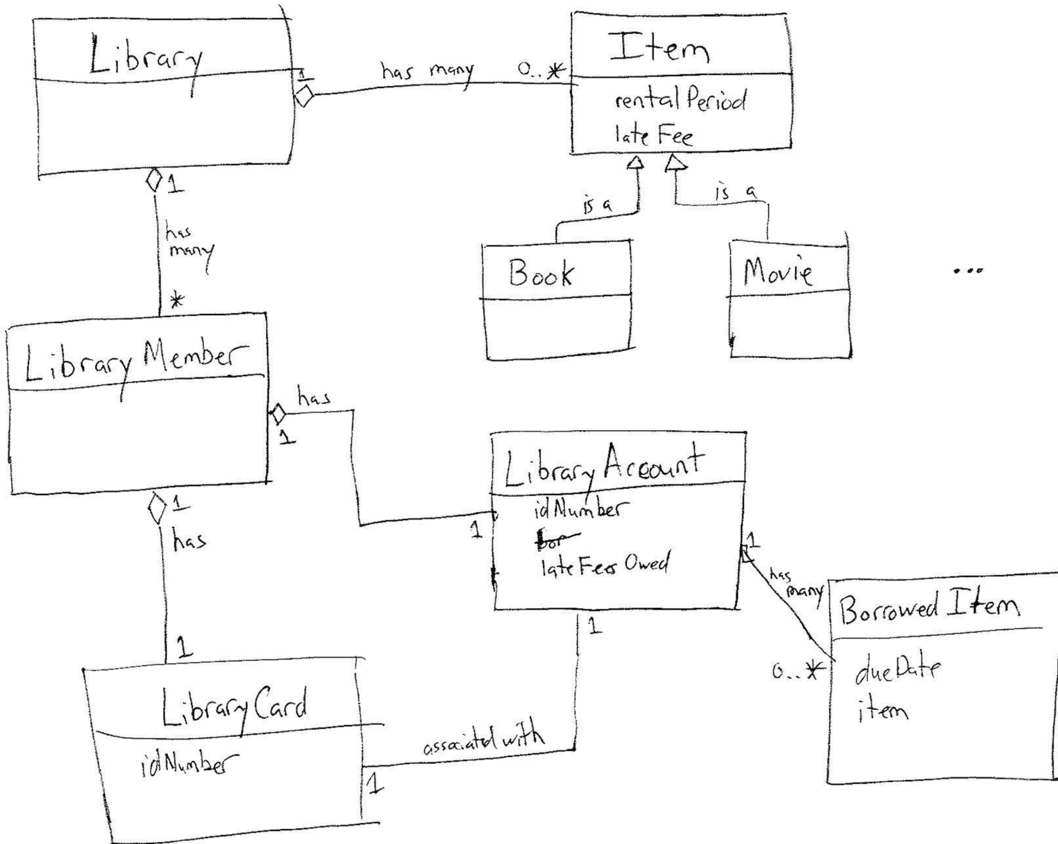
OO Analysis: Understanding the problem

OO Design: Defining a solution

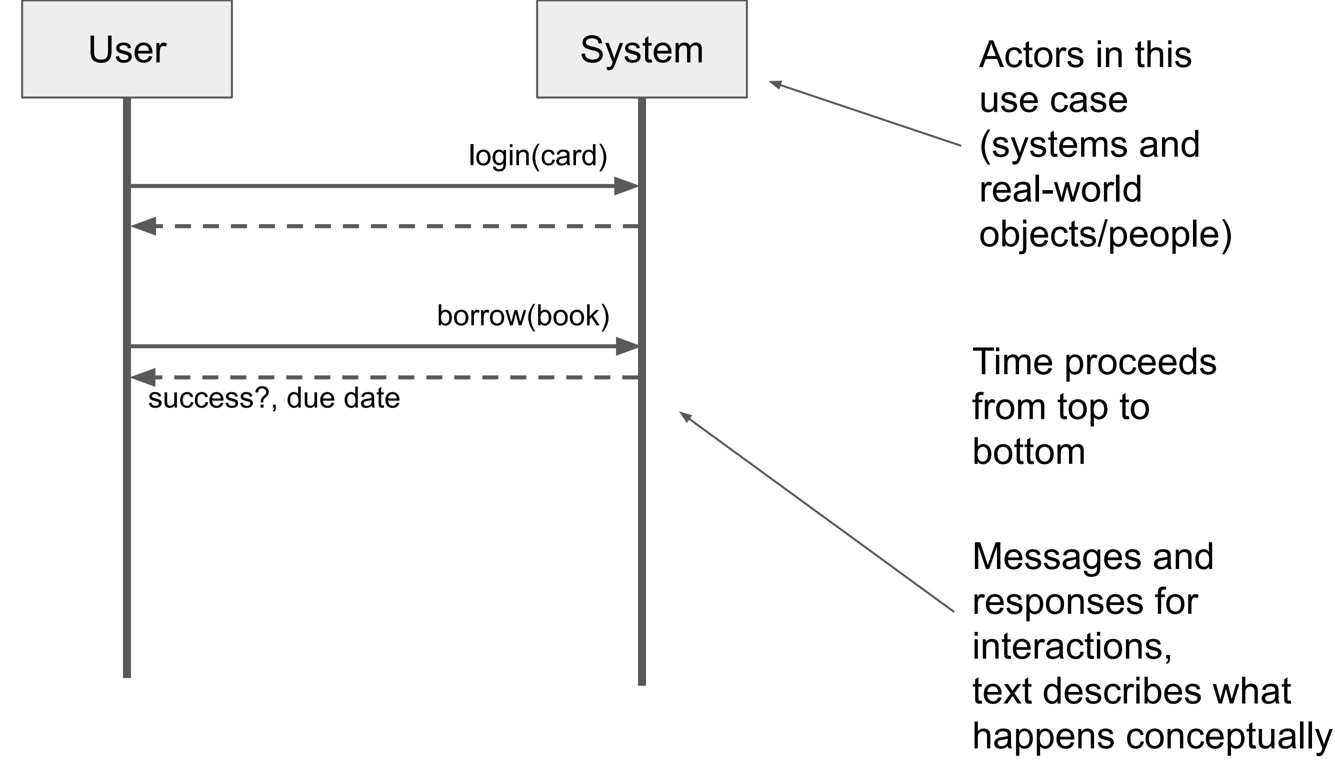
Visual notation: UML



One domain model for the library system

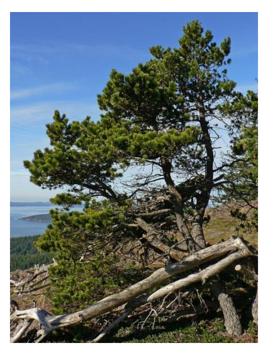


UML Sequence Diagram Notation

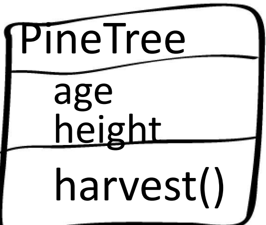


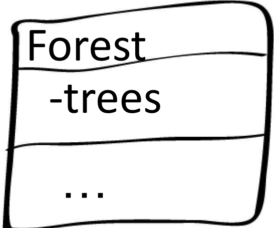
Representational gap

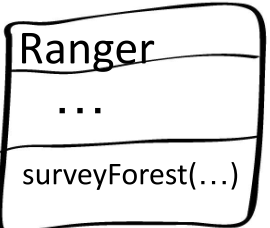
- Real-world concepts:



- Software concepts:







Principles of Software Construction: Objects, Design, and Concurrency

Responsibility Assignment

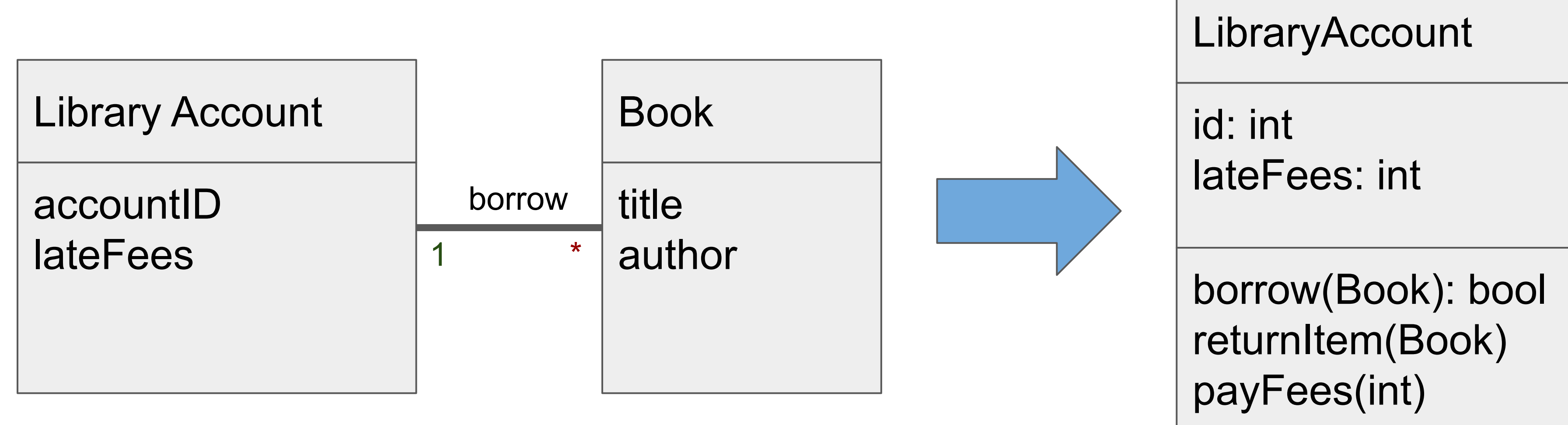
Bogdan Vasilescu Jonathan Aldrich

Where we are

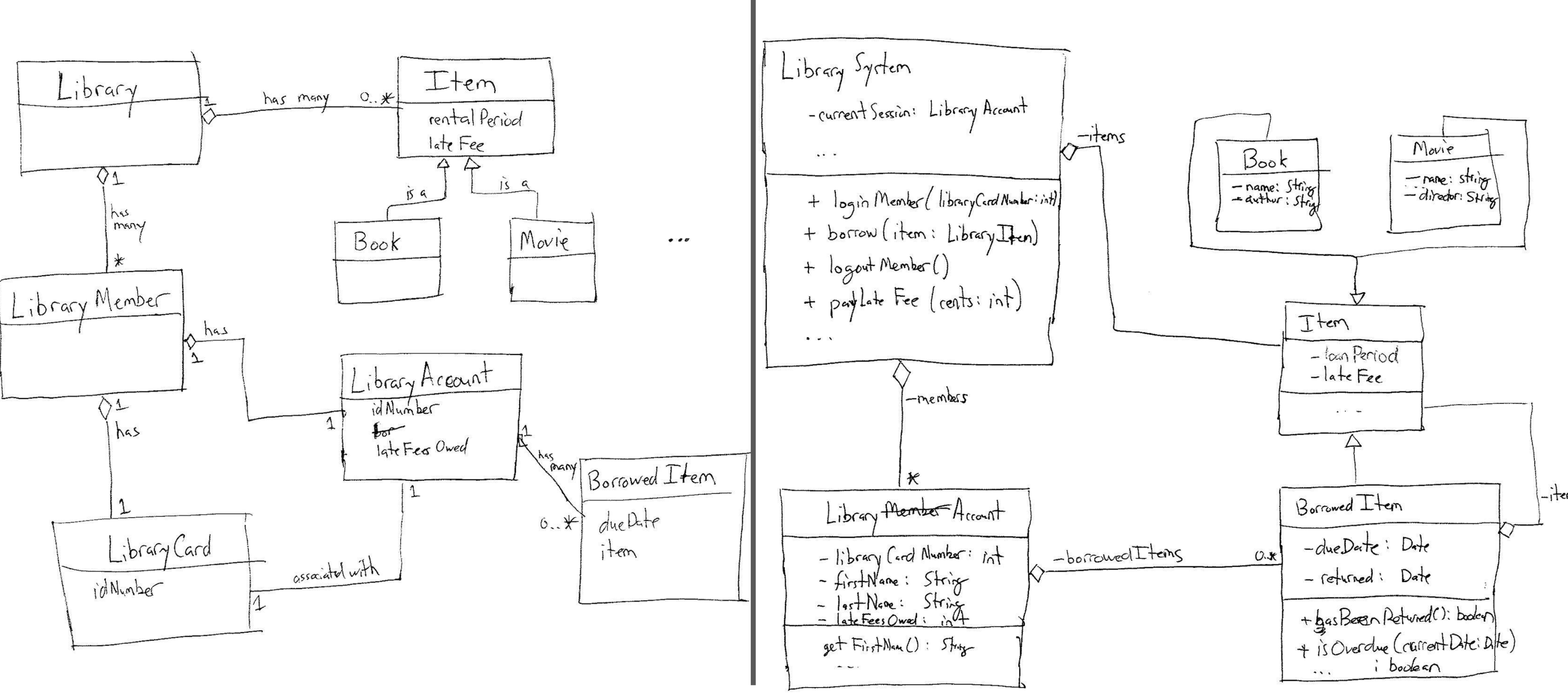
	Small scale: One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
Design for understanding change/ext. reuse robustness ...	Subtype Polymorphism ✓ Information Hiding, Contracts ✓ Immutability ✓ Types ✓ Static Analysis ✓ Unit Testing ✓	Domain Analysis ✓ Inheritance & Del. ✓ **Responsibility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

From concepts to objects

- How are domain concepts different from classes?
  - Should every concept become a class?
  - Does every class need to represent a concept?



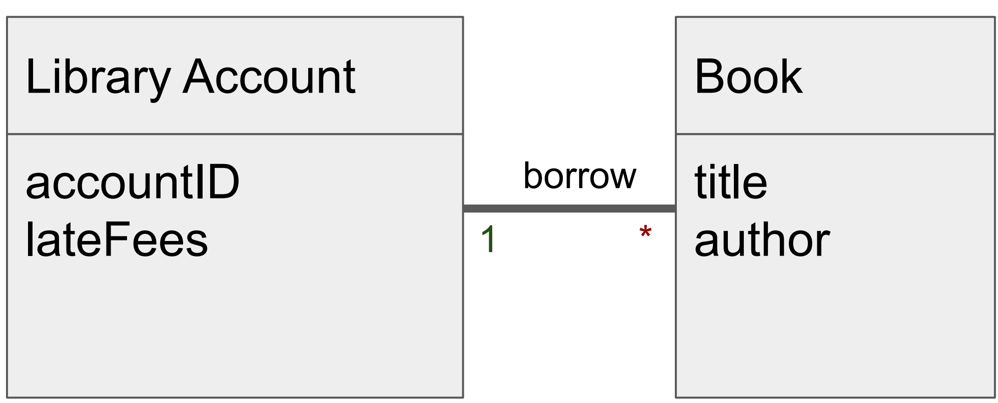
Domain model (left) vs object model (right)



Low Representational Gap

Identified concepts provide inspiration for classes in the implementation

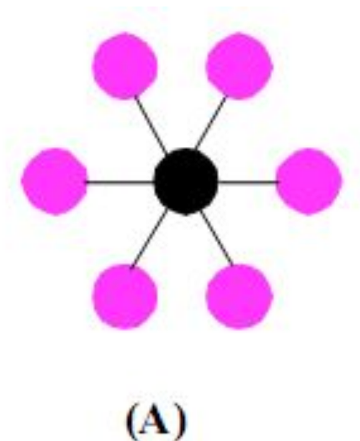
Classes mirroring domain concepts often intuitive to understand, rarely change (low representational gap)



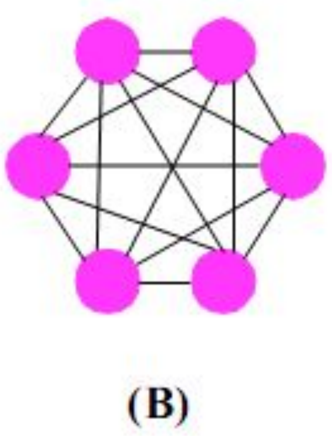

```
class Account {
    id: Int;
    lateFees: Int;
    borrowed: List<Book>;
    boolean borrow(Book) { … }
    void save();
}
class Book { … }
```

Topologies with different coupling

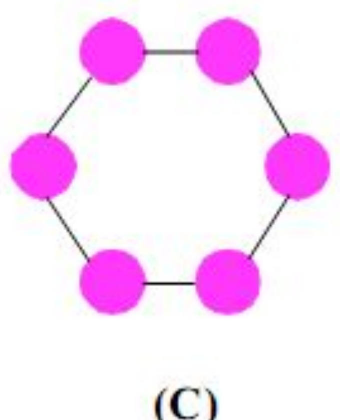
Types of module interconnection structures



(A)



(B)



(C)

S3D

Design Heuristic: Law of Demeter

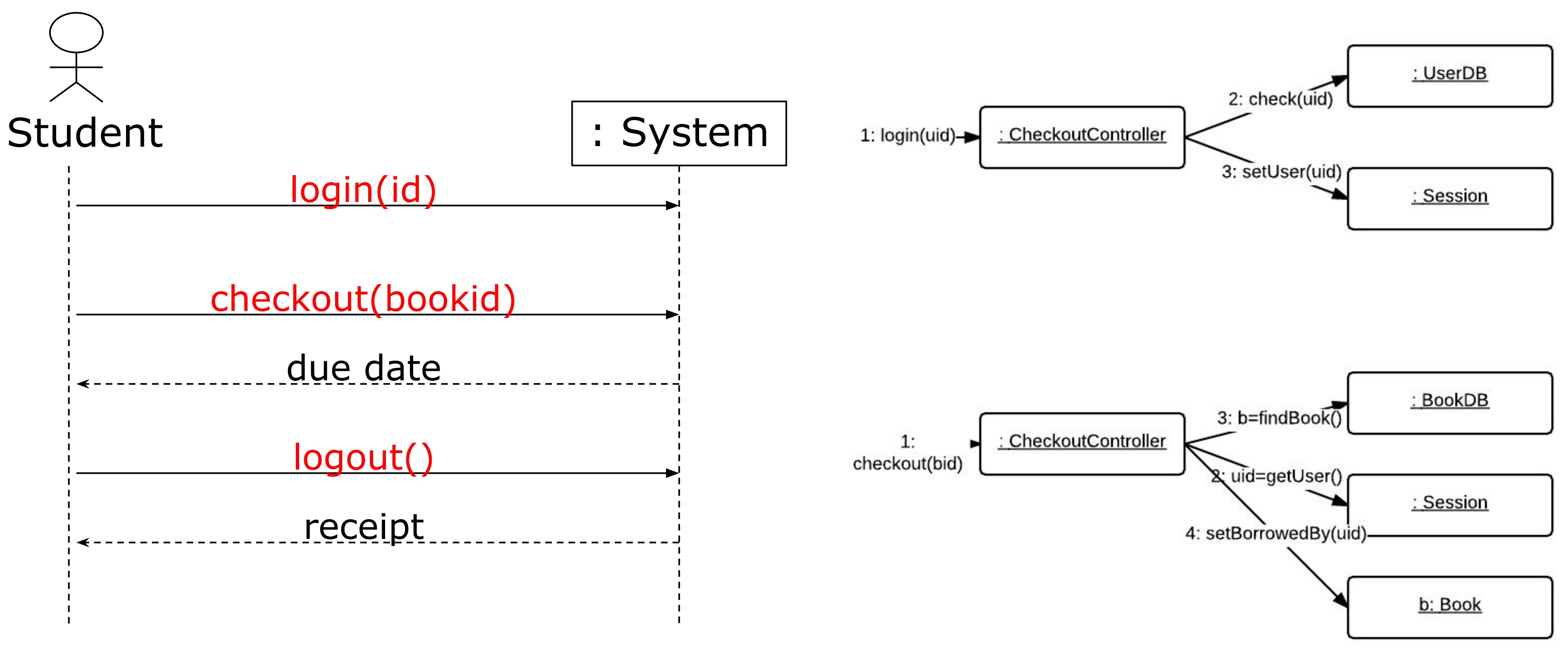
- *Each module should have only limited knowledge about other units: only units "closely" related to the current unit*
- In particular: Don’t talk to strangers!
- For instance, no a.getB().getC().foo()

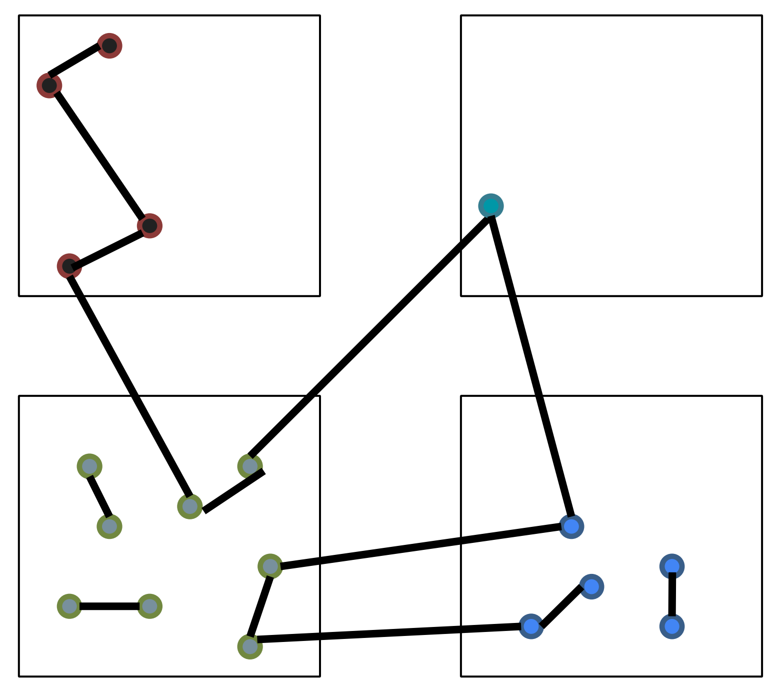
```
for (let i of shipment.getBox().getItems())
    shipmentWeight += i.getWeight() …
```

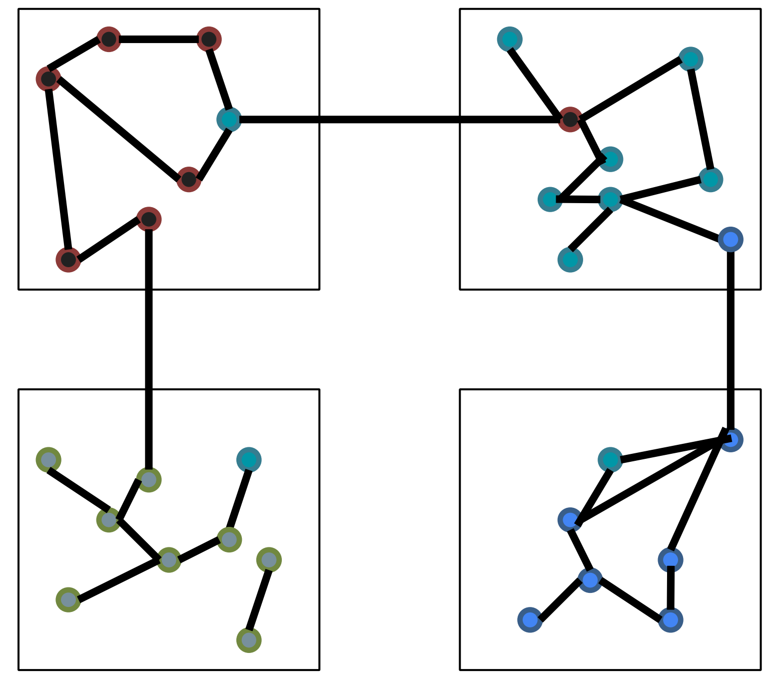
So don’t do this ^ !!

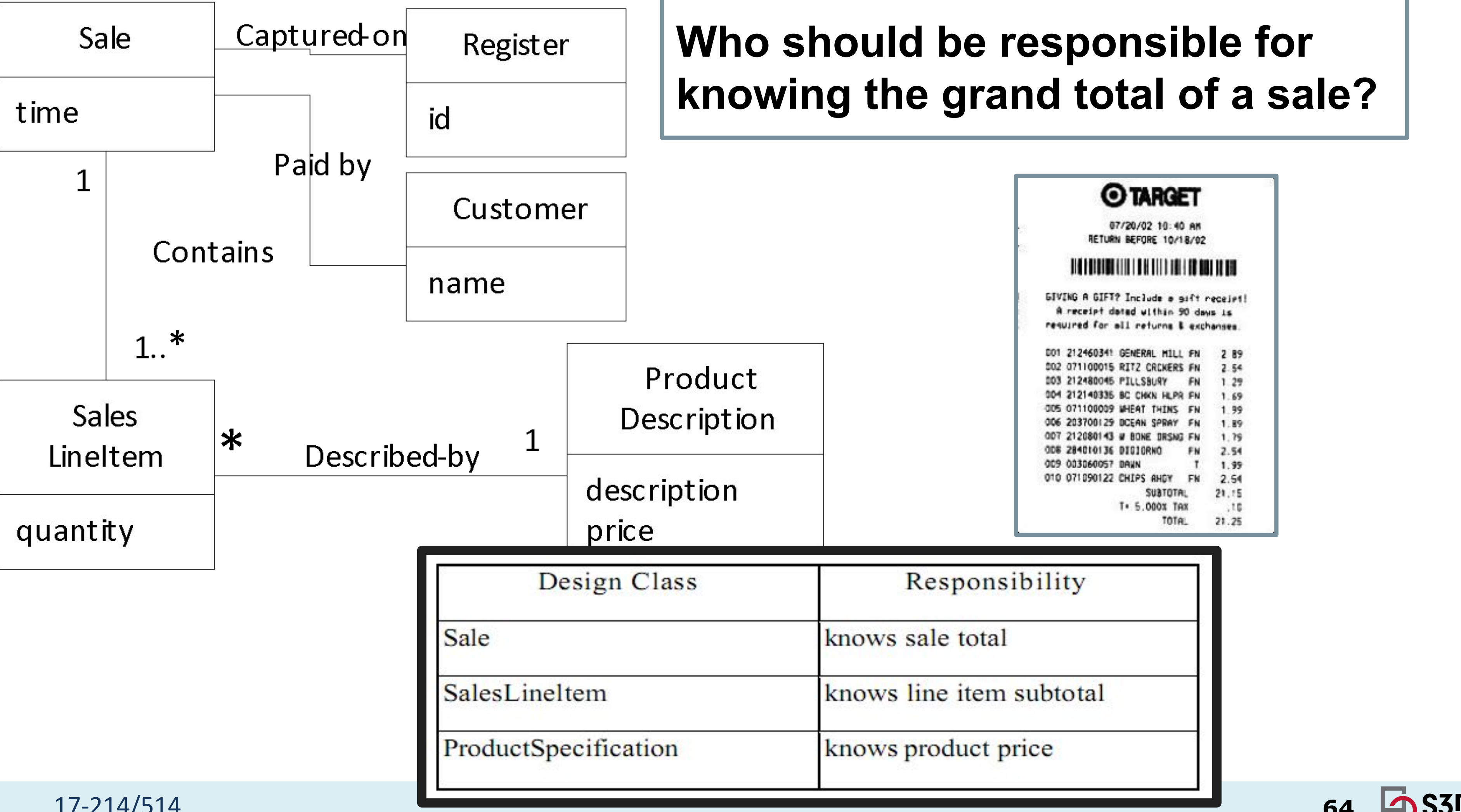
Requirements Analysis

Object-Level Design









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

Anti-Pattern: God Object

```
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;
```

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

Information Expert (Design Heuristic)

- Heuristic: **Assign a responsibility to the class that has the information necessary to fulfill the responsibility**
- Typically follows common intuition
- Software classes instead of Domain Model classes
  - If software classes do not yet exist, look in Domain Model for fitting abstractions (-> correspondence)
- Design process: Derive from domain model (key principles: Low representational gap and low coupling)

HW3: Santorini (Base game)

Principles of Software Construction: Objects, Design, and Concurrency

Inheritance and delegation

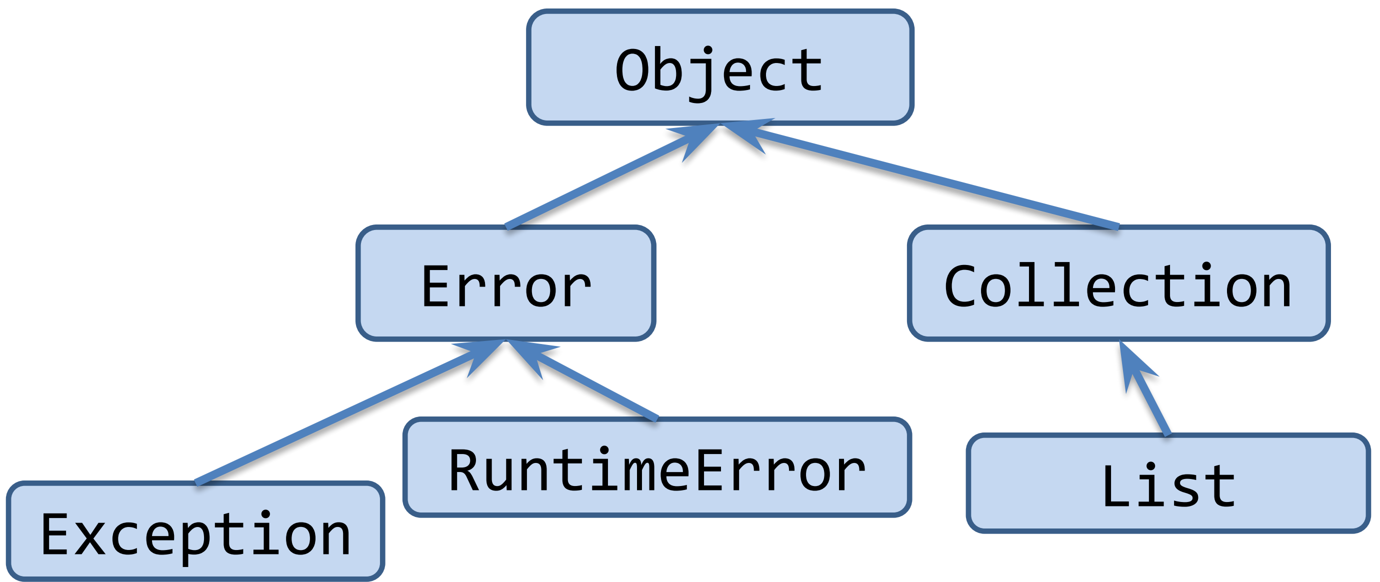
Jonathan Aldrich Bogdan Vasilescu

Where we are

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

All object types exist in a *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”
```

Declared Type

Compile-time Check (Java)

Instantiated Type

Is Square a behavioral subtype of Rectangle?

```
class Rectangle {

   int width;
   int height;

   public Rectangle(int width,
                    int height) {
       this.width = width;
       this.height = height;
   }
   public void scale(int factor) {
       width=width*factor;
       height=height*factor;
   }
}
```

```
public class Square extends Rectangle {

   public Square(int width) {
       super(width, width);
   }

}
```

Design option 3

PaymentCard
CreditCard
DeditCard
CardData

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

    public CardData(…) {…}
    public String getCardHolderName() {…}
    public BigInteger getDigits() {…}
    public Date getExpiration() {…}
}
```

Is this better?

```
class CreditCard implements PaymentCard {
        private CardData cardData = new(…);
        public BigInteger getDigits() {
                return cardData.getDigits();
        }
        …
}
class DebitCard implements PaymentCard {
        …
}
```

You can still achieve good reuse with composition+delegation!

This is the Template Method Design Pattern!

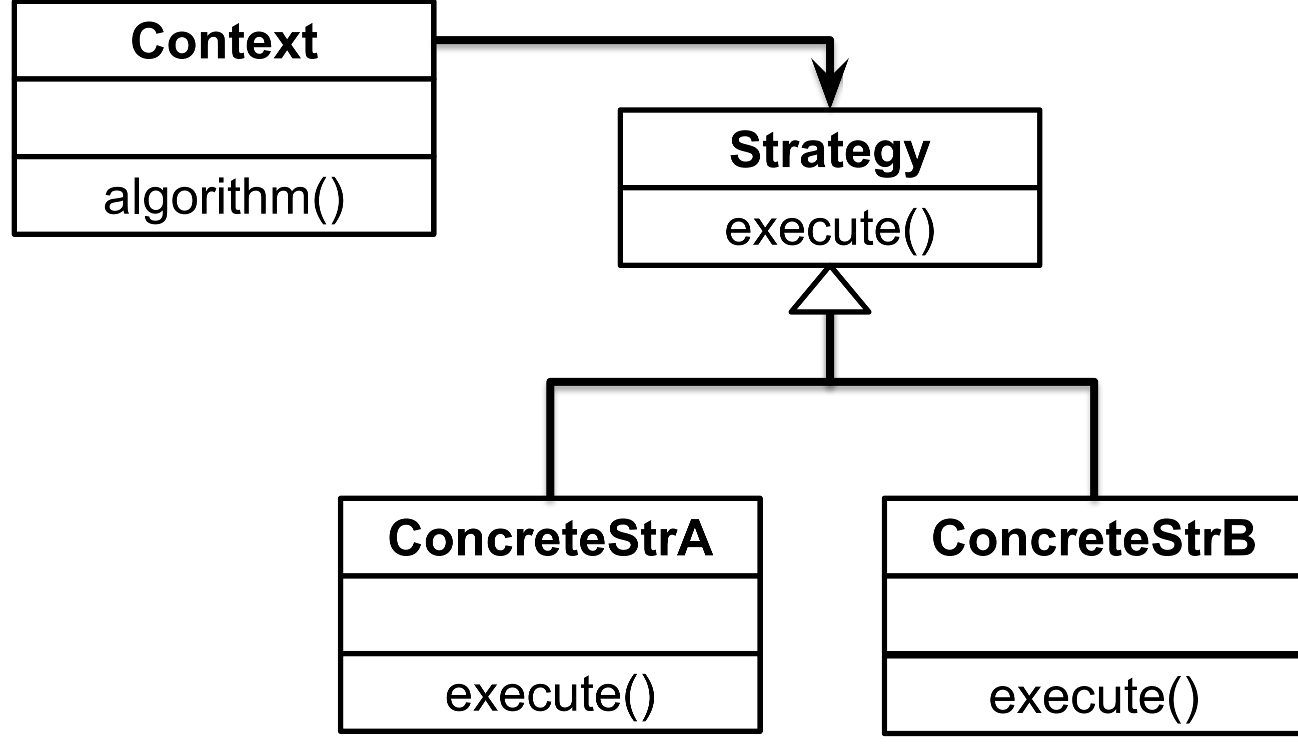
```
abstract class AbstractCashCard
            implements PaymentCard {
   private int balance;
   public AbstractCashCard(int balance) {
       this.balance = balance;
   }

   public boolean pay(int amount) {
       if (amount <= this.balance) {
           this.balance -= amount;
           chargeFee();
           return true;
       }
       return false;
   }
   abstract void chargeFee();
}
```

```
class GiftCard extends AbstractCashCard {
   @Override
   void chargeFee() {
      return; // Do nothing.
   }
}
```

```
class DebitCard extends AbstractCashCard {
   @Override
   void chargeFee() {
      this.balance -= this.fee;
   }
}
```

Design Tradeoffs?

Strategy Pattern in UML.



Principles of Software Construction: Objects, Design, and Concurrency

Design Patterns

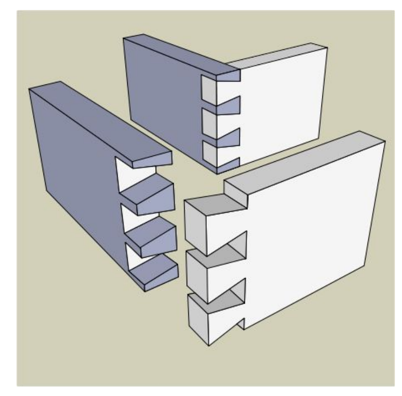
Jonathan Aldrich Bogdan Vasilescu

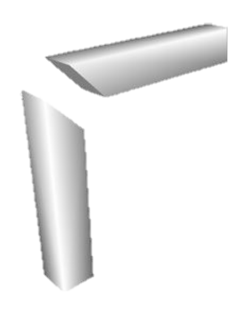
Where we are

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

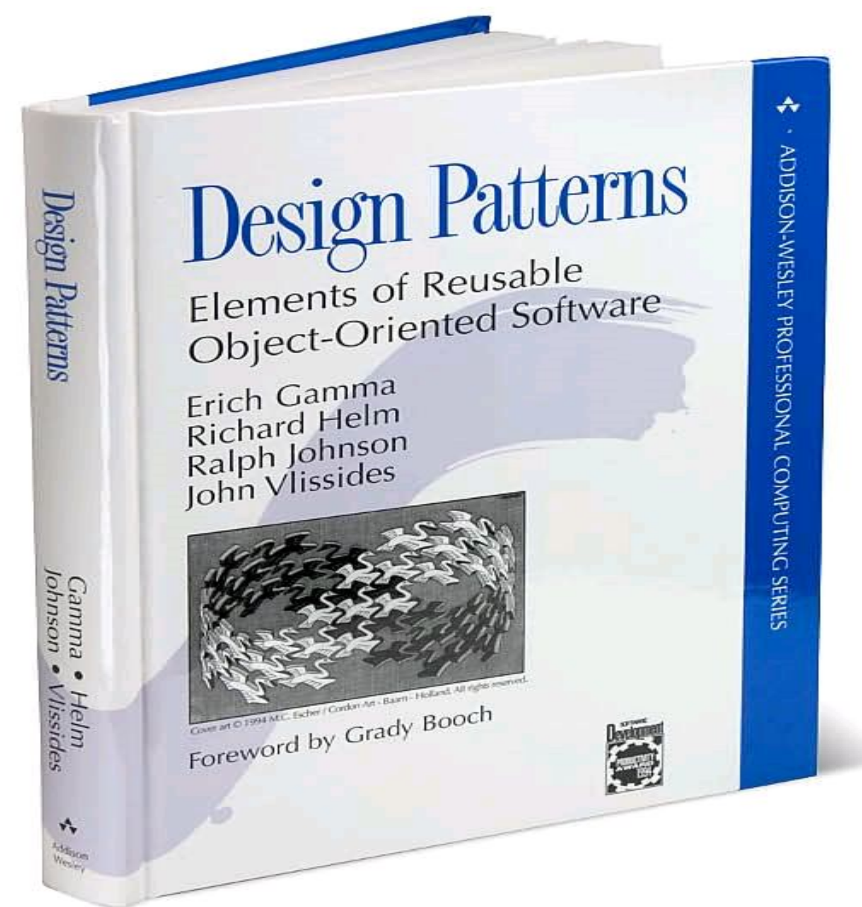
Discussion with design patterns

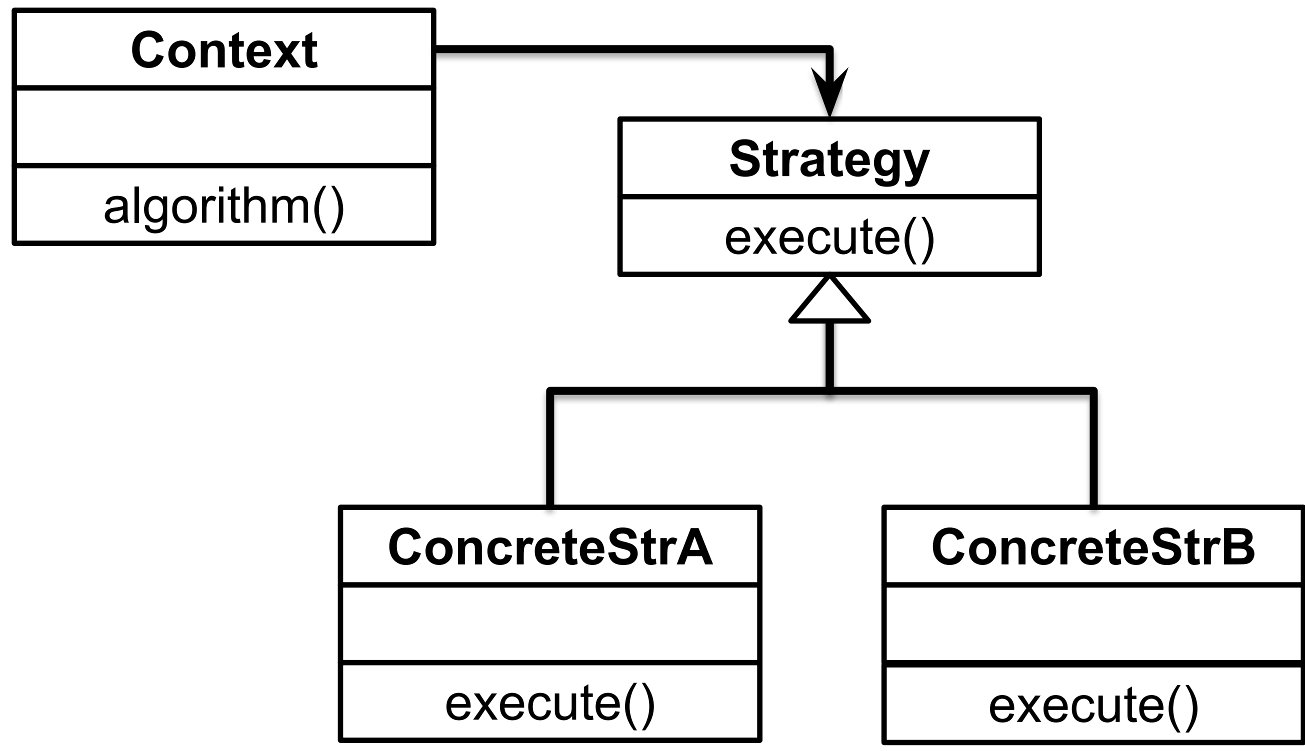
- Carpentry:
  - "Is a dovetail joint or a miter joint better here?"
- Software Engineering:
  - "Is a strategy pattern or a template method better here?"

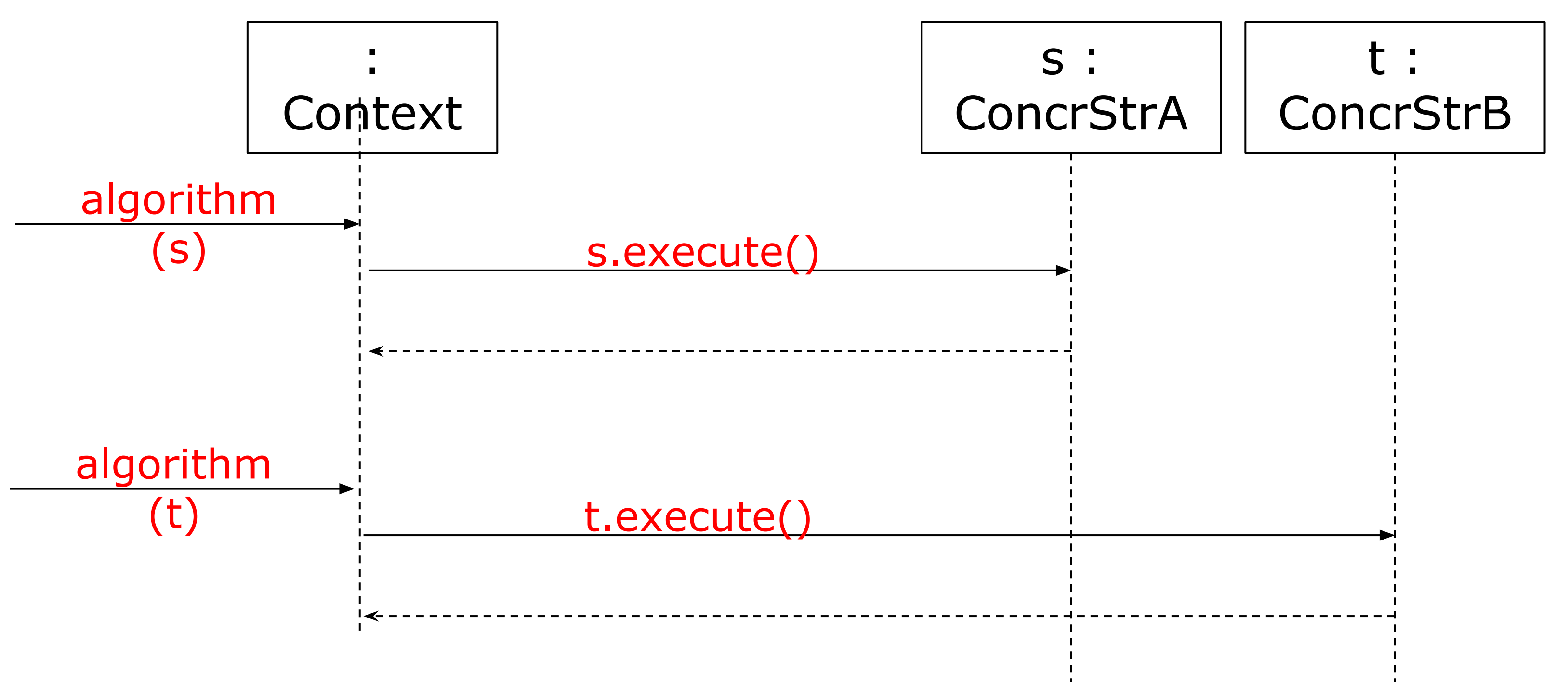




History: *Design Patterns* (1994)




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



Strategy can be provided in method call or in any other way to context

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.

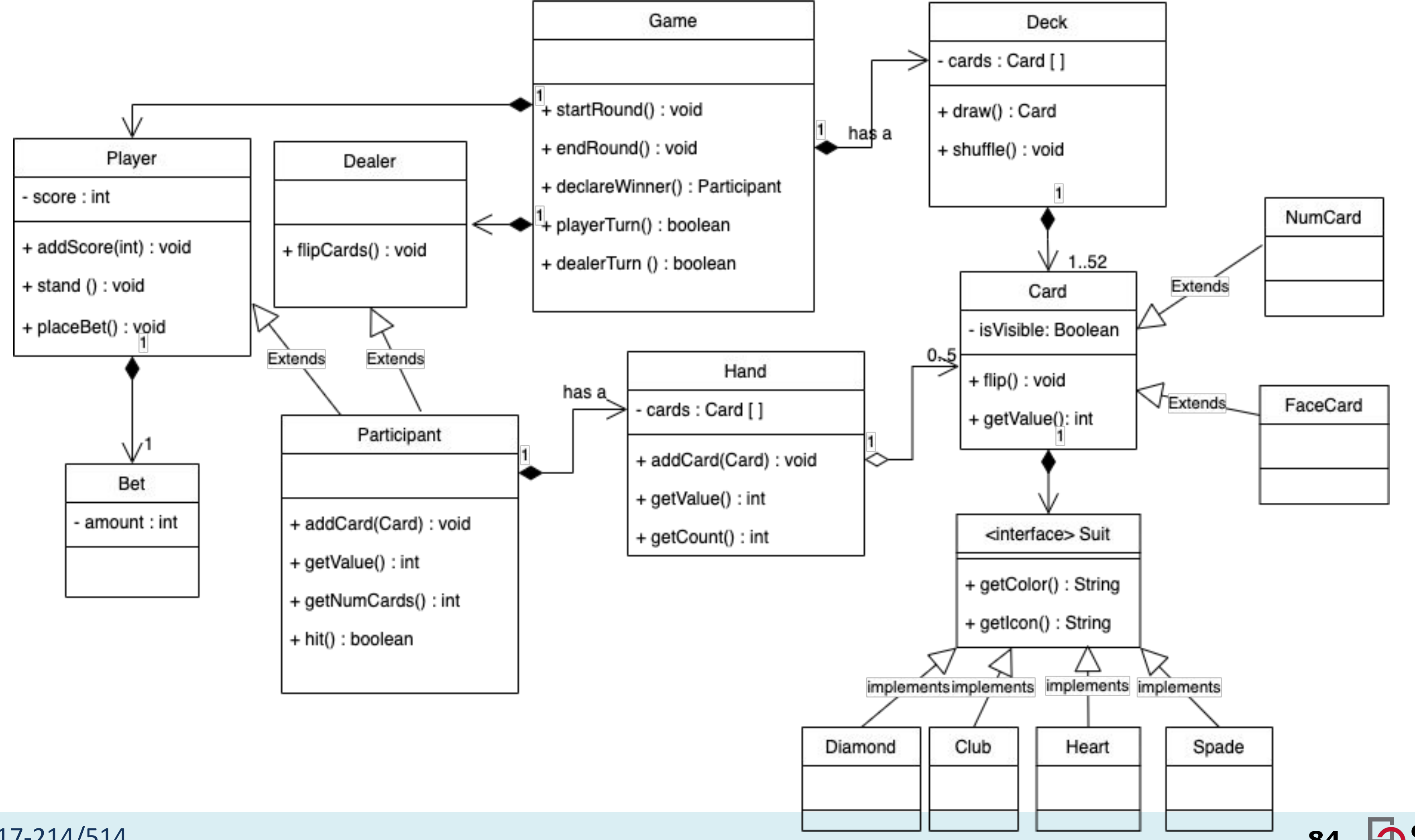
Design Patterns and Programming Languages

Design patterns address general design challenges

Some patterns address problems with built-in solutions

Example: Strategy pattern vs higher-order functions

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


Game
+ startRound() : void
+ endRound() : void
+ declareWinner() : Participant
+ playerTurn() : boolean
+ dealerTurn () : boolean
Deck
- cards : Card [ ]
+ draw() : Card
+ shuffle() : void
has a
Player
- score : int
+ addScore(int) : void
+ stand () : void
+ placeBet() : void
Dealer
+ flipCards() : void
Extends
Extends
Bet
- amount : int
Participant
+ addCard(Card) : void
+ getValue() : int
+ getNumCards() : int
+ hit() : boolean
has a
Hand
- cards : Card [ ]
+ addCard(Card) : void
+ getValue() : int
+ getCount() : int
1..52
Card
- isVisible: Boolean
+ flip() : void
+ getValue(): int
0..5
NumCard
Extends
FaceCard
Extends
<interface> Suit
+ getColor() : String
+ getIcon() : String
implements
implements
implements
implements
Diamond
Club
Heart
Spade

Module pattern: Hide internals in closure

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

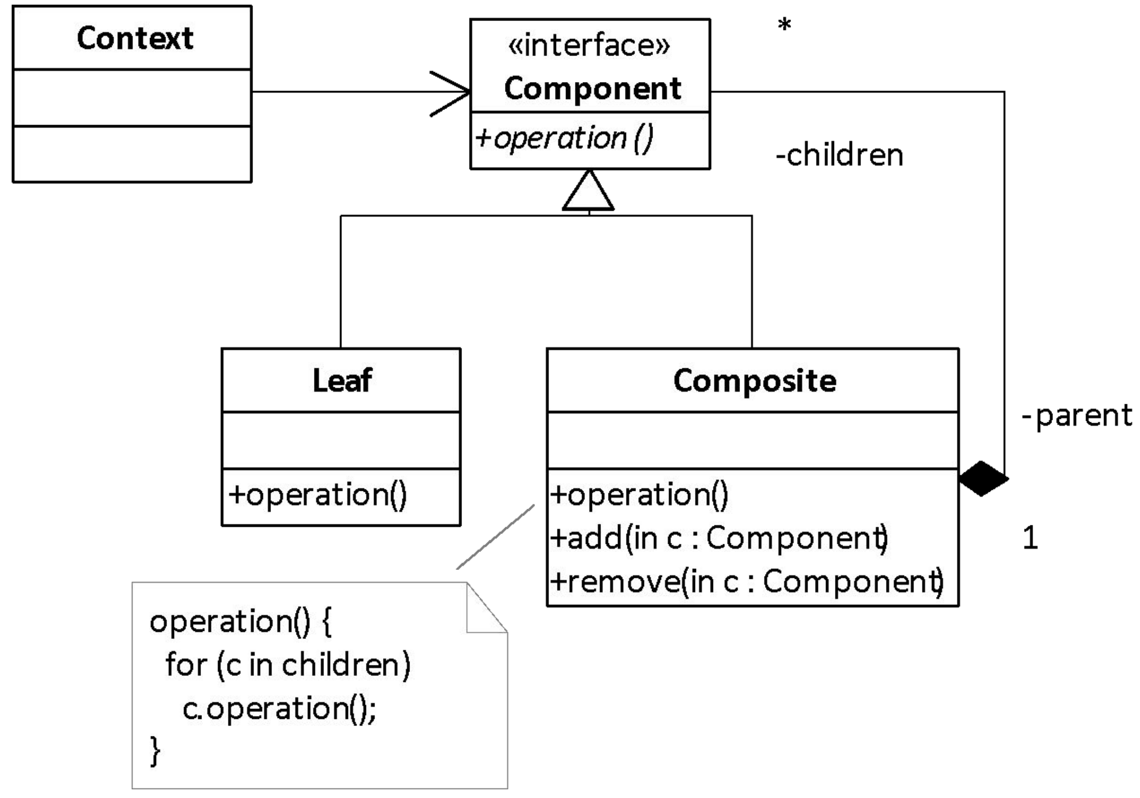
Function provides local scope, internals not accessible

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

Bogdan Vasilescu **Jonathan Aldrich**

Where we are

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

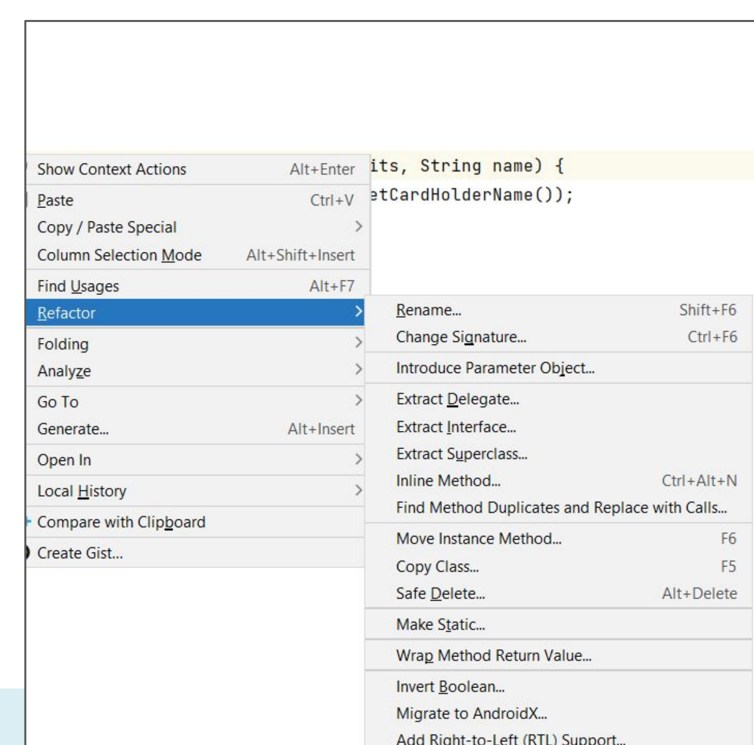
Refactoring

- Any functionality-preserving restructuring
  - That is, the semantics of the program do not change, but the syntax does
  -

```
class Player {
	Board board;
	/* in code somewhere… */ this.getSquare(n);
	Square getSquare(String name) { // named monopoly squares
		for (Square s: board.getSquares())
			if (s.getName().equals(name))
				return s;
		return null;
}}
```

Refactoring: IDE support

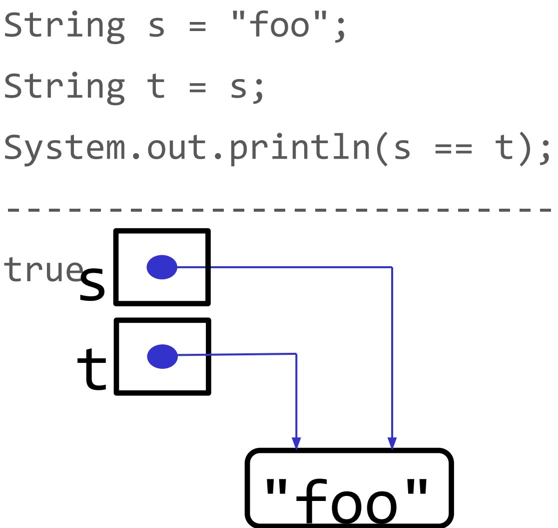
- Rename class, method, variable to something not in-scope
- Extract method/inline method
- Extract interface
- Move method (up, down, laterally)
- Replace duplicates



True or false?

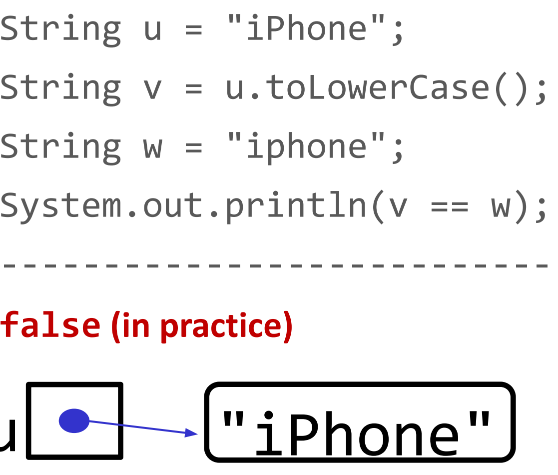
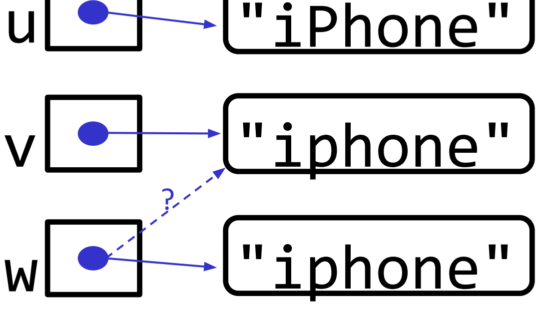
```
int i = 5;
int j = 5;
System.out.println(i == j);
-------------------------------
true
```

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



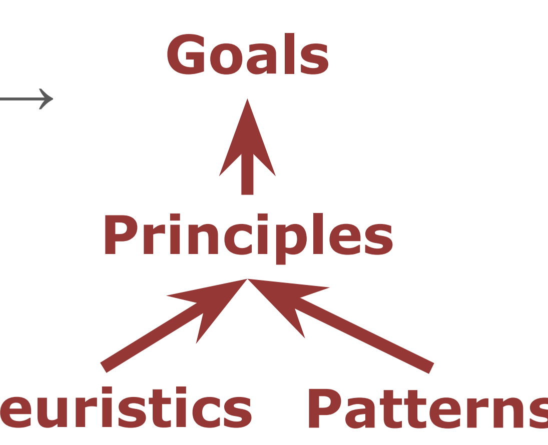

```
String u = "iPhone";
String v = u.toLowerCase();
String w = "iphone";
System.out.println(v == w);
-------------------------------
```

false (in practice)

Anti-patterns

- Kind of like the evil twins of design patterns
- Similar to the design hierarchy on the right, →
  we want to think of both:
  - The design principles they run against
  - The low-level “heuristics” to detect them in code
    - Including many “code smells”
- As before, a pattern language helps
  - Many of these can be (re)paired with a correct pattern



isr institute for SOFTWARE RESEARCH

Liquid APIs

Each method changes state,
then returns `this`

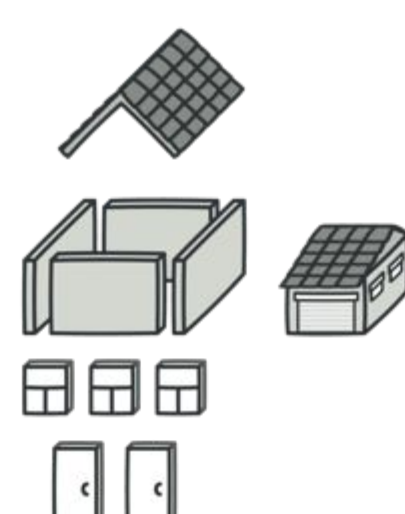
(Immutable version: Return modified copy)

```
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, ...)
    }
}
```

Under the Hood: Builder Pattern

When creating many variations of a complex object:

- Assign assembling work to a Builder object
  - When cascading, the builder returns itself, modified on every update
  - Offers a method that generates the resulting object
- Direct clients to *only* use the Builder
  - E.g., hide the constructor



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

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;
}
```

HW 4&5: Santorini with God Cards and GUI

Principles of Software Construction: Objects, Design, and Concurrency

Introduction to GUIs

Jonathan Aldrich Bogdan Vasilescu

Where we are

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

Interaction with CLI

```
Terminal
File  Edit  View  Search  Terminal  Help
scripts/kconfig/conf arch/x86/Kconfig
*
* Linux Kernel Configuration
*
*
* General setup
*
Prompt for developm
Local version - app
Automatically appen
o) [N/y/?] y
Kernel compression
> 1. Gzip (KERNEL_G
  2. Bzip2 (KERNEL_
  3. LZMA (KERNEL_L
  4. LZO (KERNEL_LZ
choice[1-4?]: 3
Support for paging
System V IPC (SYSVI
POSIX Message Queues
BSD Process Accounting (BSD_PROCESS_ACCT) [Y/n/?] n
Export task/process statistics through netlink (EXPERIMENTAL) (TASKSTATS) [Y/n/?
] y
Enable per task delay accounting (EXPERIMENTAL) (TASK_DELAY_ACCT) [Y/n/?]
```

```java
Scanner input = new Scanner(System.in);
while (questions.hasNext()) {
   Question q = question.next();
   System.out.println(q.toString());
   String answer = input.nextLine();
   q.respond(answer);
}
```


S3D

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)
}
```

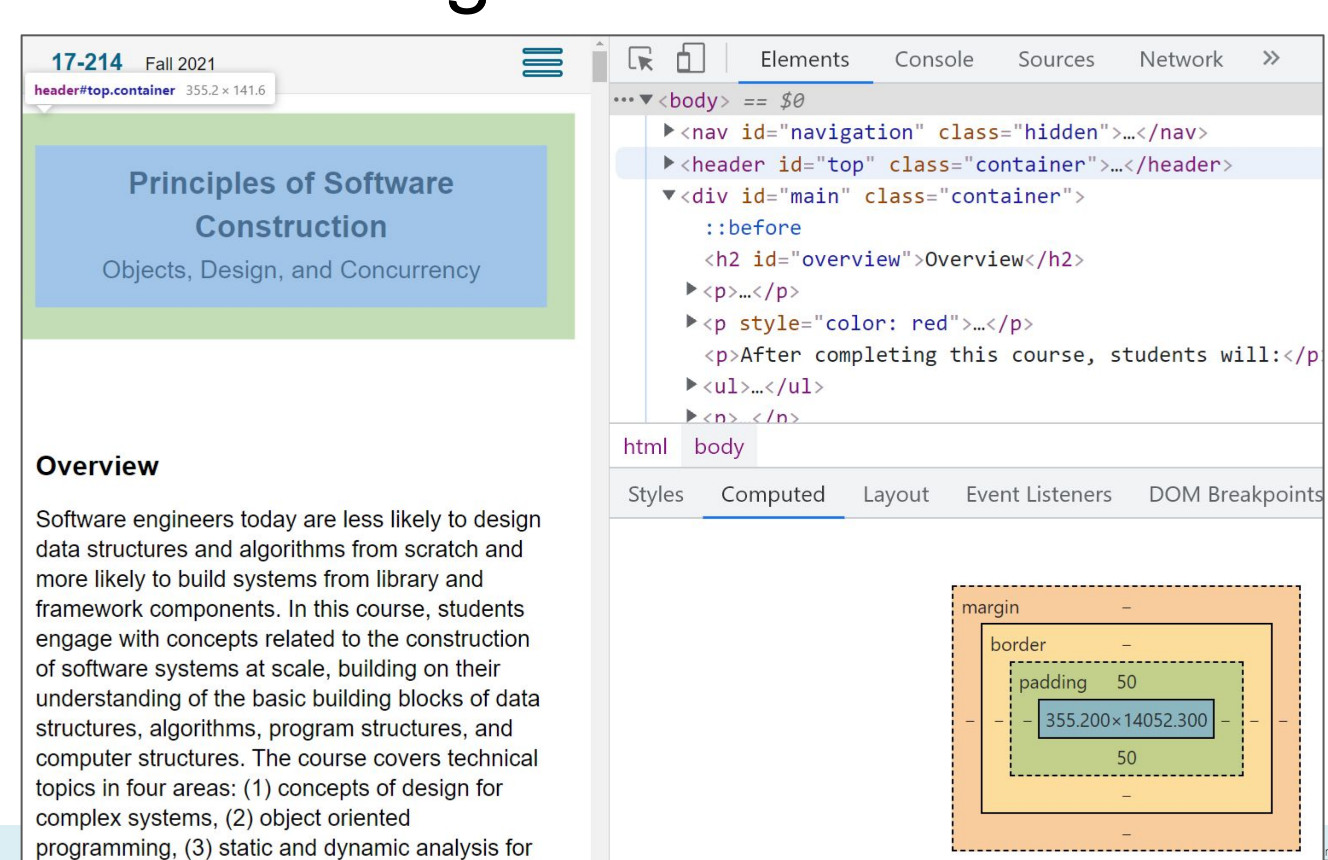
Anatomy of an HTML Page

Nested elements

- Sizing
- Attributes
- Text

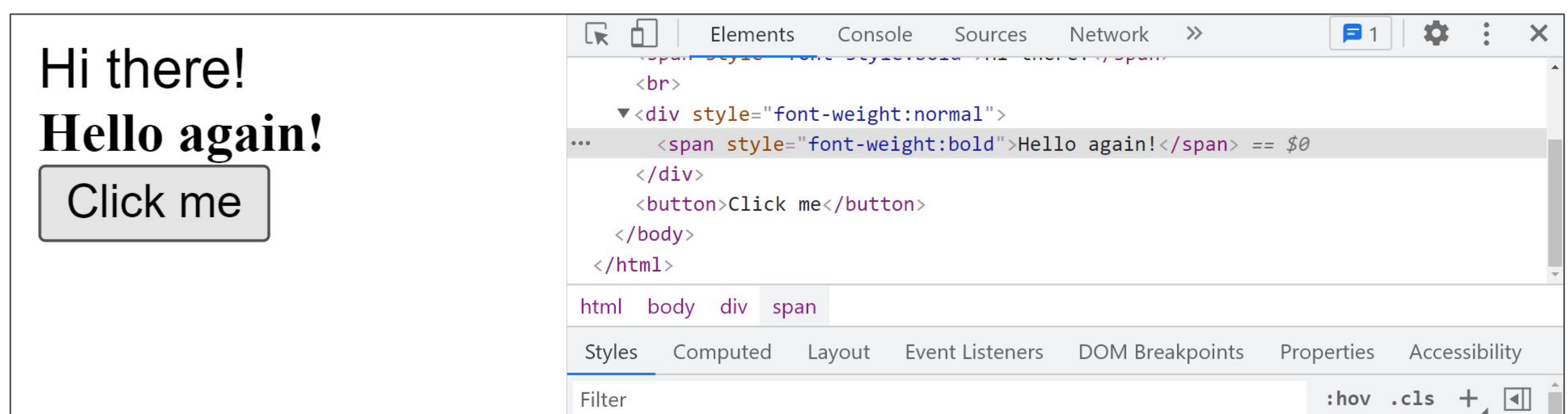
You can write these out directly, or compose and modify them programmatically!

- Or, both! (we'll see in a minute).



Interactivity: A GUI is more than just a document

- How do we make it “work”?
- This is a two-part answer: (1) we can attach scripts to elements, but (2) …how? [Design question!]



isr institute for SOFTWARE RESEARCH

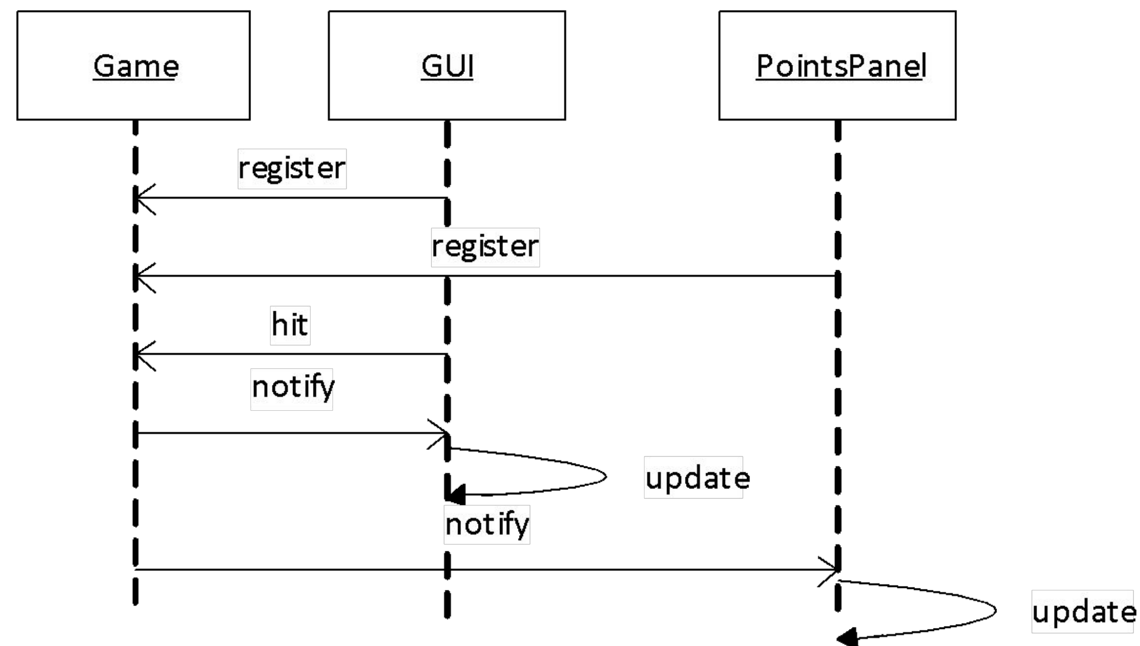
That’s extremely simple, let’s try something *slightly* more complicated.

Consider: TicTacToe

(note that this is NOT the same code you’ll see in recitation next week, but the game itself will look basically the same.)

Decoupling with the Observer pattern

- Let the Game tell *all* interested components about updates



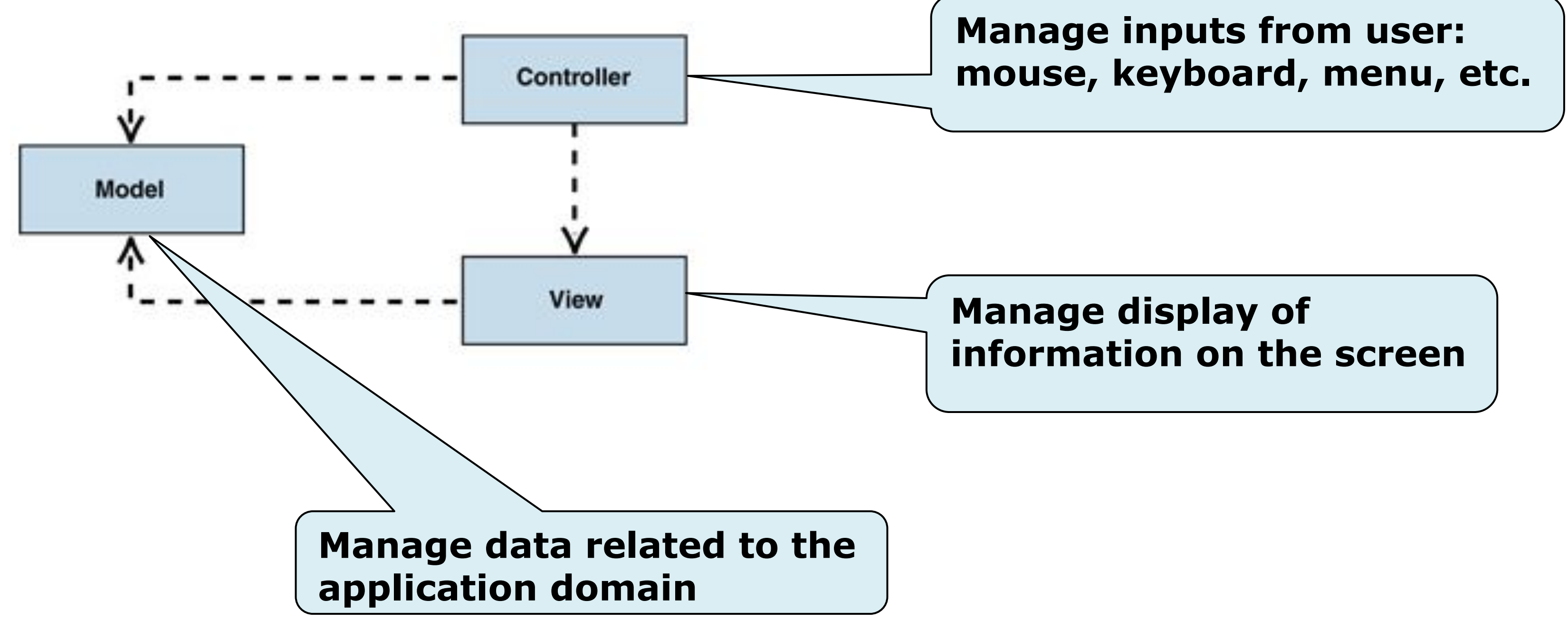
institute for SOFTWARE RESEARCH

Principles of Software Construction: Objects, Design, and Concurrency

(Towards) Building Web-Apps

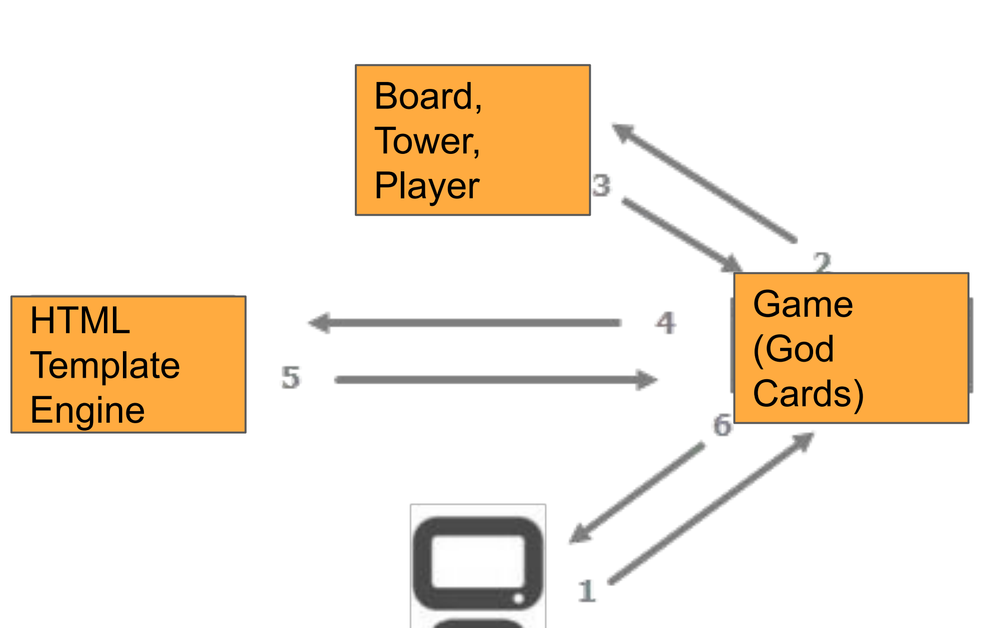
Jonathan Aldrich Bogdan Vasilescu **Matt Davis**

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



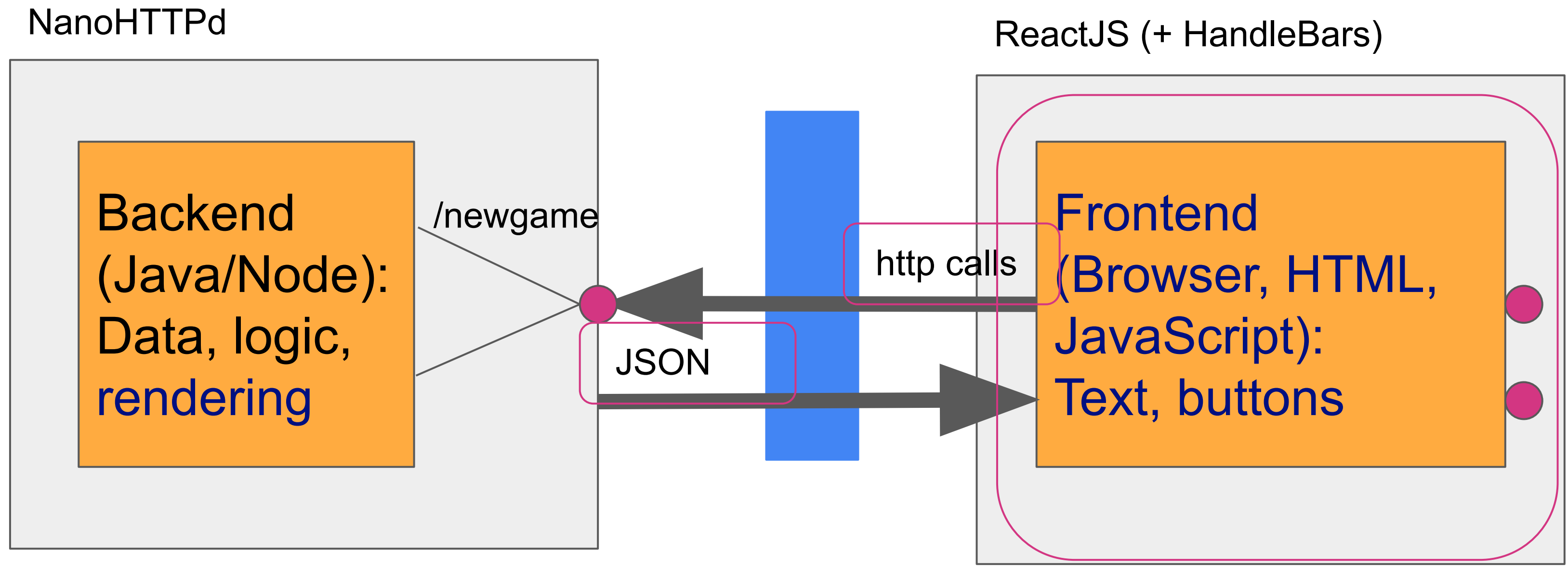
isr institute for SOFTWARE RESEARCH

Model View Controller in Santorini



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

TicTacToe



Connecting React to Some Core

Use observer pattern to let react component observe changes

Encapsulate in *useEffect()* hook

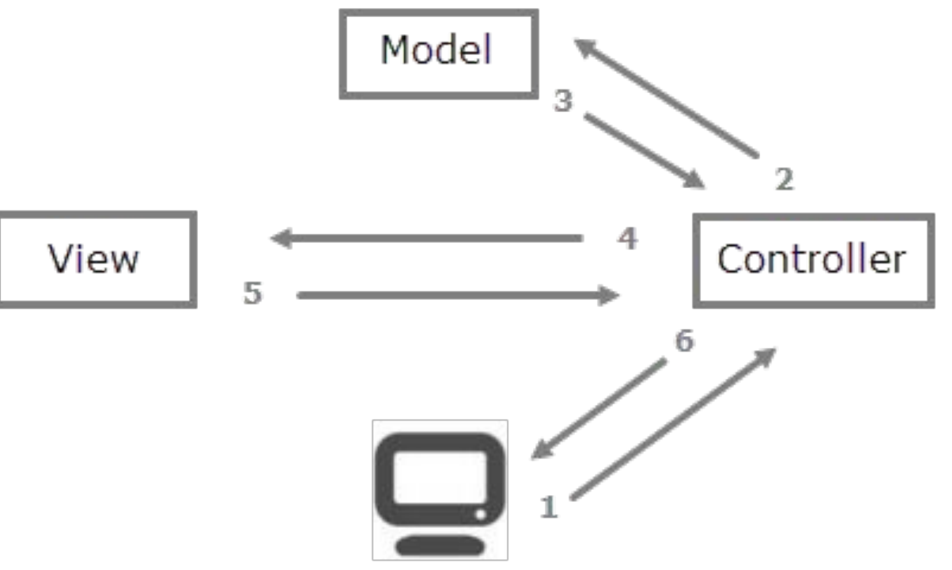
Further discussion: https://reactjs.org/docs/hooks-custom.html

```
function App() {
  const [data, setData] =
              React.useState(null);
  React.useEffect(() => {
    function handleStatChange(e) {
      setData(e.updatedData);
    }
    CoreAPI.subscribe(handleStatChange);
    return () => {
      CoreAPI.unsubscribe(handleStatChange);
    };
  });
  return (
    <div>/* using state in data */</div>
  );
```

How Do We Talk?

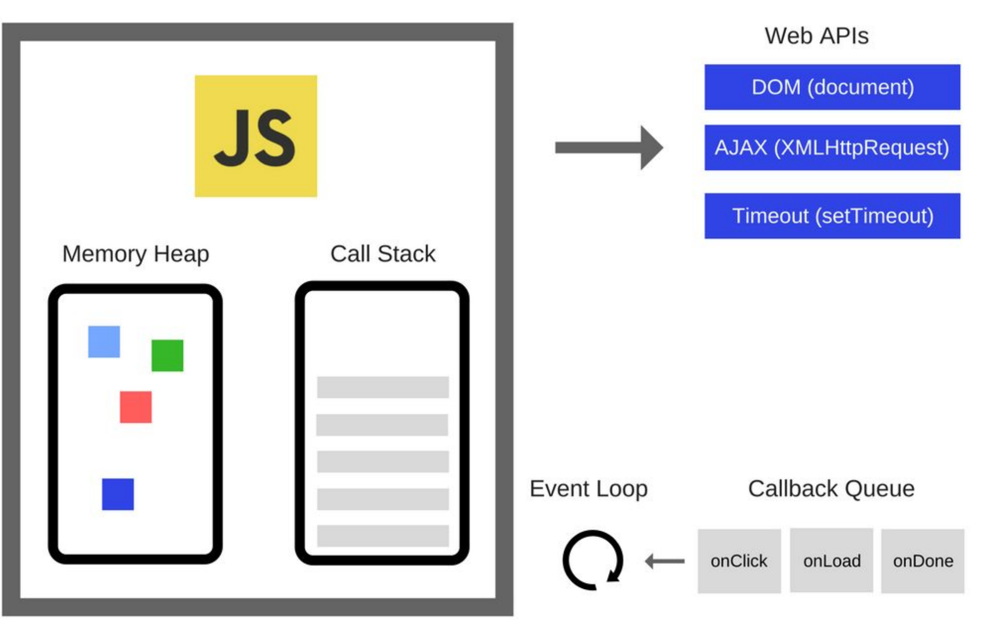
Talking to another computer is *hard*

- Why? We already covered HTTP (GET/POST), right?



isr institute for SOFTWARE RESEARCH

The JavaScript Runtime



Engine plus:

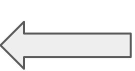
- Web APIs — provided by browsers, like the DOM, AJAX, setTimeout and more.
- Event loop
- Callback queue

isr institute for SOFTWARE RESEARCH

Solution: Callbacks

By far the most common way to express and manage asynchronicity in JavaScript programs.

```
function task(message) {
    // emulate time consuming task
    let n = 10000000000;
    while (n > 0){
        n--;
    }
    console.log(message);
}

console.log('Start script...');
setTimeout(() => {
    task('Download a file.');
}, 1000);
console.log('Done!');
```



```
Start script...
Done!
Download a file.
```


isr institute for SOFTWARE RESEARCH

“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)
})
```

Principles of Software Construction: Objects, Design, and Concurrency

Asynchrony and Concurrency

Jonathan Aldrich **Bogdan Vasilescu**

Basic concurrency in Java

- An interface representing a task

```
public interface Runnable {
    void run();
}
```

- A class to execute a task in a thread

```
public class Thread {
    public Thread(Runnable task);
    public void start();
    public void join();
    …
}
```

makes sure that thread is terminated before the next instruction is executed by the program

Solving “Callback Hell” with Promises

- You can chain promises.
  - ‘then’ returns a promise (remember cascade?)
- Promises can be resolved in parallel
- No more deep nesting
- Easy to follow control-flow

If asynchronous:

```
let bunPromise = getBuns();
let cookedBeefPromise = getBeef()
    .then(beef => cookBeef(beef));
// Resolve both promises in parallel
Promise.all([bunPromise, cookedBeefPromise])
    .then(([buns, beef]) => putBeefBetweenBuns(buns, beef))
    .then(burger => serve(burger))
```

Next Step: Async/Await

- Async functions return a promise
  - And are allowed to ‘await’ synchronously
  - May wrap concrete values
  - May return rejected promises on exceptions

```
async function copyAsyncAwait(source: string, dest: string) {
    let statPromise = promisify(fs.stat)

    // Stat dest.
    try {
        await statPromise(dest)
    } catch (_) {
        console.log("Destination already exists")
        return
    }
```

Threading Example: Money-grab (2)

```java
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());
}
```

isr institute for SOFTWARE RESEARCH

Deadlock example

Two threads:

A does `transfer(a, b, 10)`   B does `transfer(b, a, 10)`

```
class Account {
  double balance;
  void withdraw(double amount){ balance -= amount; }
  void deposit(double amount){ balance += amount; }
  void transfer(Account from, Account to, double amount){
        synchronized(from) {
            from.withdraw(amount);
            synchronized(to) {
                to.deposit(amount);
            }
        }
  }
}
```

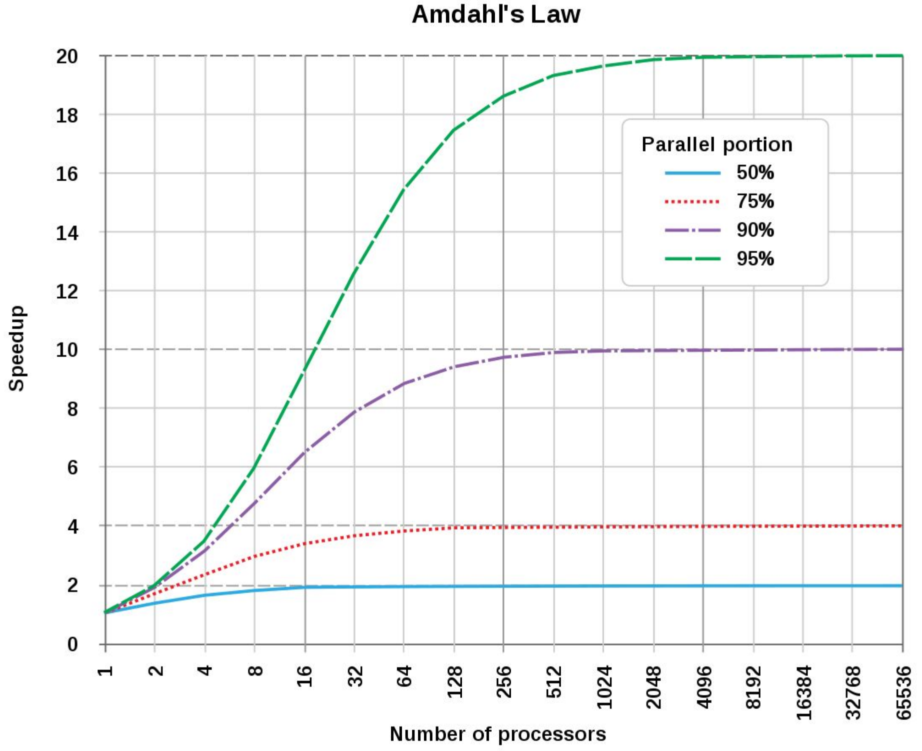
Execution trace:
A: lock a (v)
B: lock b (v)
A: lock b (x)
B: lock a (x)
A: wait
B: wait

Deadlock!

isr institute for SOFTWARE RESEARCH

Amdahl’s law

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



Principles of Software Construction: Objects, Design, and Concurrency

Concurrency: Safety & Immutability

Jonathan Aldrich **Bogdan Vasilescu**

Making a Class Immutable

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

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

    public double getRealPart()      { return re; }
    public double getImaginaryPart() { return im; }

    public double setRealPart(double re)      { this.re = re; }
    public double setImaginaryPart(double im) { this.im = im; }

    …
```

Fixed

```
class Stack {
   readonly #inner: any[]
   constructor (inner: any[]) {
       this.#inner=inner.slice()
   }
   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.slice()
       // Java: return new ArrayList(inner)
   }
}
```

Non atomicity and thread (un)safety

A: `value->9` → `9+1->10` → `value->10`

B: `value->9` → `9+1->10` → `value->10`

```
@NotThreadSafe
public class UnsafeCountingFactorizer implements Servlet {
    private long count = 0;

    public long getCount() { return count; }

    public void service(ServletRequest req, ServletResponse resp) {
        BigInteger i = extractFromRequest(req);
        BigInteger[] factors = factor(i);
        ++count;
        encodeIntoResponse(resp, factors);
    }
}
```

You can do better (?)

`volatile` *is synchronization without mutual exclusion*

```
public class StopThread {
    private static volatile boolean stopRequested;

    public static void main(String[] args) throws Exception {
        Thread backgroundThread = new Thread(() -> {
            while (!stopRequested)
                /* Do something */ ;
        });
        backgroundThread.start();

        TimeUnit.SECONDS.sleep(1);
        stopRequested = true;
    }
}
```

forces all accesses (read or write) to the volatile variable to occur in main memory, effectively keeping the volatile variable out of CPU caches.

https://stackoverflow.com/questions/3519664/difference-between-volatile-and-synchronized-in-java

Monitor Example

```
class SimpleBoundedCounter {
  protected long count = MIN;
  public synchronized long count() { return count; }
  public synchronized void inc() throws InterruptedException {
     awaitUnderMax(); setCount(count + 1);
  }
  public synchronized void dec() throws InterruptedException {
     awaitOverMin(); setCount(count - 1);
  }
  protected void setCount(long newValue) { // PRE: lock held
     count = newValue;
     notifyAll(); // wake up any thread depending on new value
  }
  protected void awaitUnderMax() throws InterruptedException {
     while (count == MAX) wait();
  }
  protected void awaitOverMin() throws InterruptedException {
     while (count == MIN) wait();
  }
}
```

Principles of Software Construction: Objects, Design, and Concurrency

Distributed Systems – Events Everywhere!

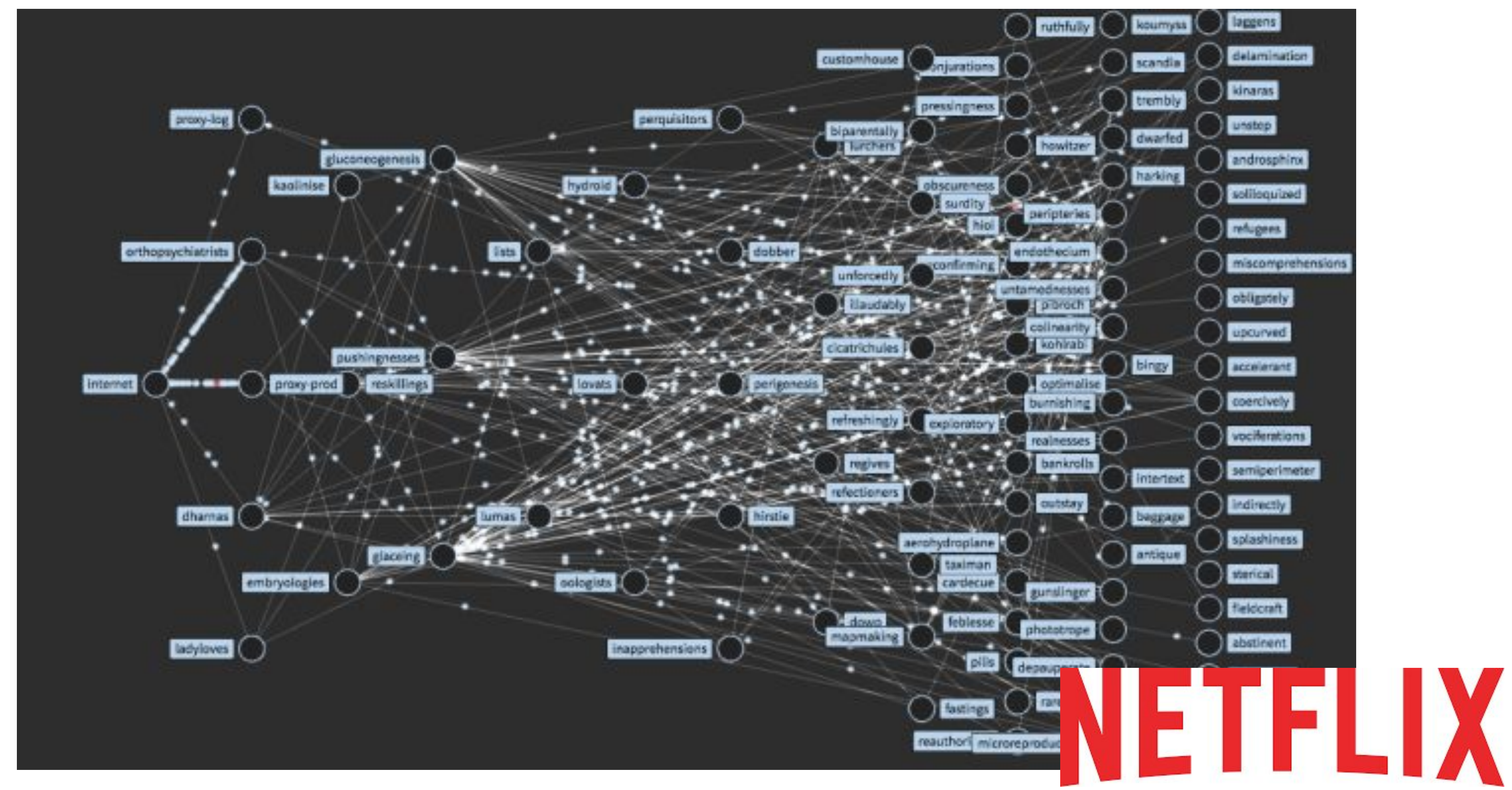
Bogdan Vasilescu **Jonathan Aldrich**

Database Server

Android Phone

Credit card server

isr institute for SOFTWARE RESEARCH


proxy-log
gluconeogenesis
kaolinise
orthopsychiatrists
internet
proxy-prod
reskillings
pushingnesses
dharmas
lumas
glaceing
embryologies
ladyloves
lists
hydroid
lovats
oologists
perquisitors
dobber
perigenesis
hirstie
inapprehensions
customhouse
biparentally
lurchers
unforcedly
cicatrichules
refreshingly
regives
refectioners
pressingness
obscureness
surdity
hioi
confirming
exploratory
aerohydroplane
taximan
cardecue
feblesse
mapmaking
pilis
fastings
ruthfully
howitzer
peripteries
endothecium
untamednesses
pibroch
colinearity
kohlrabi
optimalise
burnishing
realnesses
bankrolls
outstay
gunslinger
phototrope
koumyss
scandia
trembly
dwarfed
harking
bingy
intertext
baggage
antique
laggens
delamination
kinaras
unstop
androsphinx
soliloquized
refugees
miscomprehensions
obligately
upcurved
accelerant
coercively
vociferations
semiperimeter
indirectly
splashiness
sterical
fieldcraft
abstinent
NETFLIX

Retry!

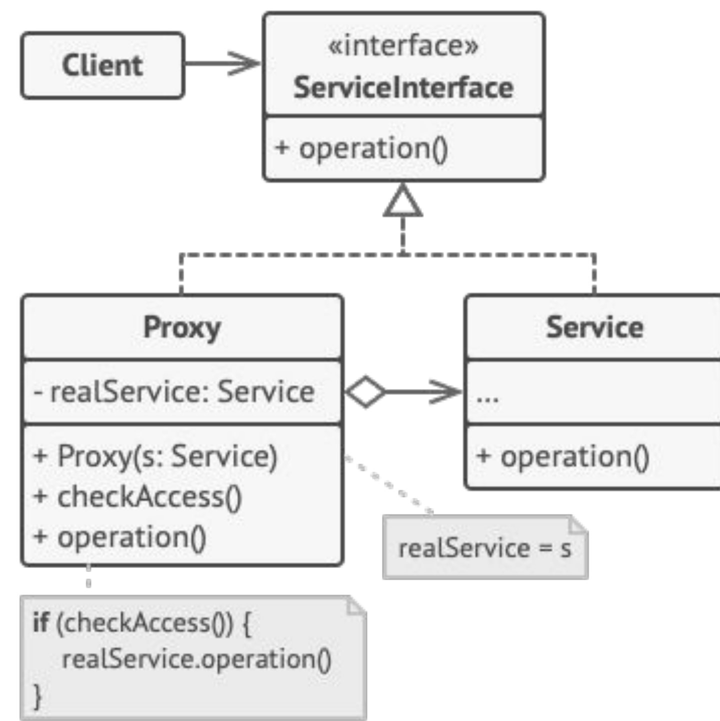
- Still need an exit-strategy
  - Learn HTTP response codes
    - Don't bother retrying on a 403 (go find out why)
  - Use the API response, if any
    - Errors are often documented -- e.g., GitHub will send a “rate limit exceeded” message

```
const delay = retryCount => new Promise(resolve =>
                    setTimeout(resolve, 10 ** retryCount));

const getResource = async (retryCount = 0, lastError = null) => {
  if (retryCount > 5) throw new Error(lastError);
  try {
    return apiCall();
  } catch (e) {
    await delay(retryCount);
    return getResource(retryCount + 1, e);
  }
```

https://www.bayanbennett.com/posts/retrying-and-exponential-backoff-with-promises/

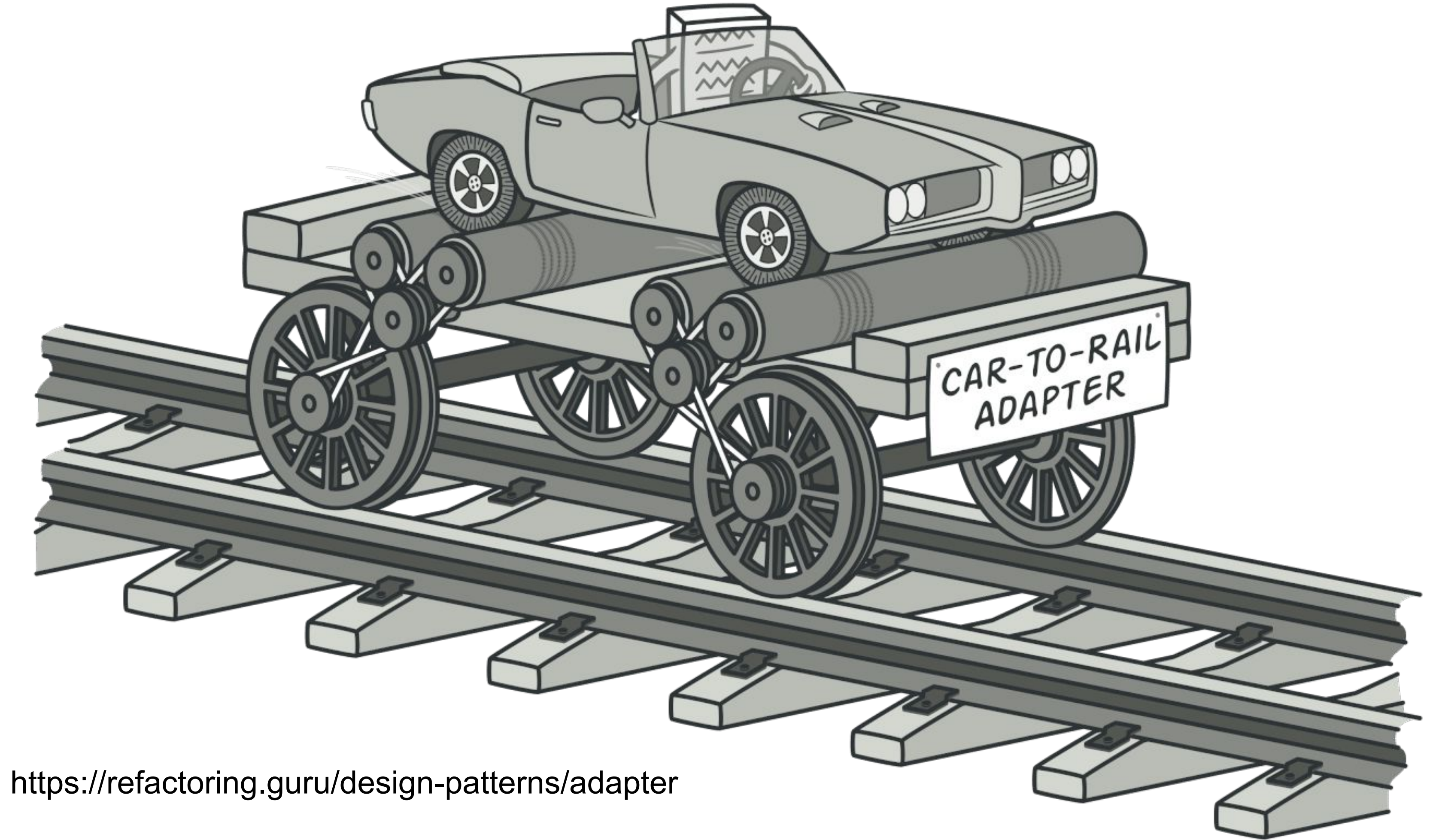
Proxy Design Pattern

- 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

isr institute for SOFTWARE RESEARCH



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

Principles of Software Construction: Objects, Design, and Concurrency

Libraries and Frameworks

(Design for large-scale reuse)

Jonathan Aldrich Bogdan Vasilescu

Where we are

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

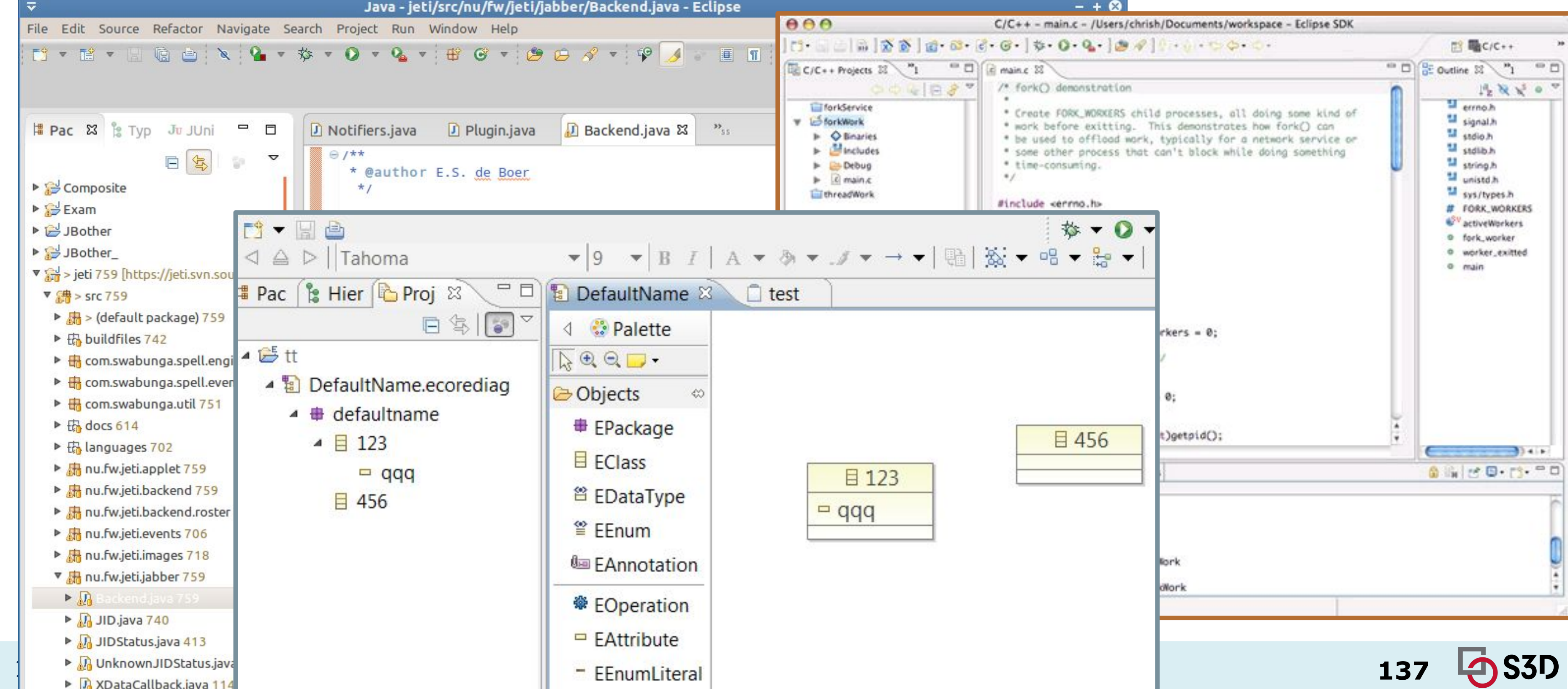
- Small interfaces
- Information hiding
- Low coupling
- High cohesion

Design patterns supporting reuse

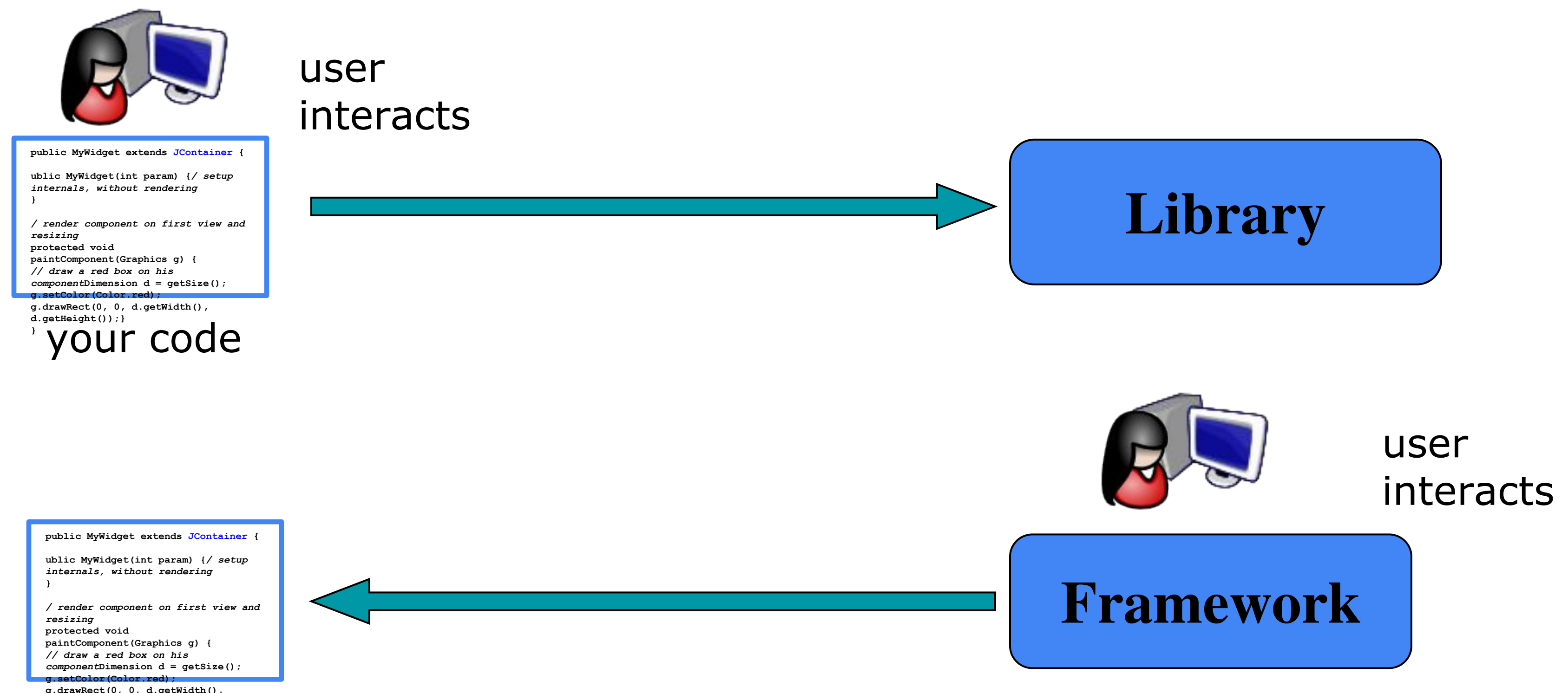
- Template method, decorator, strategy, composite, adapter, …

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

Reuse and variation:
Family of development tools



General distinction: Library vs. framework



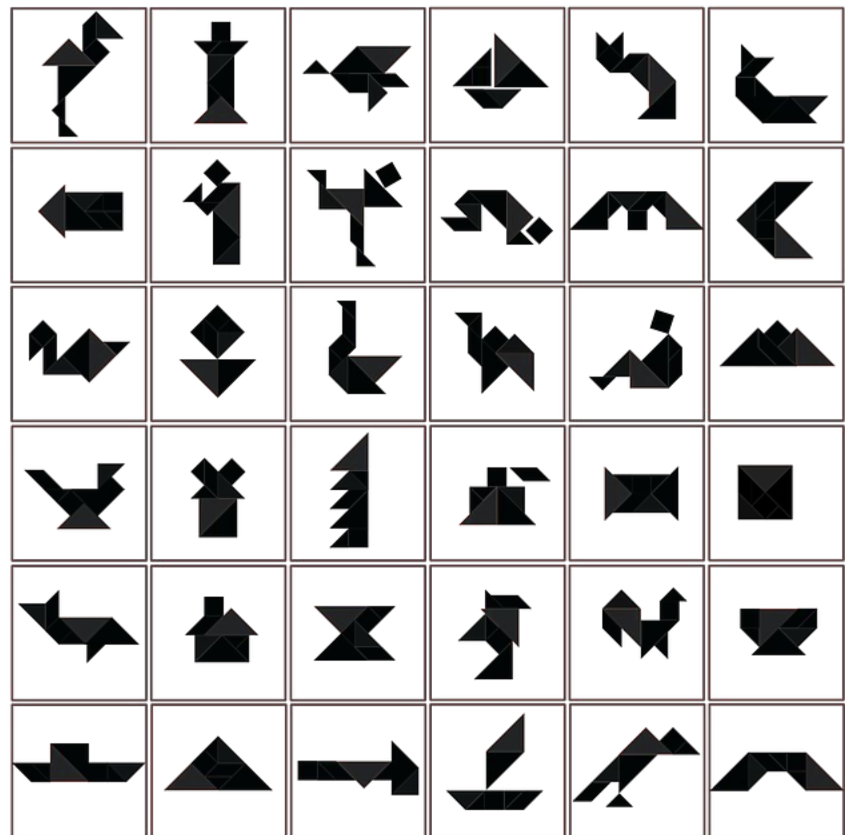
Is this a whitebox or blackbox framework?

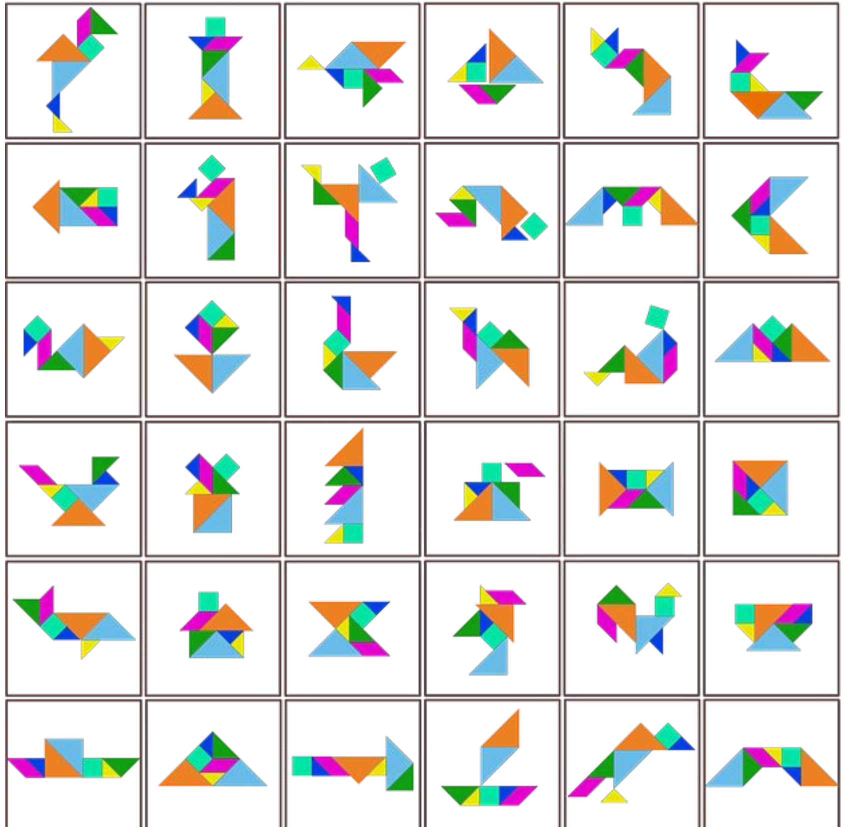
```
public abstract class Application extends JFrame {
  protected String getApplicationTitle() { return ""; }
  protected String getButtonText() { return ""; }
  protected String getInitialText() { return ""; }
```

```
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





S3D

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

```
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(BevelBor
            JButton button = new JButton();
            if (plugin != null)
                button.setText(plugin.getButtonText());
            else
                butto
            contentPane
            textfield =
            if (plugin
                textf
            textfield.s
            this.setCon
```

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

```
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); }}
```

An example plugin loader in Node.js

```
const args = process.argv
if (args.length < 3)
    console.log("Plugin name not specified");
else {
    const plugin = require("plugins/"+args[2]+".js")()
    startApplication(plugin)
}
```

Principles of Software Construction

API Design

Jonathan Aldrich **Bogdan Vasilescu**

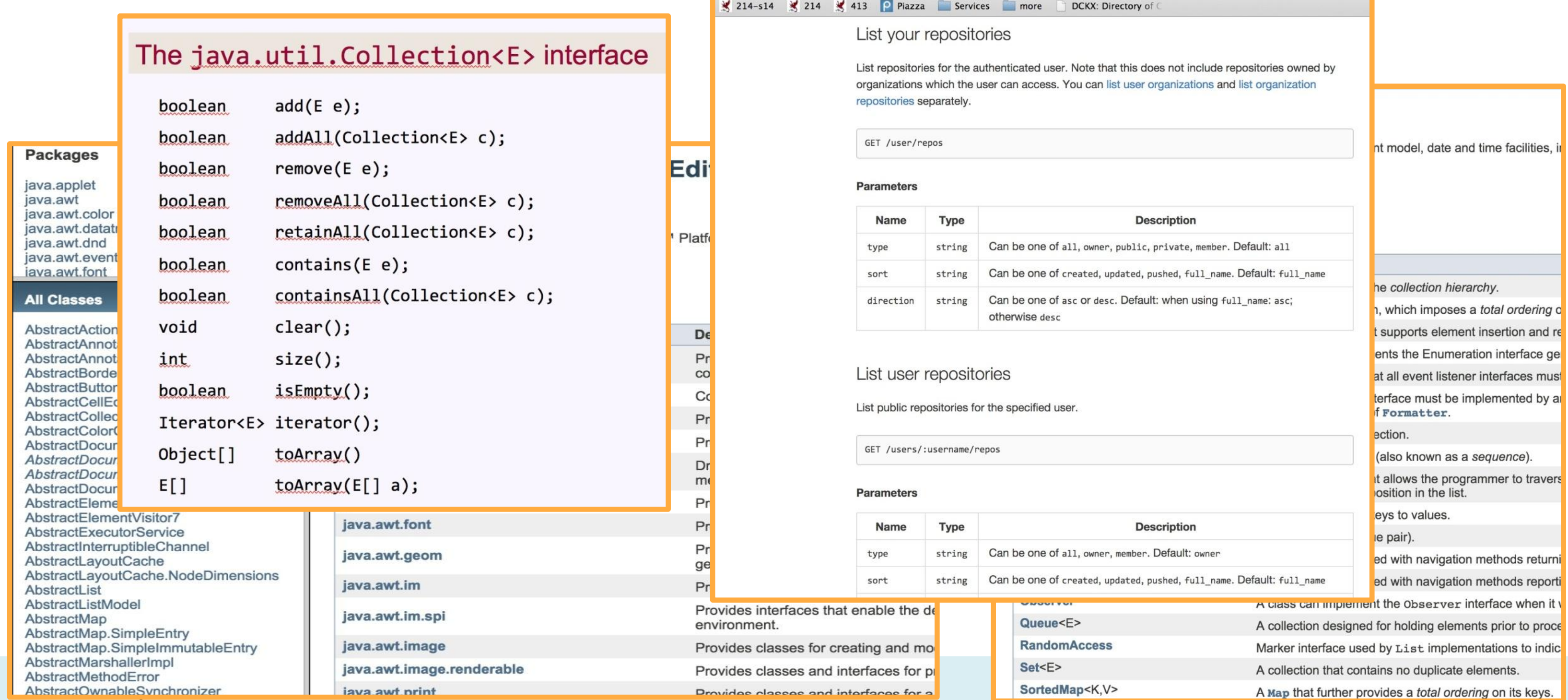
(Many slides originally from Josh Bloch, some from Christian Kästner)

Where we are

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

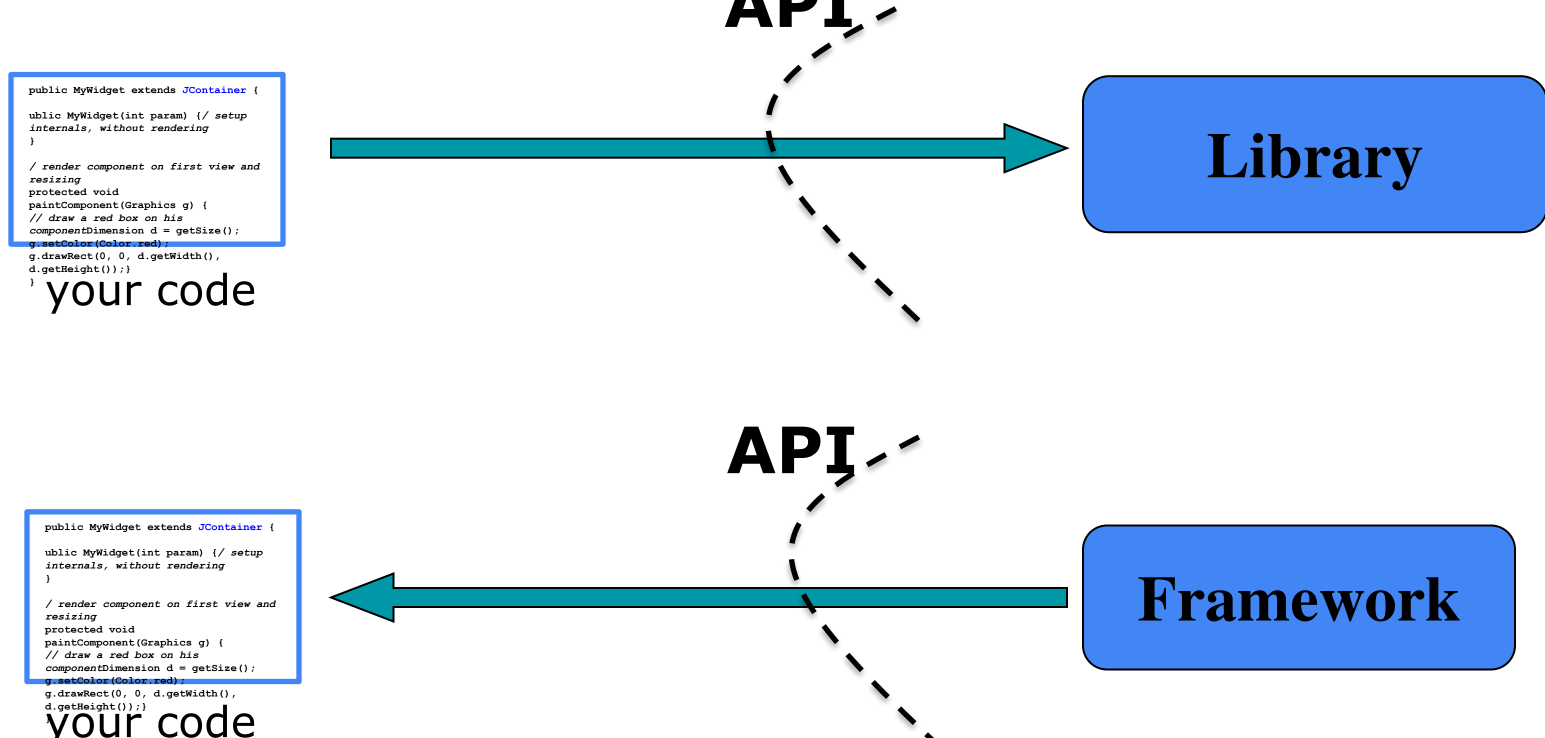
API: Application Programming Interface

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



S3D

Libraries and frameworks (and protocols!) define APIs



An API design process: plan with use cases

- Similar to our framework discussion!
- Define the scope of the API
  - Collect use-case stories, define requirements
  - Be skeptical: Distinguish true requirements from so-called solutions, "When in doubt, leave it out."
  - Be explicit about *non-goals*
- Draft a specification, gather feedback, revise, and repeat. Keep it simple, short!
- Code early, code often: Write *client code* before you implement the API

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
}
```

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/

LATEST: 10.17 UPDATE

CHANGES IN VERSION 10.17:
THE CPU NO LONGER OVERHEATS WHEN YOU HOLD DOWN SPACEBAR.

COMMENTS:

LONGTIMEUSER4 WRITES:
THIS UPDATE BROKE MY WORKFLOW! MY CONTROL KEY IS HARD TO REACH, SO I HOLD SPACEBAR INSTEAD, AND I CONFIGURED EMACS TO INTERPRET A RAPID TEMPERATURE RISE AS "CONTROL".

ADMIN WRITES:
THAT'S HORRIFYING.

LONGTIMEUSER4 WRITES:
LOOK, MY SETUP WORKS FOR ME. JUST ADD AN OPTION TO REENABLE SPACEBAR HEATING.

EVERY CHANGE BREAKS SOMEONE'S WORKFLOW.

https://xkcd.com/1172/

Applying Information hiding: Factories

```
public class Rectangle {
        public Rectangle(Point e, Point f) …
}
// …
Point p1 = PointFactory.Construct(…);
// new PolarPoint(…); inside
Point p2 = PointFactory.Construct(…);
// new PolarPoint(…); inside
Rectangle r = new Rectangle(p1, p2);
```

Don't let your output become your de facto API

- Document the fact that output formats may evolve in the future
- Provide programmatic access to all data available in string form

```
public class Throwable {
    public void printStackTrace(PrintStream s);
}
```

```
org.omg.CORBA.MARSHAL: com.ibm.ws.pmi.server.DataDescriptor; IllegalAccessException  minor code: 4942F23E   comp
        at com.ibm.rmi.io.ValueHandlerImpl.readValue(ValueHandlerImpl.java:199)
        at com.ibm.rmi.iiop.CDRInputStream.read_value(CDRInputStream.java:1429)
        at com.ibm.rmi.io.ValueHandlerImpl.read_Array(ValueHandlerImpl.java:625)
        at com.ibm.rmi.io.ValueHandlerImpl.readValueInternal(ValueHandlerImpl.java:273)
        at com.ibm.rmi.io.ValueHandlerImpl.readValue(ValueHandlerImpl.java:189)
        at com.ibm.rmi.iiop.CDRInputStream.read_value(CDRInputStream.java:1429)
        at com.ibm.ejs.sm.beans._EJSRemoteStatelessPmiService_Tie._invoke(_EJSRemoteStatelessPmiService_Tie.ja
        at com.ibm.CORBA.iiop.ExtendedServerDelegate.dispatch(ExtendedServerDelegate.java:515)
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:2377)
        at com.ibm.CORBA.iiop.OrbWorker.run(OrbWorker.java:186)
        at com.ibm.ejs.oa.pool.ThreadPool$PooledWorker.run(ThreadPool.java:104)
        at com.ibm.ws.util.CachedThread.run(ThreadPool.java:137)
```

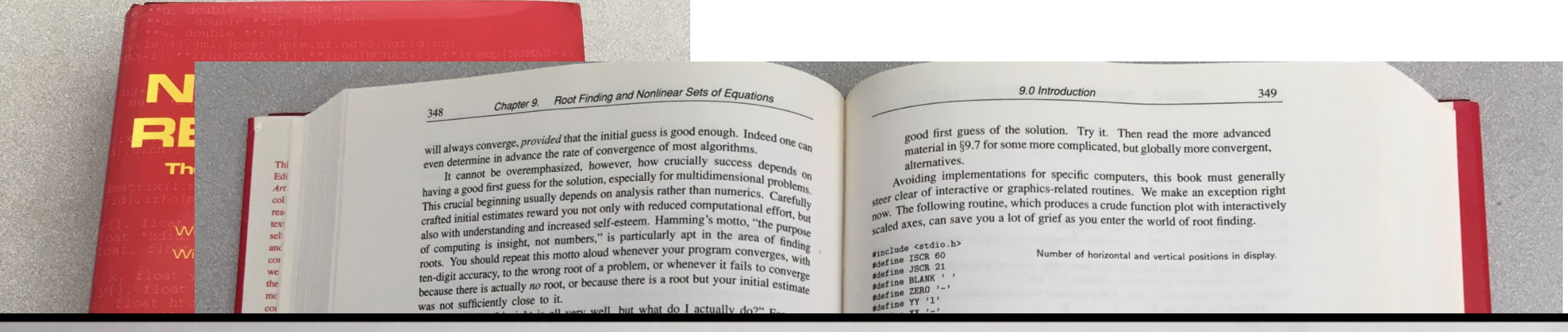
Principle: Minimize conceptual weight

- 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.

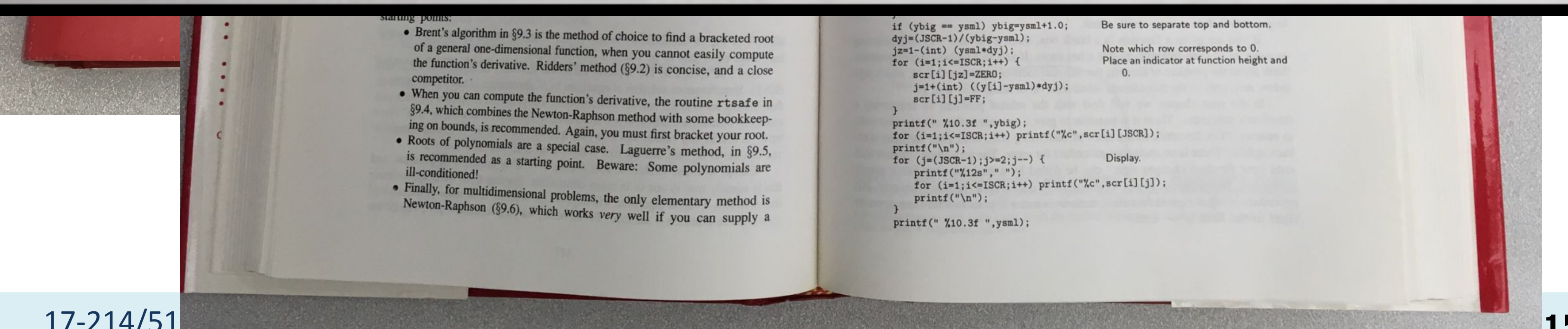
9.0 Introduction 349

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        Number of horizontal and vertical positions in display.
#define JSCR 21
#define BLANK ' '
#define ZERO '-'
#define YY '1'
```

```
int jz,j,i;
float ysml,ybig,x2,x1,x,dyj,dx,y[ISCR+1];
char scr[ISCR+1][JSCR+1];
```



- 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
    scr[i][jz]=ZERO;                        0.
    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);
```

HW6: Data Analytics Framework

Principles of Software Construction: Objects, Design, and Concurrency

Organizing Systems at Scale: Modules, Dependencies, Breaking Changes

Jonathan Aldrich **Bogdan Vasilescu**

REST (or RESTful) API

API of a web service “that conforms to the constraints of the REST architectural style.”

Uniform interface over HTTP requests

Send parameters to URL, server responds with the representation of a resource (JSON, XML common)

Stateless: Each request is self-contained

Language independent, distributed

Packages enough?

```
edu.cmu.cs214.santorini
edu.cmu.cs214.santorini.gui
edu.cmu.cs214.santorini.godcards
edu.cmu.cs214.santorini.godcards.impl
edu.cmu.cs214.santorini.logic
edu.cmu.cs214.santorini.utils
```

Toward Module Systems

Stronger encapsulation sometimes desired

Expose only select public packages (and all public classes therein) to other modules

Dynamic adding and removal of modules desired

OSGi (most prominently used by Eclipse)

- Bundle Java code with Manifest
- Framework handles loading with multiple classloaders

```
Bundle-Name: Hello World
Bundle-SymbolicName: org.wikipedia.helloworld
Bundle-Description: A Hello World bundle
Bundle-ManifestVersion: 2
Bundle-Version: 1.0.0
Bundle-Activator: org.wikipedia.Activator
Export-Package:
org.wikipedia.helloworld;version="1.0.0"
Import-Package:
org.osgi.framework;version="1.3.0"
```

The Module Pattern

Learning

Patterns

By Lydia Hallie and Addy Osmani

```
var myRevealingModule = (function () {
    var privateVar = "Ben Cherry",
        publicVar = "Hey there!";

    function privateFunction() {
        console.log( "Name:" + privateVar );
    }

    function publicSetName( strName ) {
        privateVar = strName;
    }

    function publicGetName() {
        privateFunction();
    }

    // Reveal public pointers to
    // private functions and properties
    return {
        setName: publicSetName,
        greeting: publicVar,
        getName: publicGetName
    };
})();

myRevealingModule.setName( "Paul Kinlan" );
```

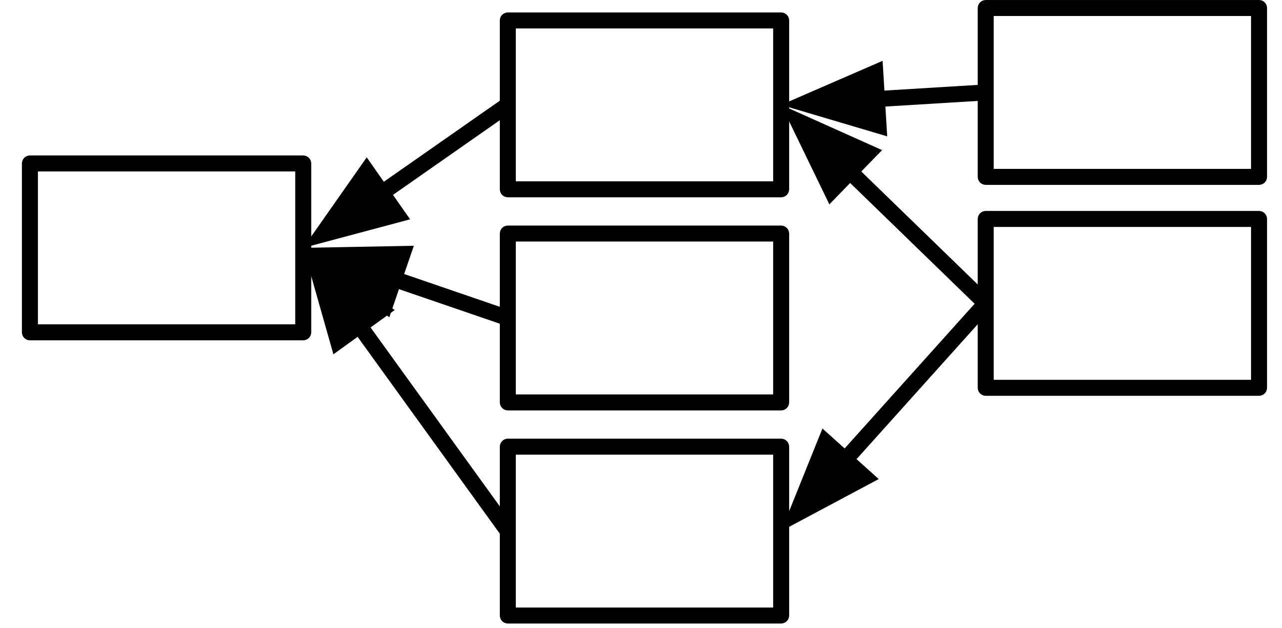
Java Platform Module System

Since Java 9 (2017); built-in alternative to OSGi

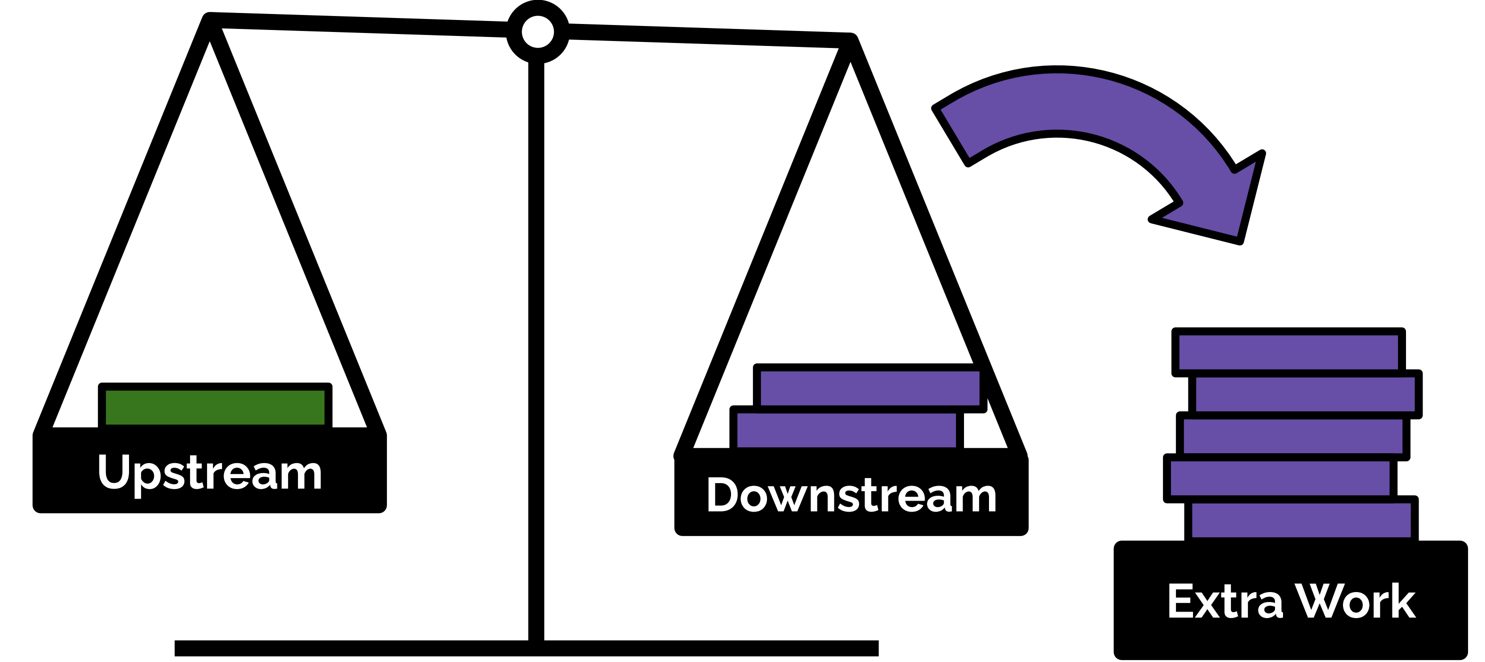
Modularized JDK libraries itself

Several technical differences to OSGi (e.g., visibility vs access protection, handling of diamond problem)

```
module A {
     exports org.example.foo;
     exports org.example.bar;
}
module B {
     require A;
}
```



Software Ecosystem



Avoiding dependencies
Encapsulating from change

How to Break an API?

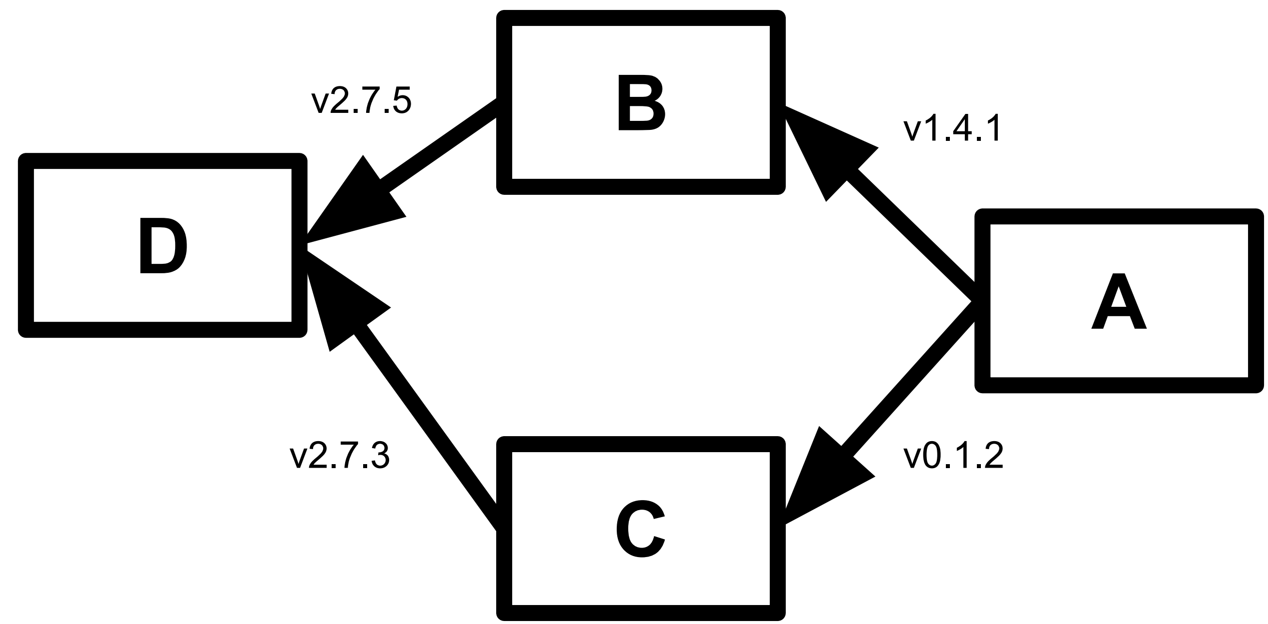
In Eclipse, you don't.

In CRAN, you reach out to affected downstream developers.

In Node.js, you increase the major version number.

Photo Credit: axi11a (cc)

The Diamond Problem



What now?

Principles of Software Construction: Objects, Design, and Concurrency

Designing for Robustness in Large & Distributed Systems

Jonathan Aldrich **Bogdan Vasilescu**

Where we are

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

isr institute for SOFTWARE RESEARCH

What Do We Test?

Android client — Code — Facebook

```
void buttonClicked() {
   render(getFriends());
}
List<Friend> getFriends() {
   Connection c = http.getConnection();
   FacebookAPI api = new FacebookAPI(c);
   return api.getFriends("john");
}
```

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

isr institute for SOFTWARE RESEARCH

Fault injection

Test driver — Code — Mock Facebook

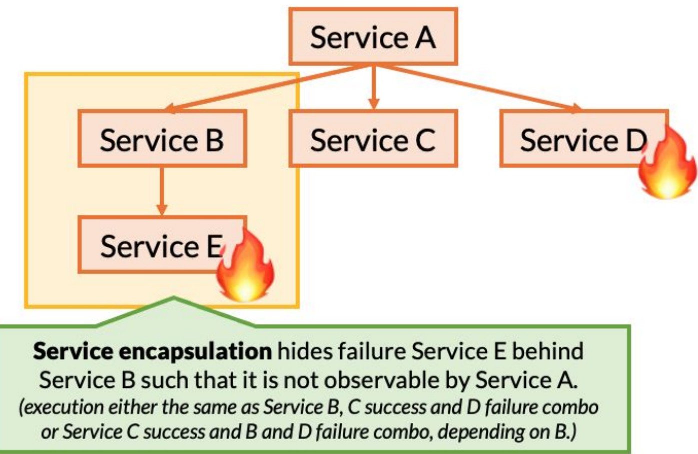
```
class FacebookErrorStub implements FacebookAPI {
  void connect() {}
  int counter = 0;
  List<Node> getFriends(String name) {
    counter++;
    if (counter % 3 == 0)
      throw new SocketException("Network is unreachable");
    else if (name.equals("john")) {
      return List.of(...);
    } //
```

Chaos Engineering

Experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production

Considerations in HW6

- What should the framework do when a plugin fails?
    - Recall this figure? Think of framework as Service A, plugin as B, and the API that B depends on ass as E



isr institute for SOFTWARE RESEARCH

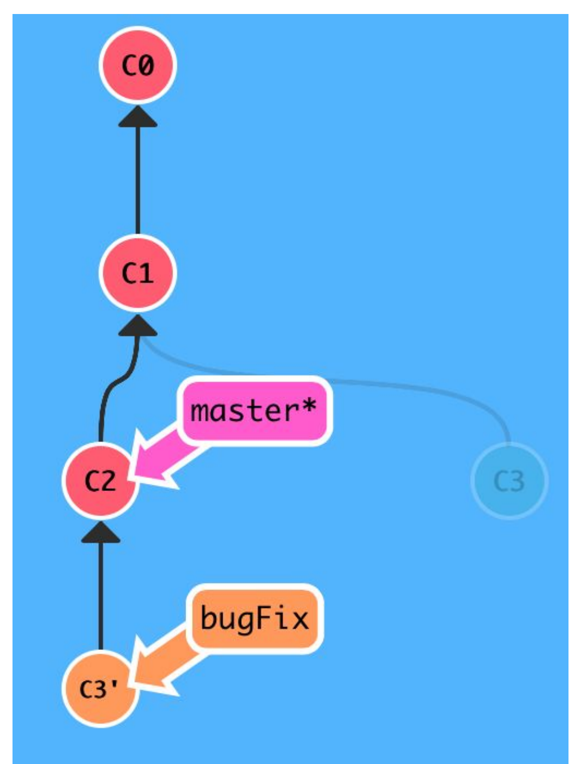
Principles of Software Construction: Objects, Design, and Concurrency

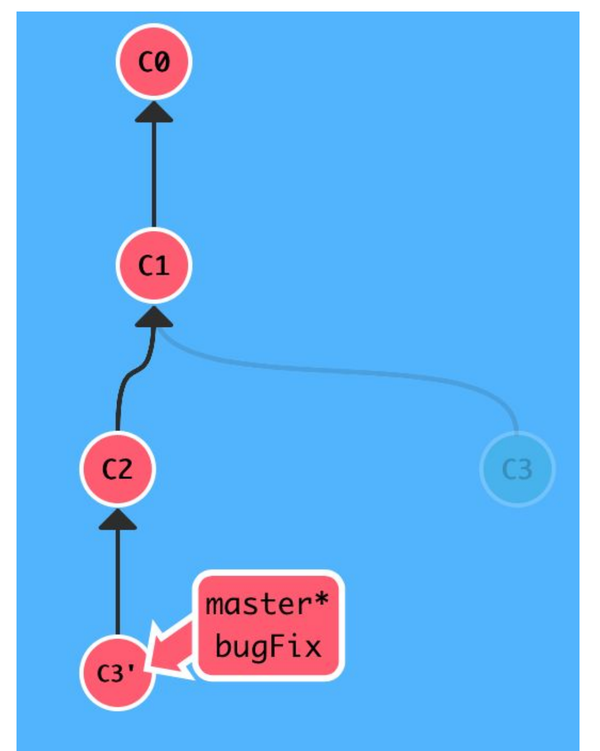
Git Workflows in Practice

Jonathan Aldrich **Bogdan Vasilescu**

But master hasn't been updated, so:

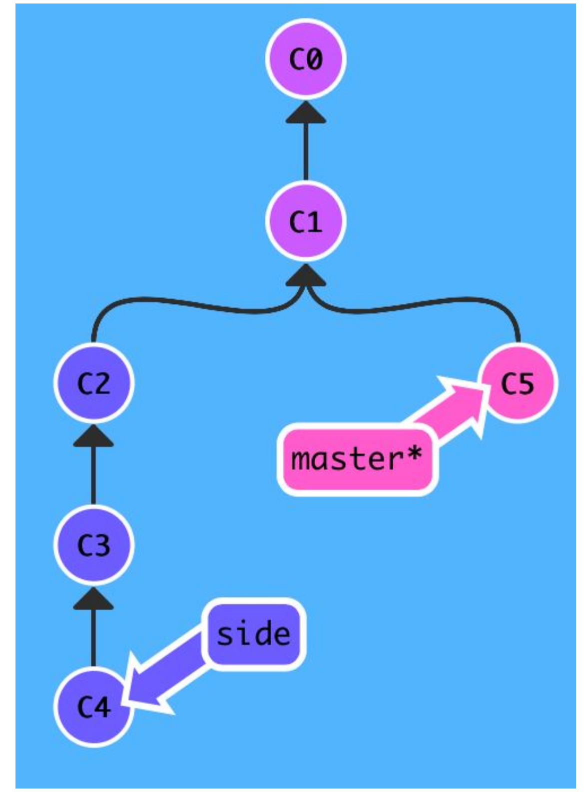
```
git checkout master; git rebase bugFix
```

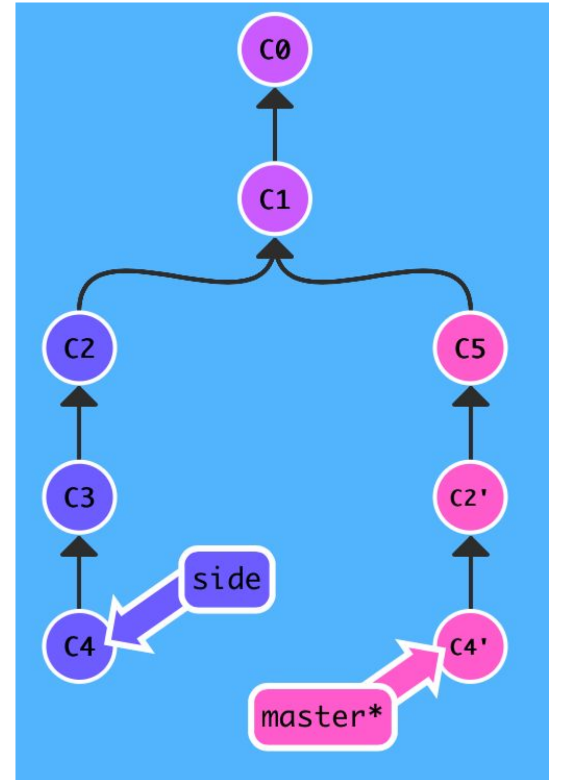




Copy a series of commits below current location

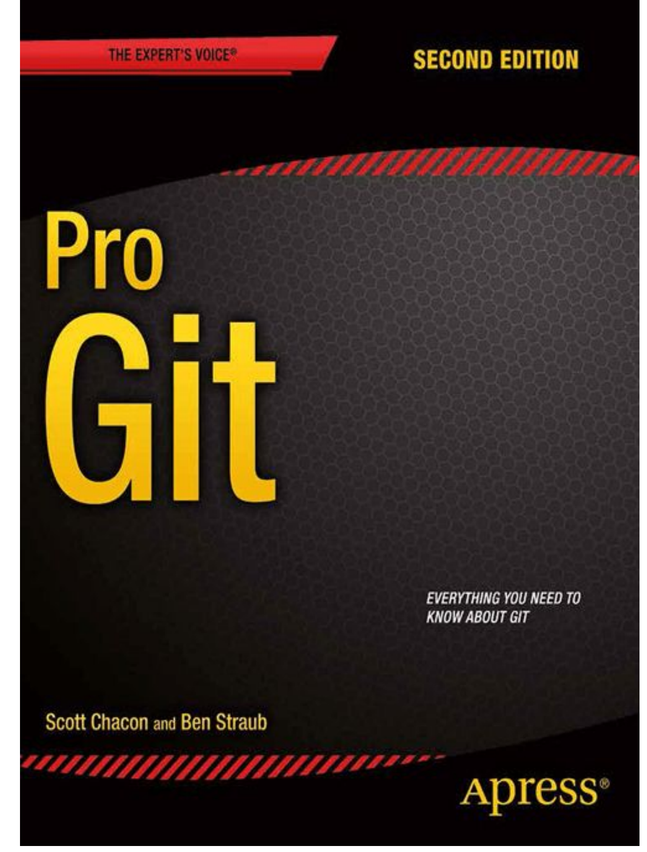
3) git cherry-pick C2 C4





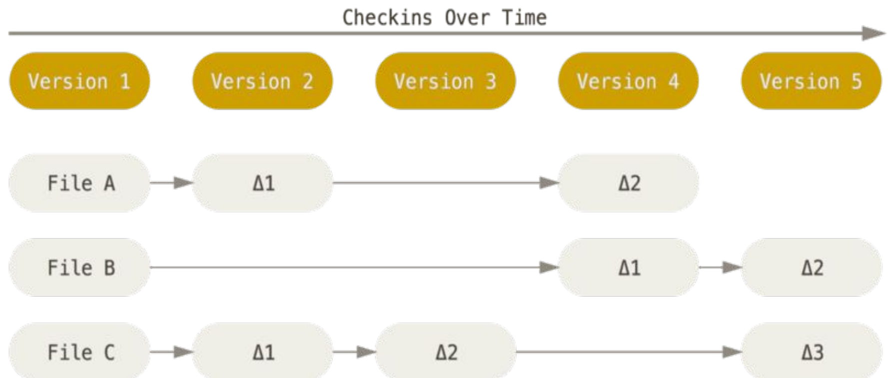
Highly Recommended

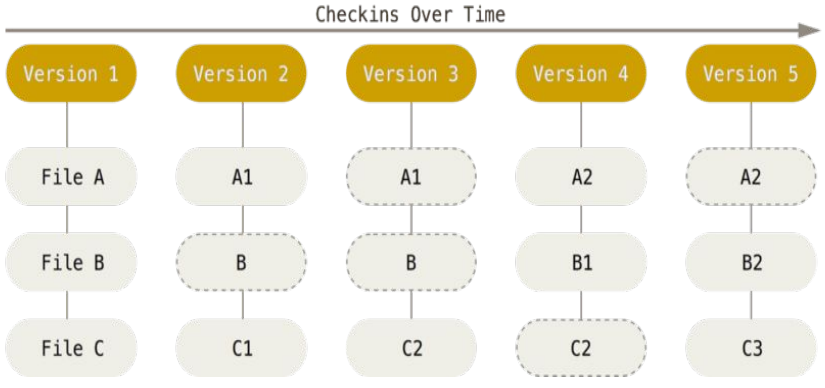
- Courtesy of Prof. Bogdan Vasilescu (teaches this course last & next Spring)
- (second) most useful life skill you will have learned in 214/514



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

SVN (left) vs. Git (right)





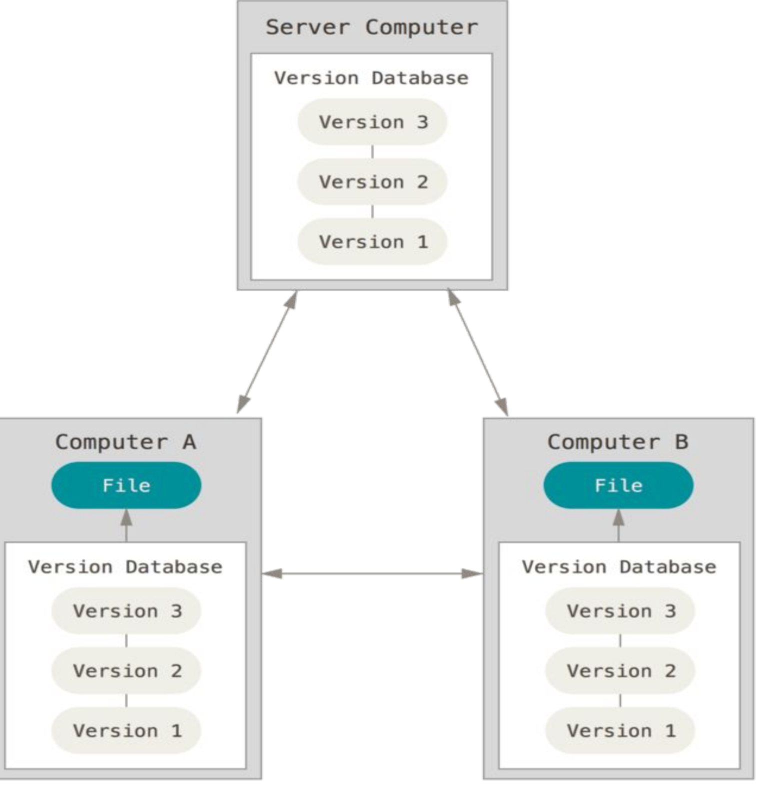
- SVN stores changes to a base version of each file
- Version numbers (1, 2, 3, …) are increased by one after each commit

- Git stores each version as a snapshot
- If files have not changed, only a link to the previous file is stored
- Each version is referred by the SHA-1 hash of the contents

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

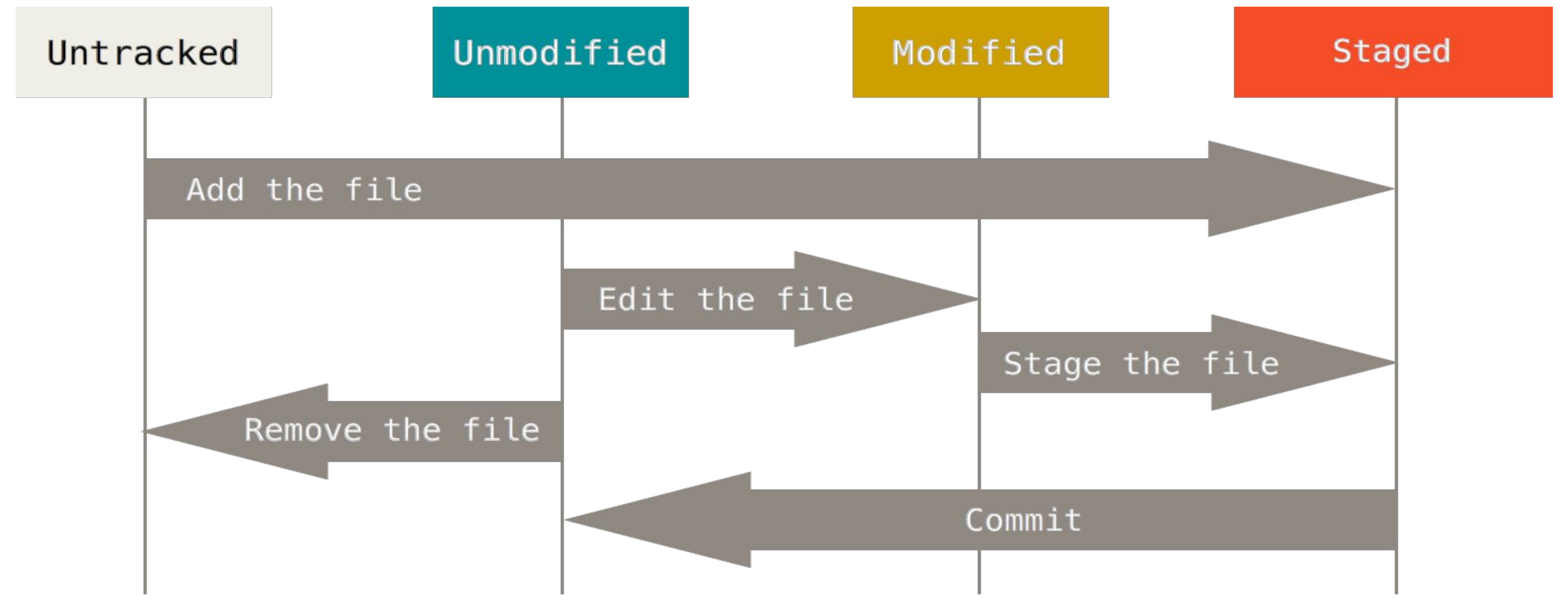
Distributed version control

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



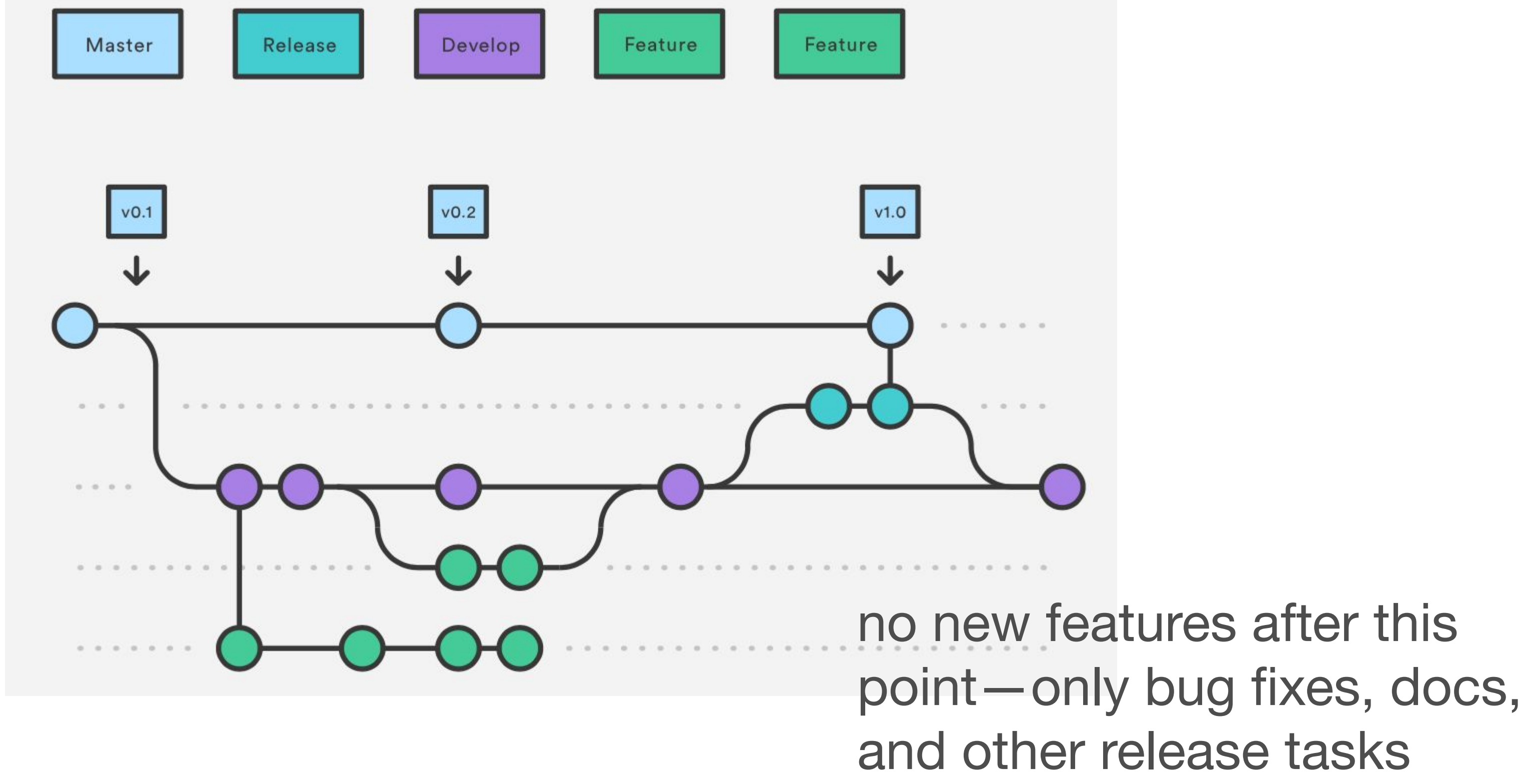
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Aside: Git process



© Scott Chacon "Pro Git"

GitFlow release branches (eventually into master)



no new features after this point—only bug fixes, docs, and other release tasks

institute for SOFTWARE RESEARCH

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.

Principles of Software Construction: Objects, Design, and Concurrency

A Tour of the 23 GoF Design Patterns

Bogdan Vasilescu **Jonathan Aldrich**

Where we are

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

Course so far…

Creational:

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

Structural:

1. **Adapter**
2. Bridge
3. **Composite**

9. **Decorator**
10. Façade
11. Flyweight
12. **Proxy**

Behavioral:

9. Chain of Responsibility
10. Command
11. Interpreter

16. **Iterator**
17. Mediator
18. Memento
19. **Observer**
20. State
21. **Strategy**
22. **Template method**
23. Visitor

Not in the book:

- Model view controller
- Promise
- Module (JS)

isr institute for SOFTWARE RESEARCH

Warm Up Scenario

You are developing a mobile application for cities where users can report potholes and similar problems (with photos) and city crews can investigate, prioritize, and address reports.

Design problem 1: You want to create monthly reports. However, different cities want this report slightly differently, with different text on top and sorted in different ways. You want to vary text and sorting in different ways.

Singleton Illustration

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

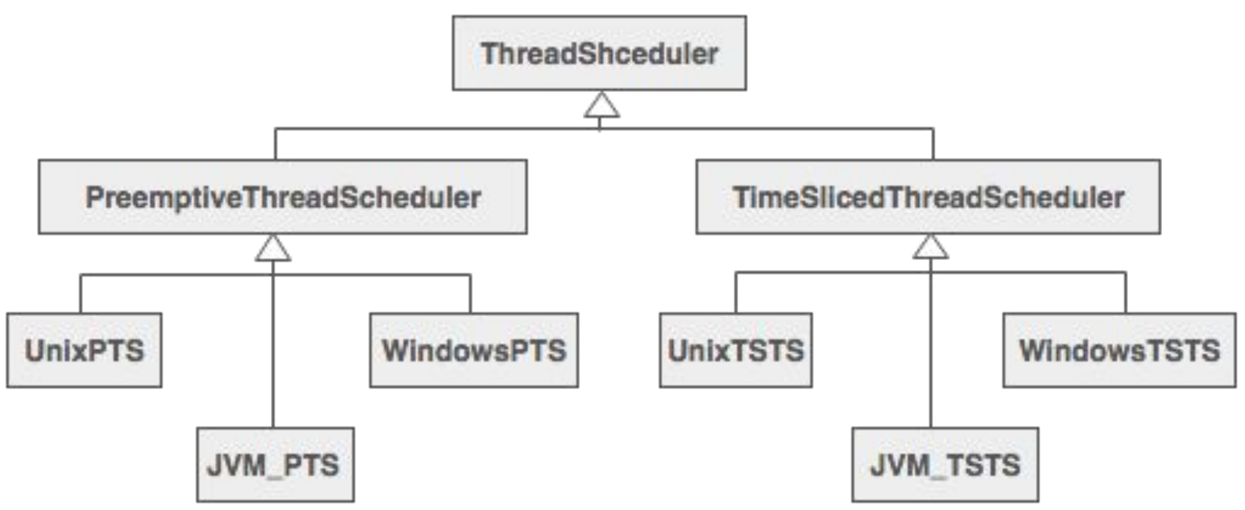
```javascript
const elvis = { … }
function getElvis() {

export { getElvis }
```

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 */ }
}
```

(New) Problem: we have to define a class for each permutation of these two dimensions



How would you redesign this?

image source: https://sourcemaking.com

Bridge Pattern: Decompose the component's interface and implementation into orthogonal class hierarchies.

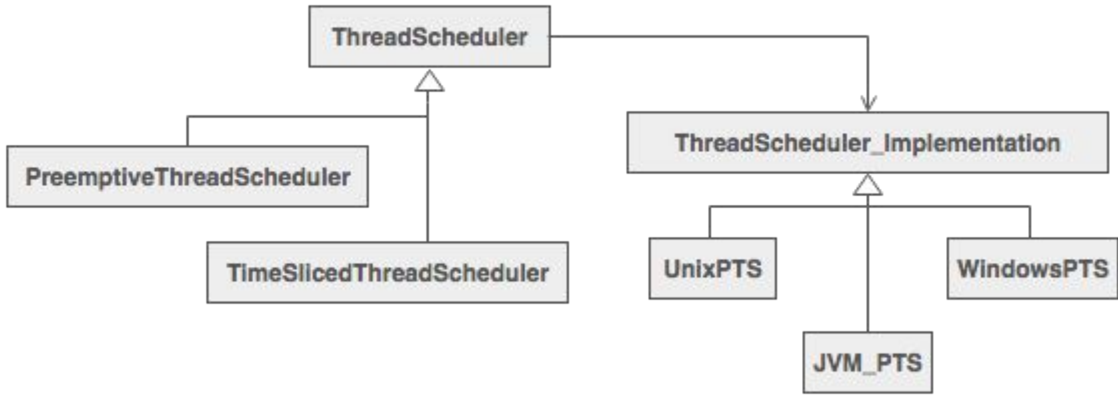
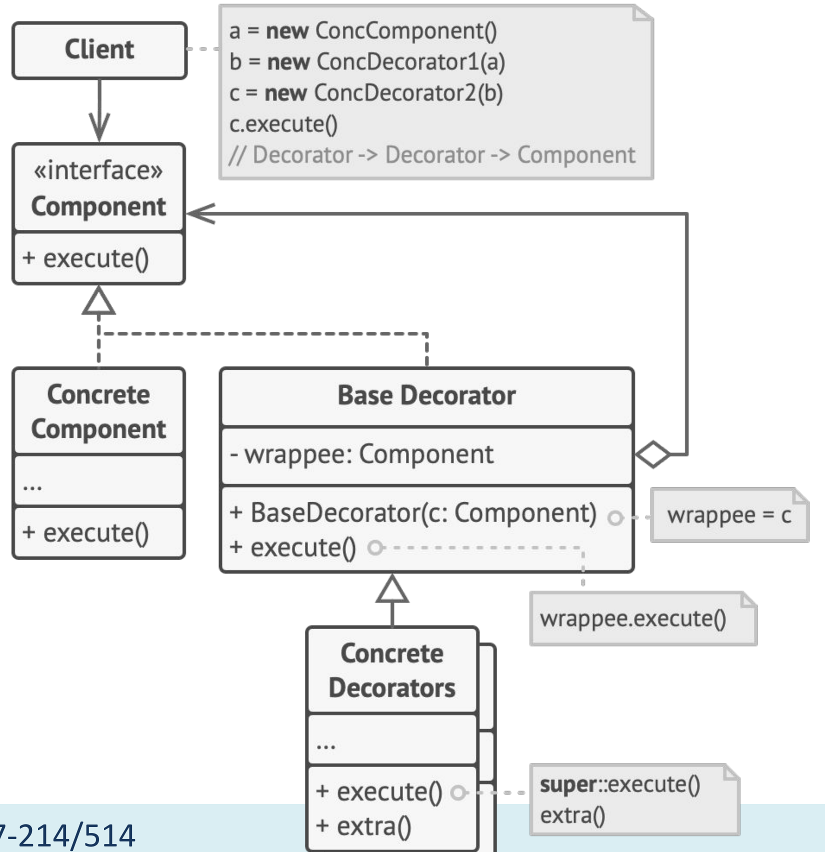
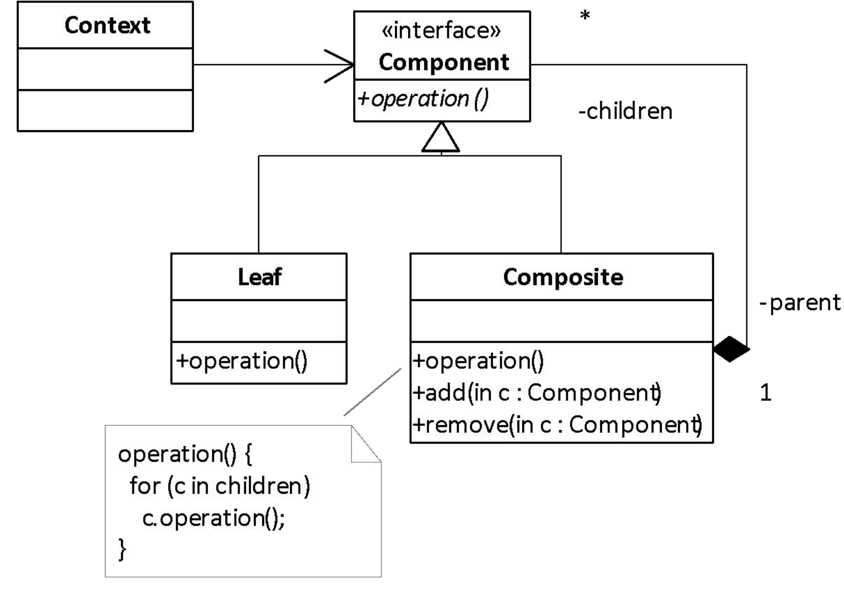


image source: https://sourcemaking.com

Decorator vs Composite?

Cardinality is the difference, but also the intent.

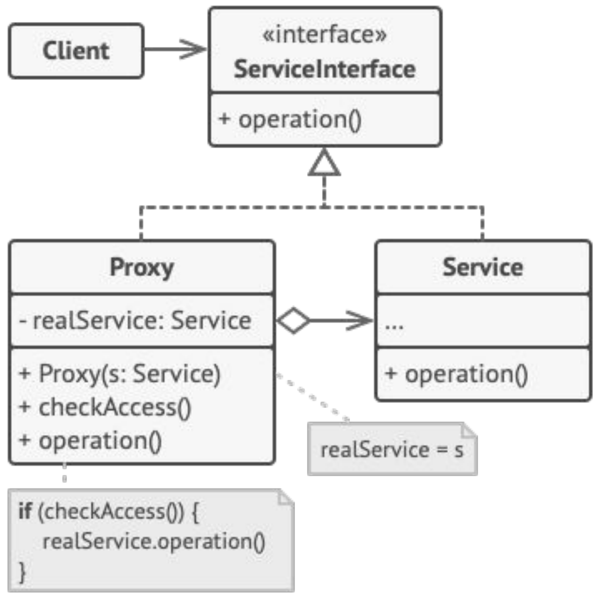


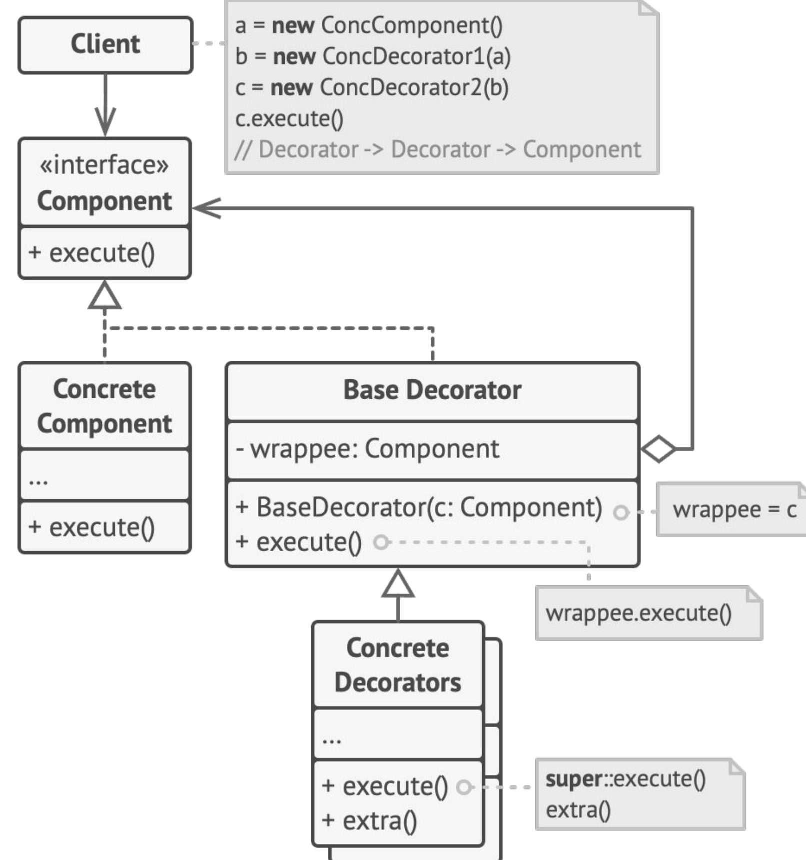


isr institute for SOFTWARE RESEARCH

Proxy vs Decorator?

Some variants of proxy are almost identical to decorator. But the intents of the patterns are different.





isr institute for SOFTWARE RESEARCH

Principles of Software Construction: Objects, Design, and Concurrency

{Static & Dynamic} x {Typing & Analysis}

Jonathan Aldrich Bogdan Vasilescu

How Do You Find Bugs?

- Run it?

```java
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)
```

Also: Static Analysis!

How?

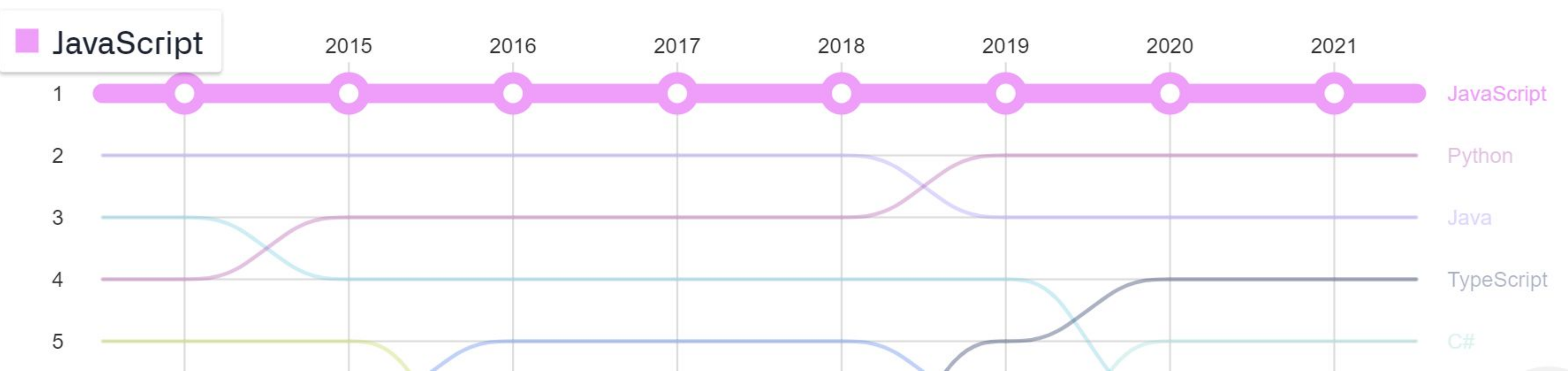
- We know at *compile time* where `getValue` gets routed to
- `getValue` calls a method on `i`
- `i` can be `null`

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

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

Passing 'null' argument to parameter annotated as @NotNull

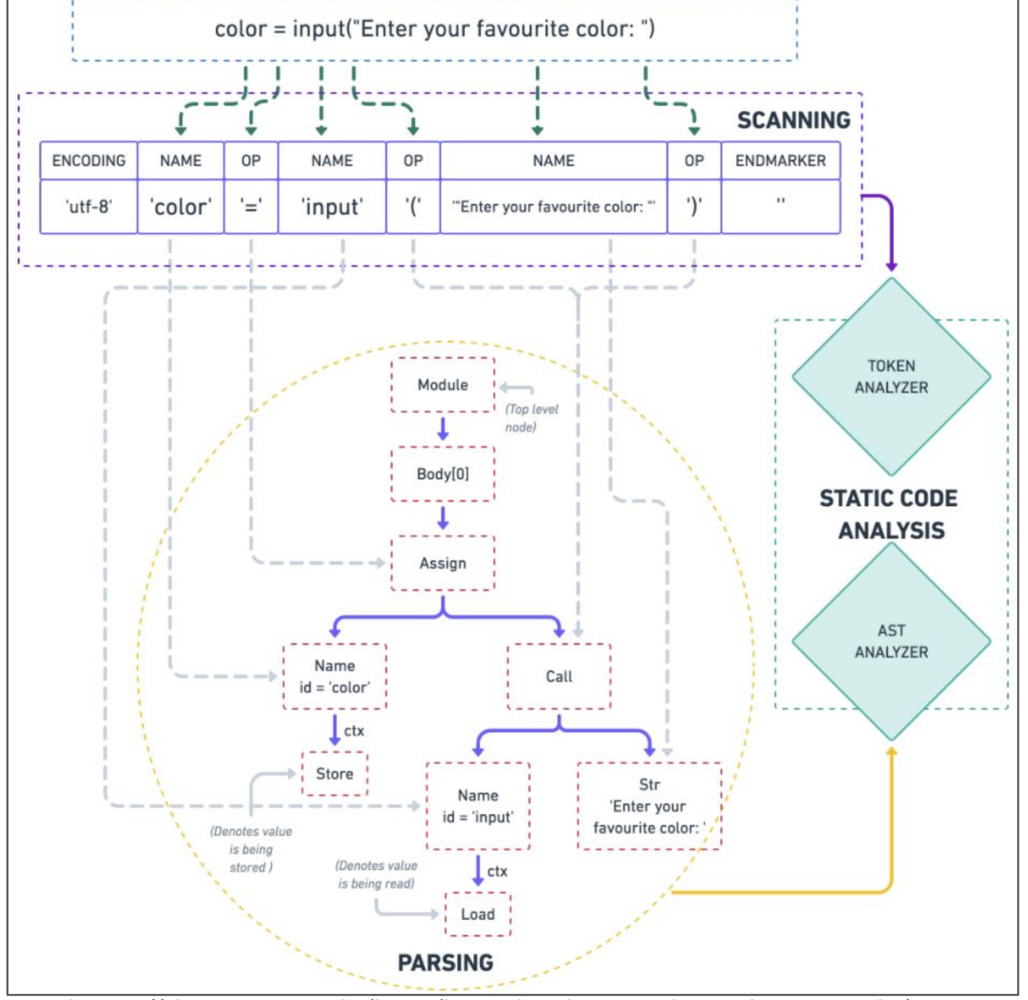
Static vs. Dynamic Typing

Okay, but:



Static Analysis

- How?
  - Program analysis + Vocabulary of patterns



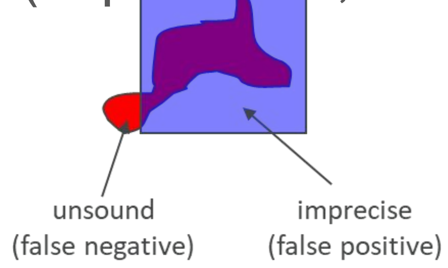
https://deepsource.io/blog/introduction-static-code-analysis/

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

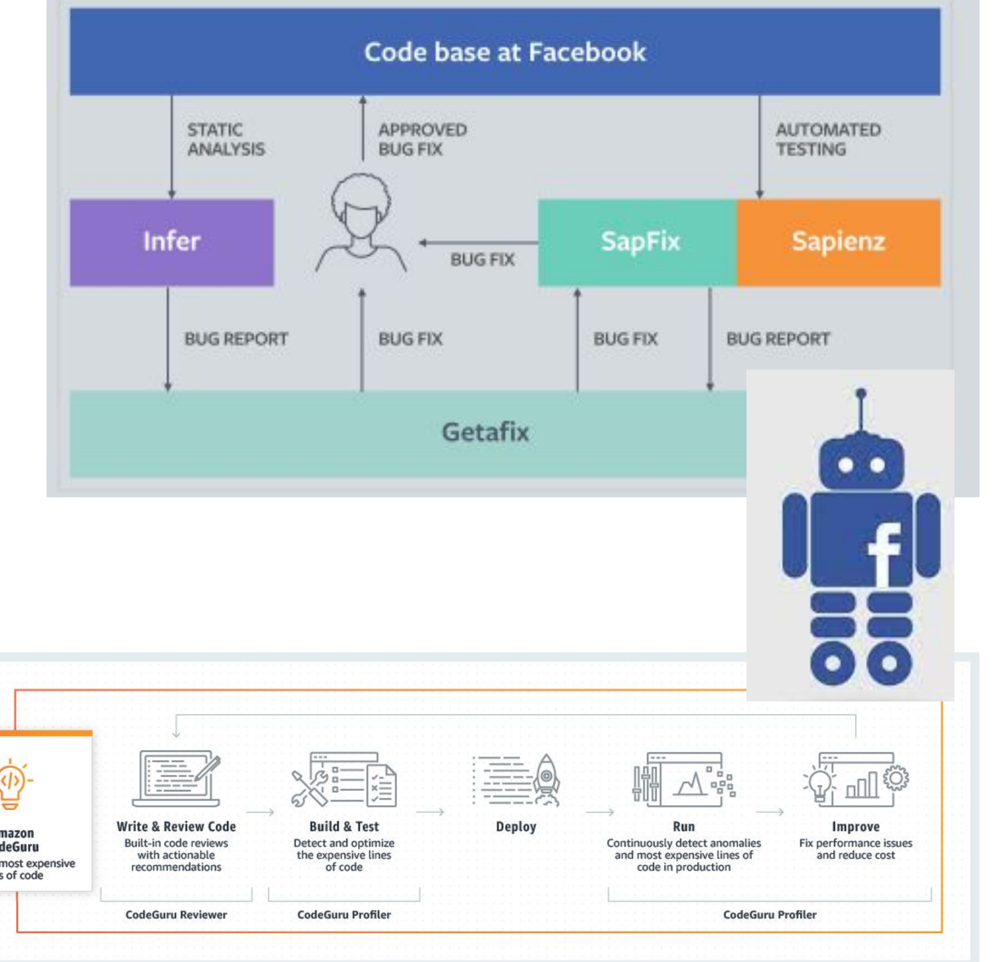
```
package com.google.devtools.staticanalysis;

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

DevOps (part 1)

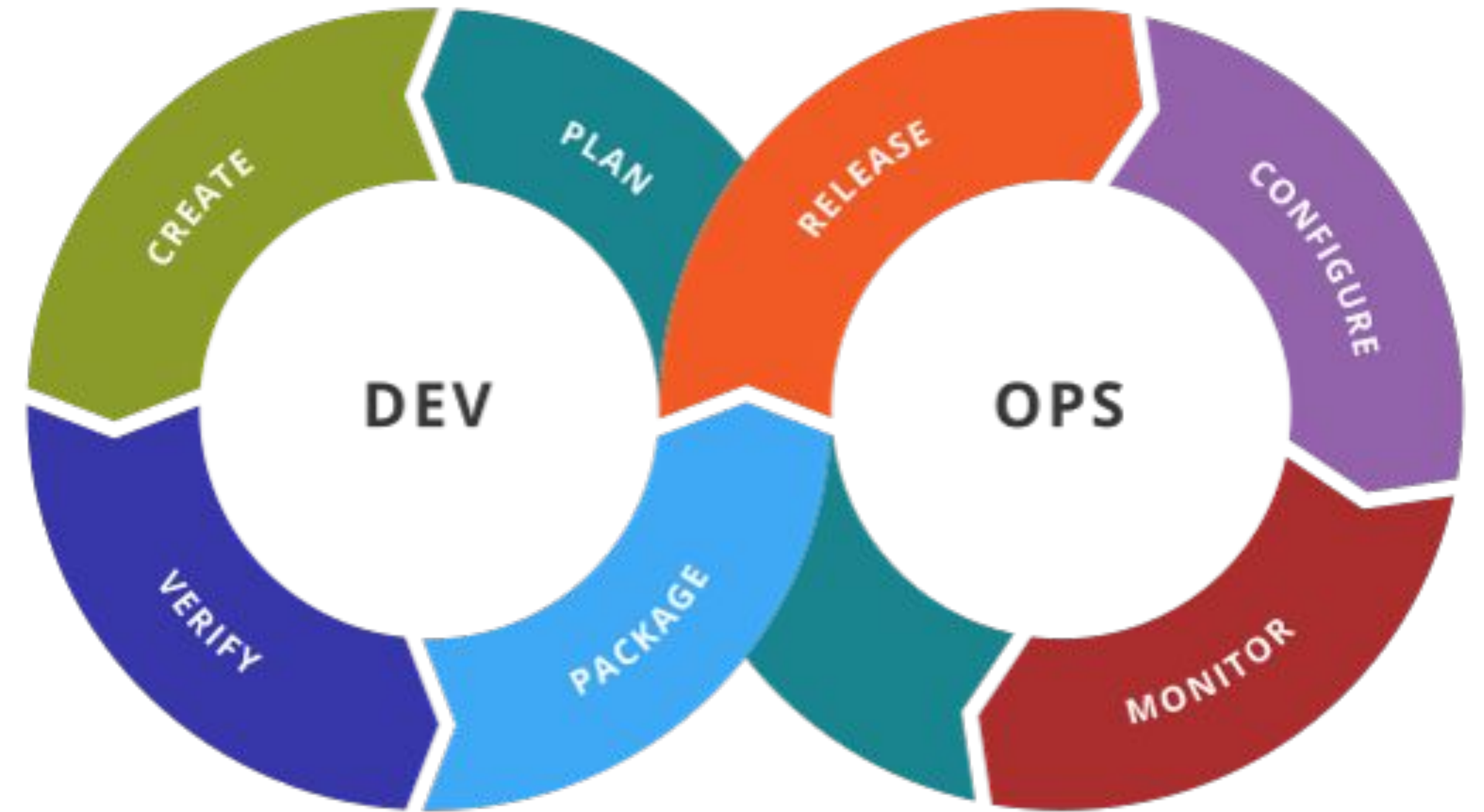
Jonathan Aldrich **Bogdan Vasilescu**

Where we are

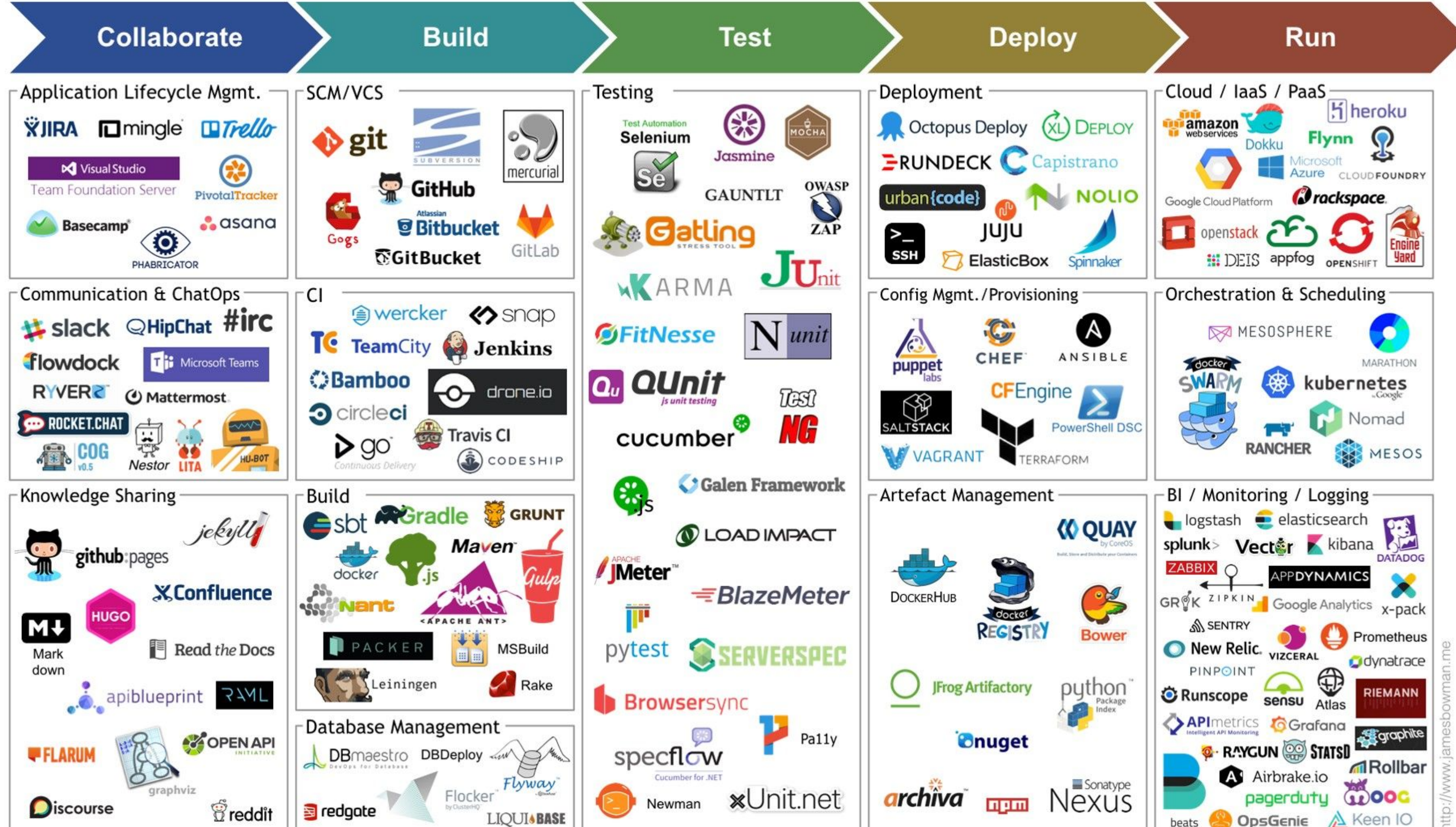
	Small scale: One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
Design for understanding change/ext. reuse robustness ...	Subtype Polymorphism ✓ Information Hiding, Contracts ✓ Immutability ✓ Types ✓ Static Analysis ✓ Unit Testing ✓	Domain Analysis ✓ Inheritance & Del. ✓ Responsibility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

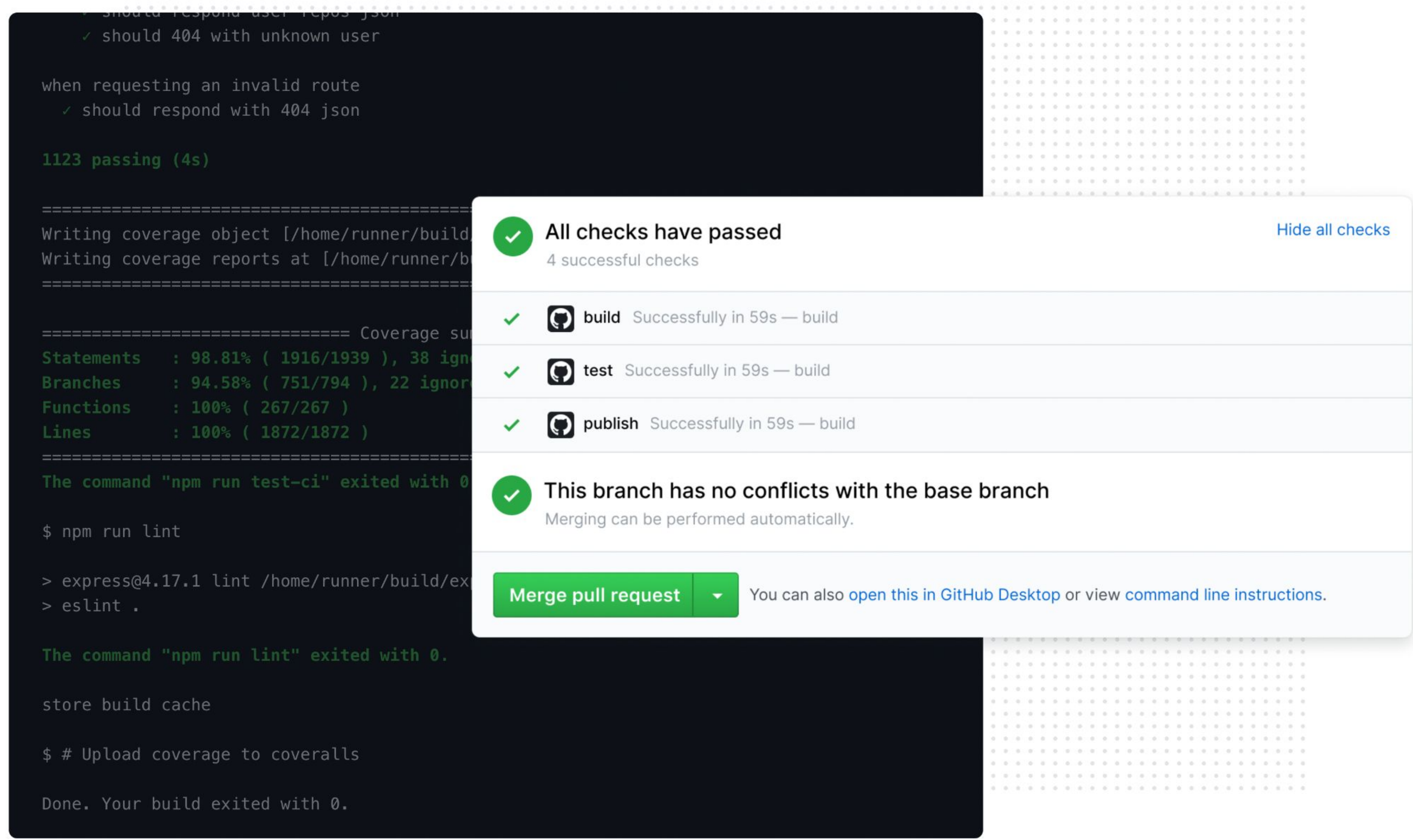
Early days:
Boxed software, infrequent releases

S3D


PLAN
CREATE
VERIFY
PACKAGE
RELEASE
CONFIGURE
MONITOR
DEV
OPS

Heavy Automation, Lots of Tooling




✓ should 404 with unknown user
when requesting an invalid route
✓ should respond with 404 json
1123 passing (4s)
Writing coverage object [/home/runner/build/
Writing coverage reports at [/home/runner/b
Coverage su
Statements : 98.81% (1916/1939), 38 ign
Branches : 94.58% (751/794), 22 ignor
Functions : 100% (267/267)
Lines : 100% (1872/1872)
The command "npm run test-ci" exited with 0
$ npm run lint
> express@4.17.1 lint /home/runner/build/ex
> eslint .
The command "npm run lint" exited with 0.
store build cache
$ # Upload coverage to coveralls
Done. Your build exited with 0.
All checks have passed
4 successful checks
Hide all checks
build Successfully in 59s — build
test Successfully in 59s — build
publish Successfully in 59s — build
This branch has no conflicts with the base branch
Merging can be performed automatically.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.

Aside: The role of signaling

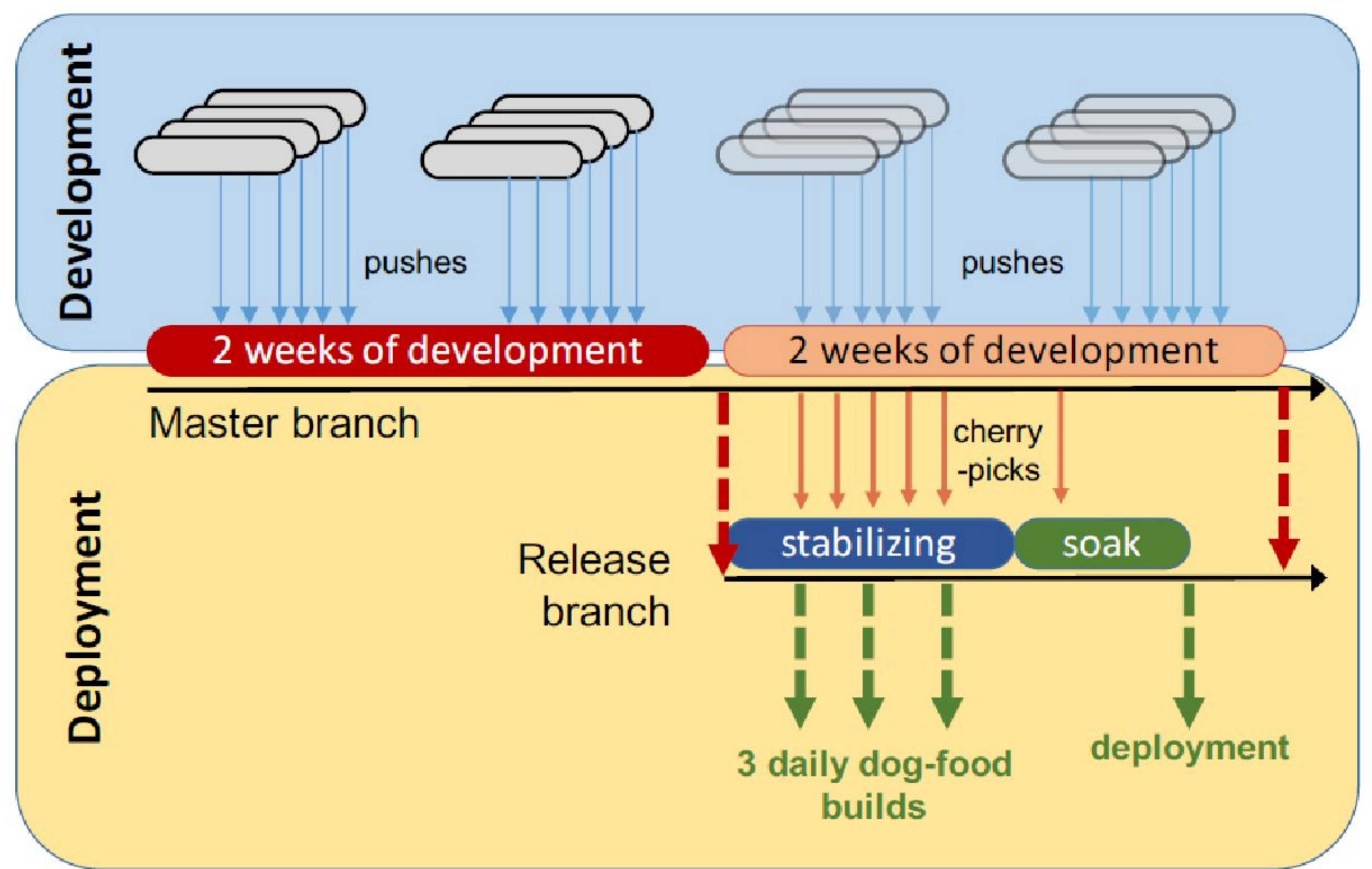
Status

Build Pipeline

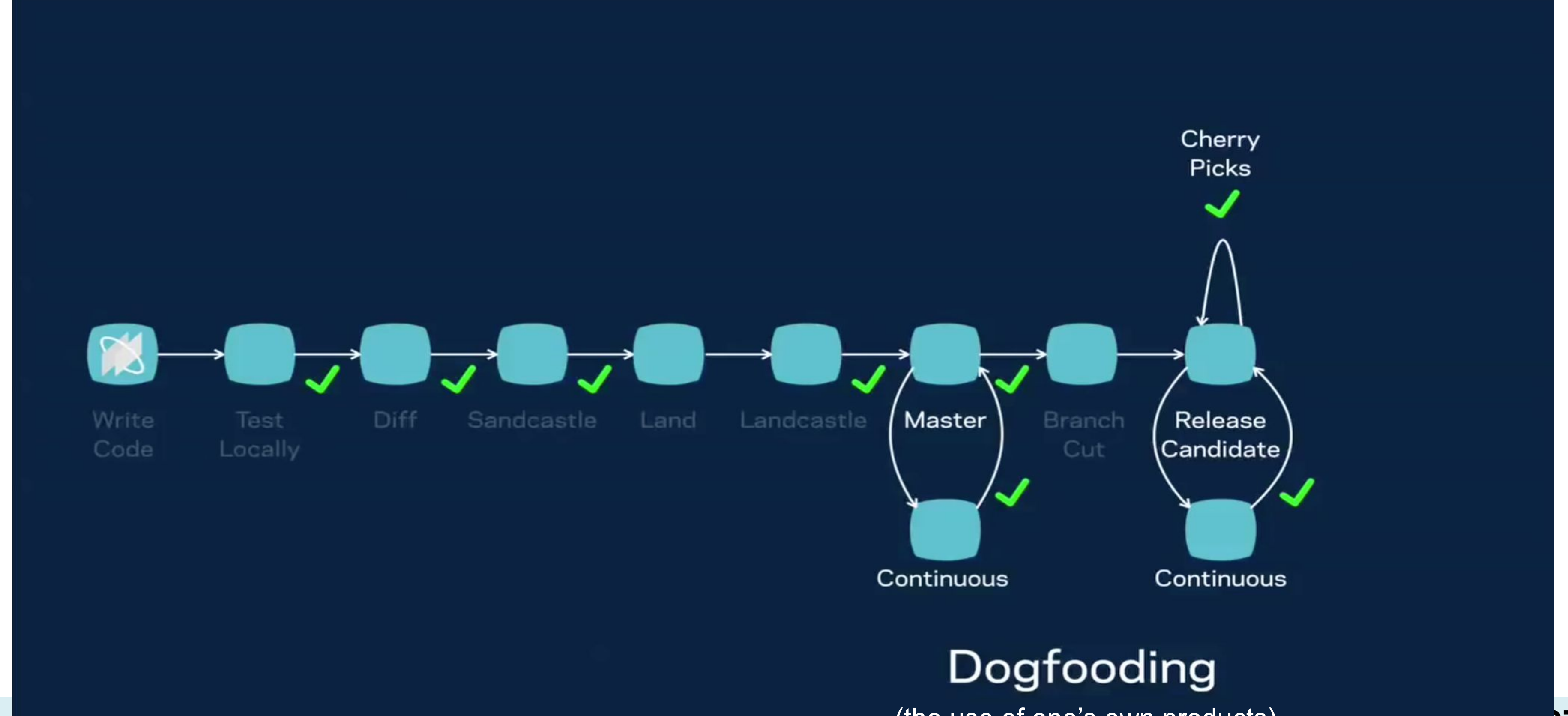
Release Pipeline

Dev	Test	Prod
deployment succeeded	deployment succeeded	deployment succeeded
NuGet 0.6.0	NuGet 0.6.0	NuGet 0.4.0

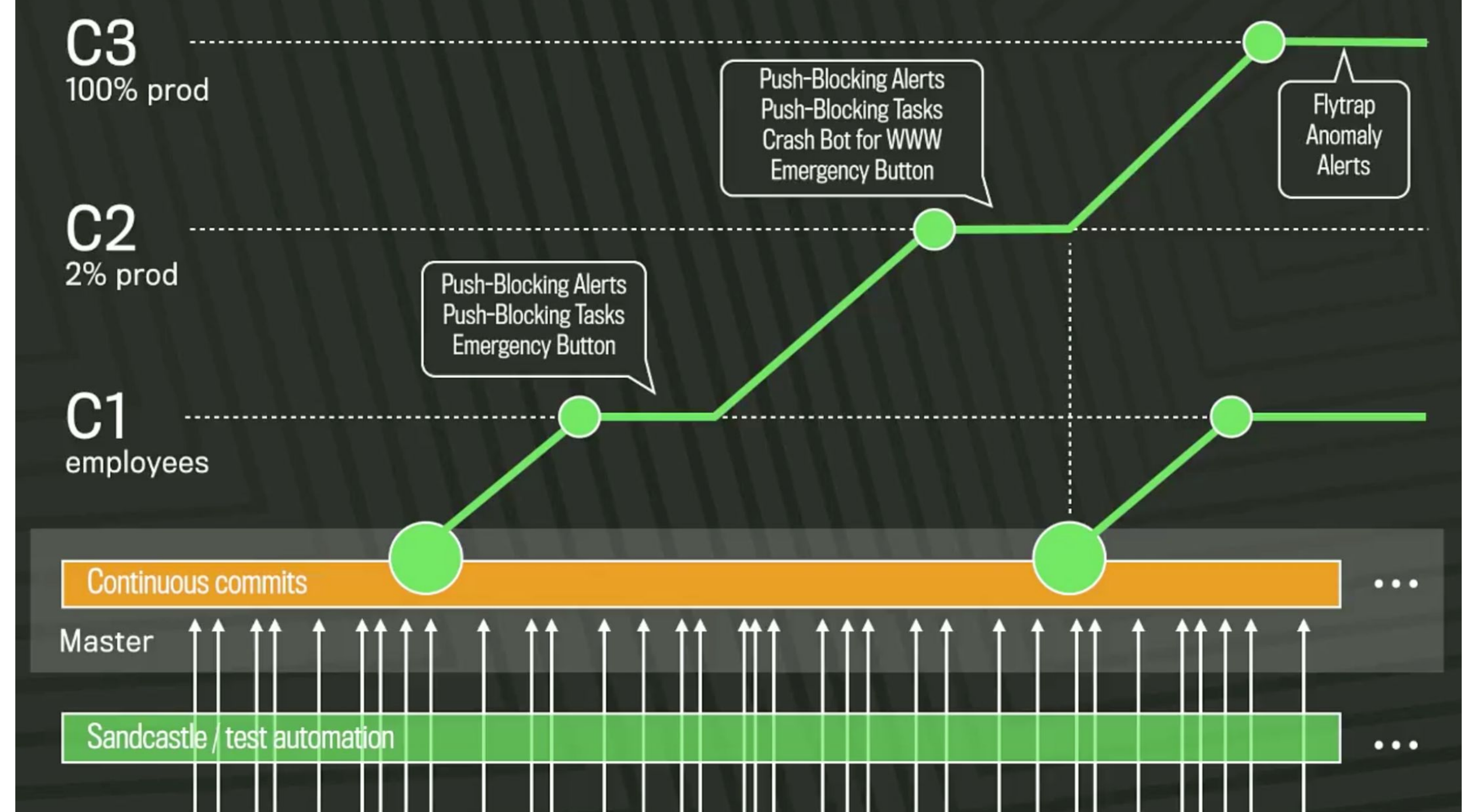
https://blog.devops4me.com/status-badges-in-azure-devops-pipelines/


Development
pushes
pushes
2 weeks of development
2 weeks of development
Master branch
cherry
-picks
Deployment
Release
branch
stabilizing
soak
3 daily dog-food
builds
deployment

Diff lifecycle: diff ends up on main branch



Quasi-continuous push from master (1,000+ devs, 1,000 diffs/day); 10 pushes/day



A/B Testing

Original: 2.3%

Long Form: 4.3%

Groove

Sign Up

Everything you need to deliver awesome, personal support to every customer.

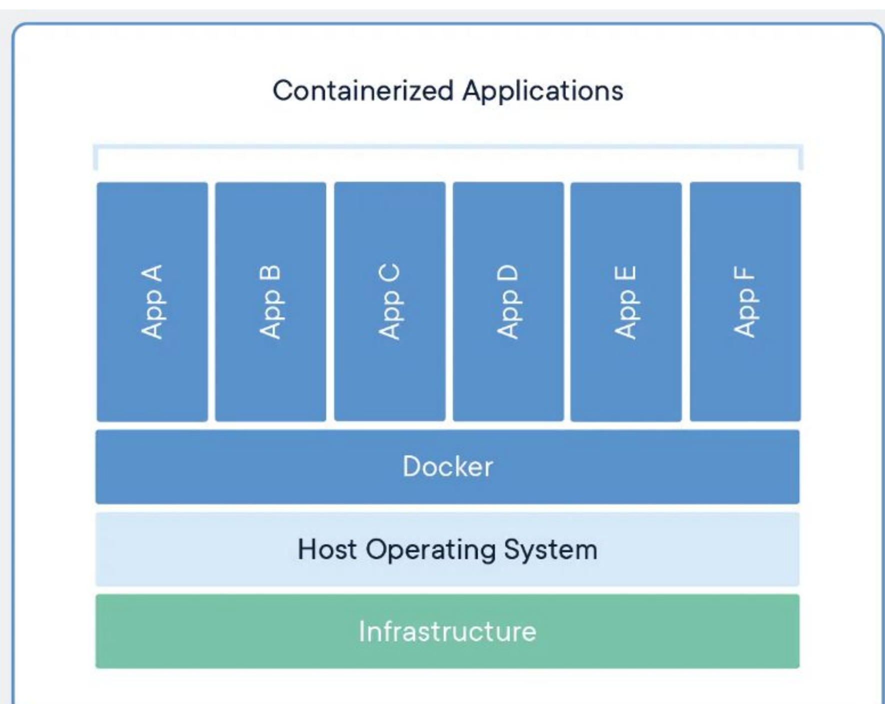
Assign support emails to the right people, feel confident that customers are being followed up with and always know what's going on.

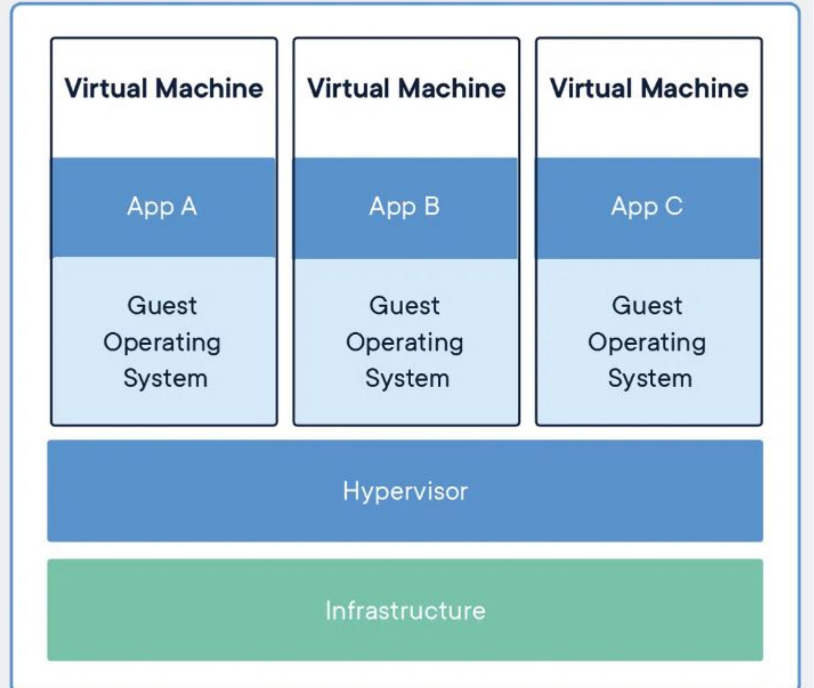
Principles of Software Construction: Objects, Design, and Concurrency

Containers & Cloud (or DevOps part 2)

Jonathan Aldrich Bogdan Vasilescu **Matt Davis**

Containers offer Virtualization on the OS





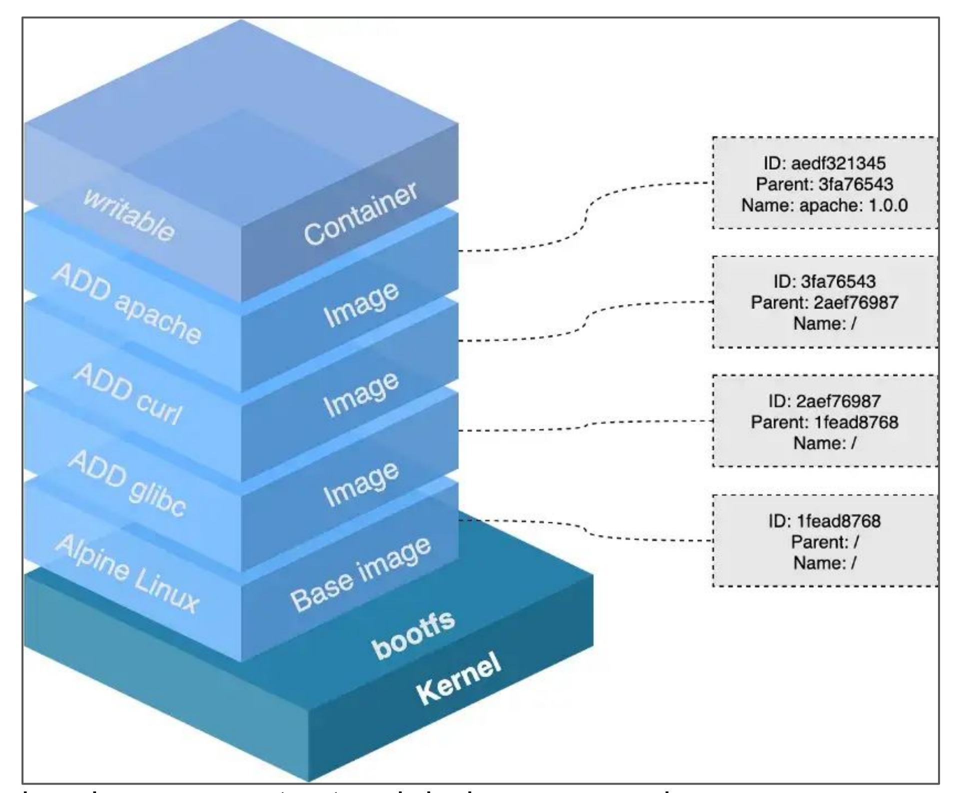
https://www.docker.com/resources/what-container/

Docker images are *layers*

- Each action yields a new layer
- The base layer is typically an OS
  - E.g., “ubuntu:20.04”
- Data from previous layers is “copy-on-write”

Consequences:

- Layer-stacks are easily reused making images very light
- Security via IO permissions



https://ragin.medium.com/docker-what-it-is-how-images-are-structured-docker-vs-vm-and-some-tips-part-1-d9686303590f

There's more in the cloud

SaaS

PaaS

IaaS

Hosted applications

Development tools, database management, business analytics

Operating systems

Servers and storage

Networking firewall

...sical plant / ...ng

Run

Cloud / IaaS / PaaS

amazon web services, Dokku, heroku, Flynn, Microsoft Azure, CLOUD FOUNDRY, Google Cloud Platform, rackspace, openstack, DEIS, appfog, OPENSHIFT, Engine Yard

Orchestration & Scheduling

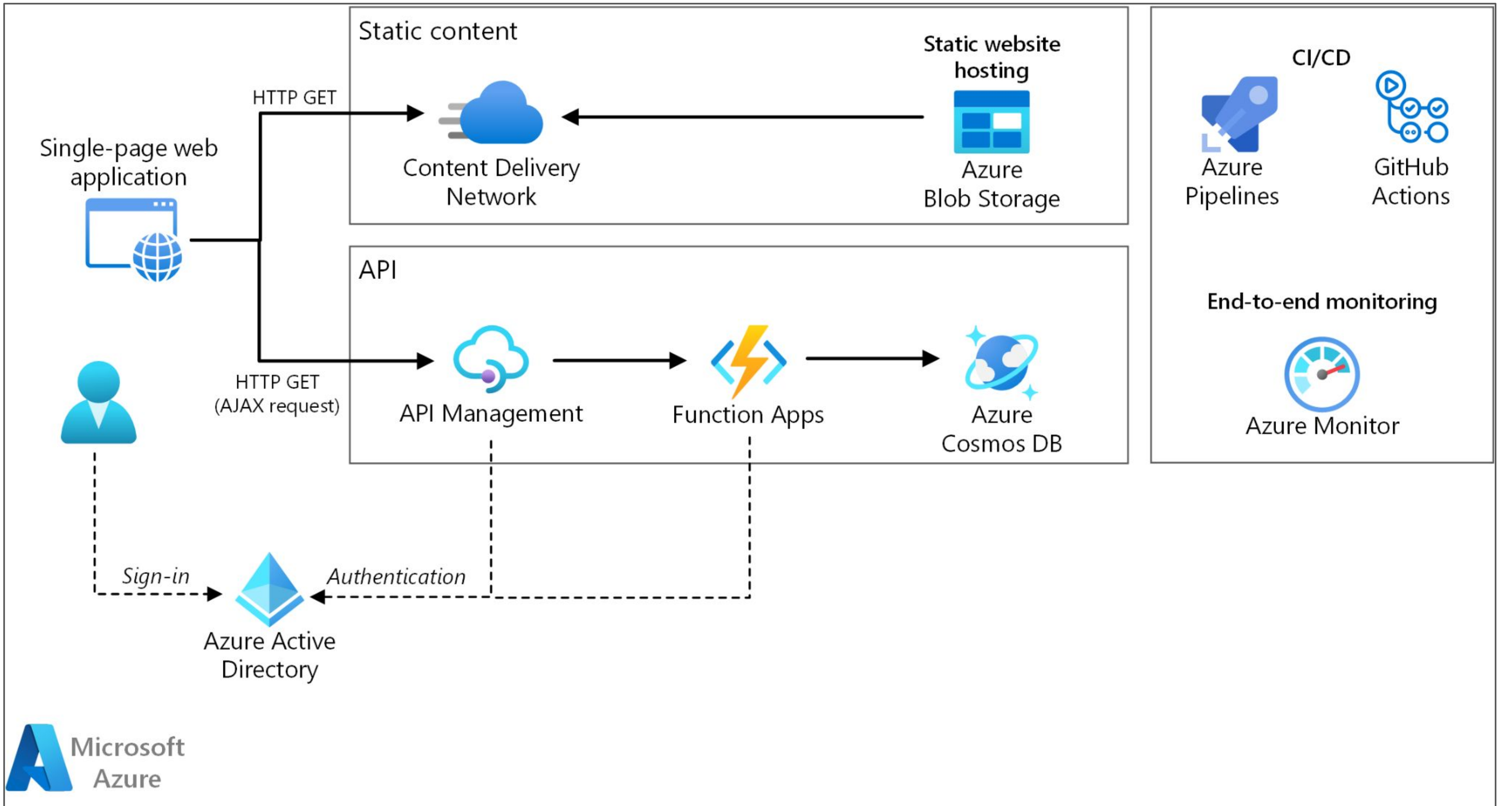
MESOSPHERE, MARATHON, docker SWARM, kubernetes, RANCHER, Nomad, MESOS

BI / Monitoring / Logging

logstash, elasticsearch, splunk>, Vector, kibana, DATADOG, ZABBIX, APPDYNAMICS, GROK, ZIPKIN, Google Analytics, x-pack, SENTRY, New Relic, VIZCERAL, Prometheus, dynatrace, PINPOINT, Runscope, sensu, Atlas, RIEMANN, APImetrics, Grafana, graphite, RAYGUN, STATSD, Rollbar, Airbrake.io, pagerduty, beats, OpsGenie, Keen IO

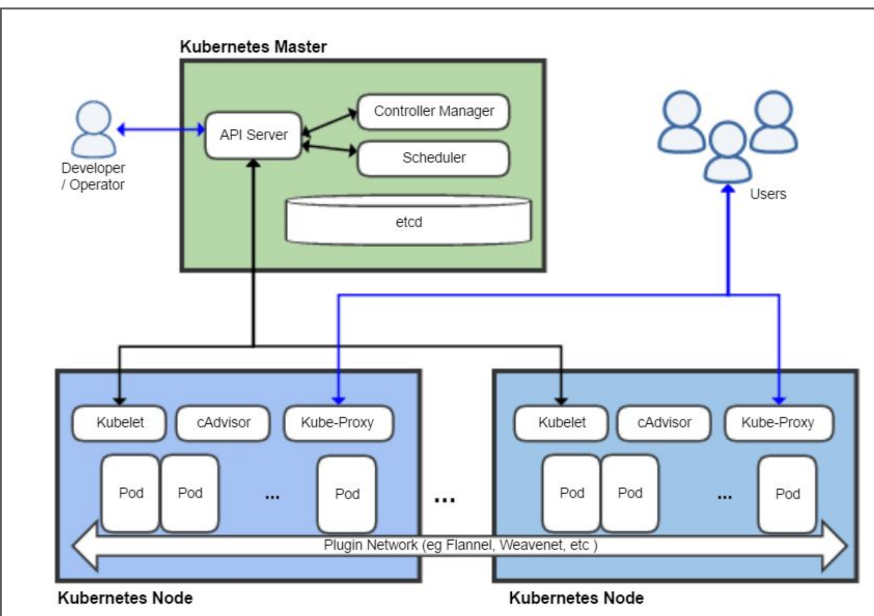
https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iaas/

Note: not everyone thinks of these as nested categories


Static content
Static website hosting
HTTP GET
Content Delivery Network
Azure Blob Storage
Single-page web application
API
HTTP GET (AJAX request)
API Management
Function Apps
Azure Cosmos DB
CI/CD
Azure Pipelines
GitHub Actions
End-to-end monitoring
Azure Monitor
Sign-in
Authentication
Azure Active Directory
Microsoft Azure

Managing Systems with Kubernetes

- Note how much this decouples the client from the code
  - In our previous systems, the client talked directly to the frontend
  - Now, to a data center, which talks to a proxy, to a pod, to a container, to code



Finally, is the Cloud right for you?

- You're borrowing someone else's computer
  - That comes at a big premium
    - Hosting on-prem can be many times cheaper
    - I recall a thread where a Twitter engineer said their AWS bill would be $100M+/month if they went that way
  - Also fewer guarantees
    - Some VMs are rarely available
    - Allocating large nrs of any kind almost certainly requires discussion
- Still worth it if you:
  - Are a small team, can't spare cycles for system ops
  - Are growing quickly, won't know your computing needs far out

Looking Forward: Beyond Code-Level Concerns

Where we are

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

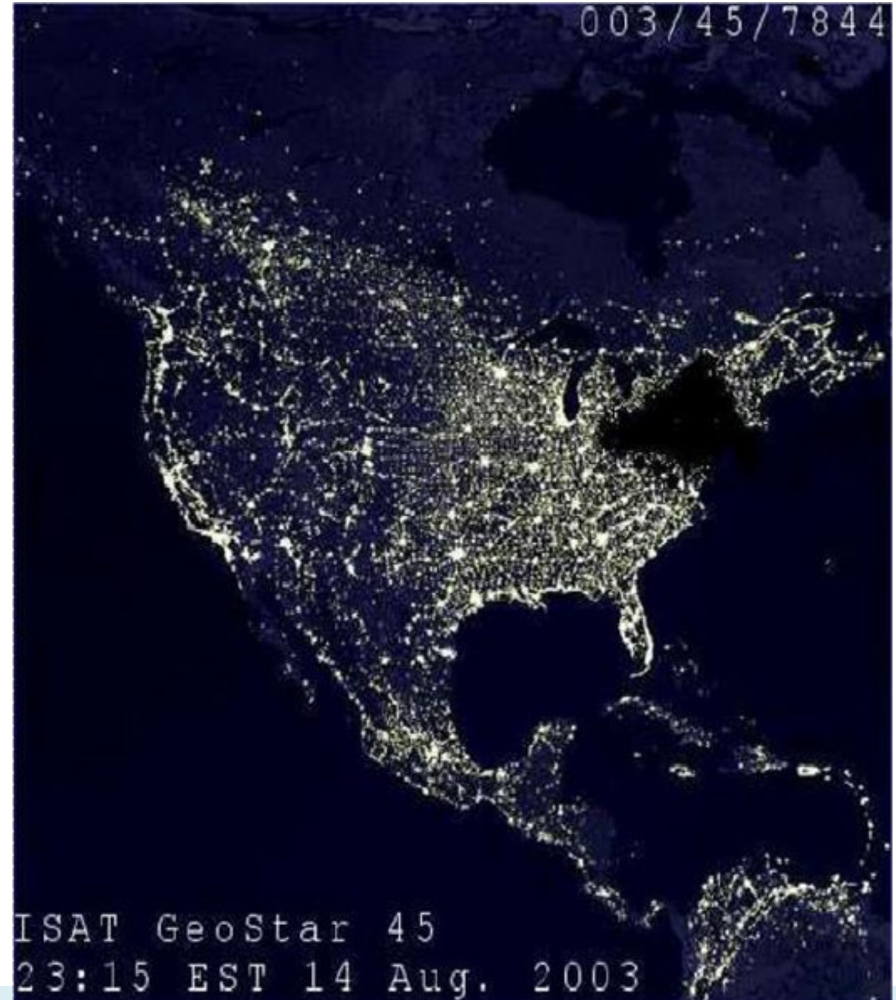
Bookout Trial Reporting

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

"Task X death in combination with other task deaths"

© Copyright 2014, Philip Koopman. CC Attribution 4.0 International license.

S3D


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

3 Share 6 Tweet 0 Linkedin 0 Email 7 Comment

GET BUSINESS NEWSLETTERS:
Enter email SUBSCRIBE

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 transparency

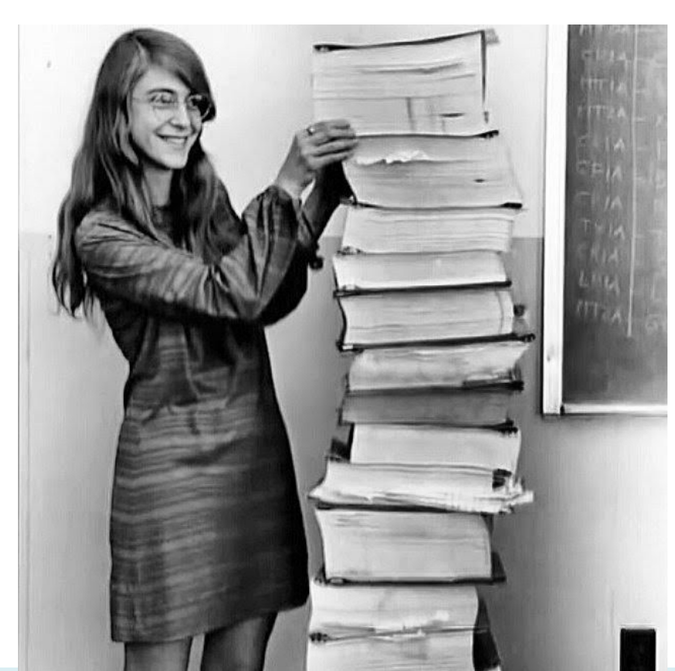
“But we’re CMU students and we are really, really smart!”

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

3 Share | 6 Tweet | 0 Linked in | 0 Email | 7 Comment

GET BUSINESS NEWSLETTERS:
Enter email SUBSCRIBE

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 transparency

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

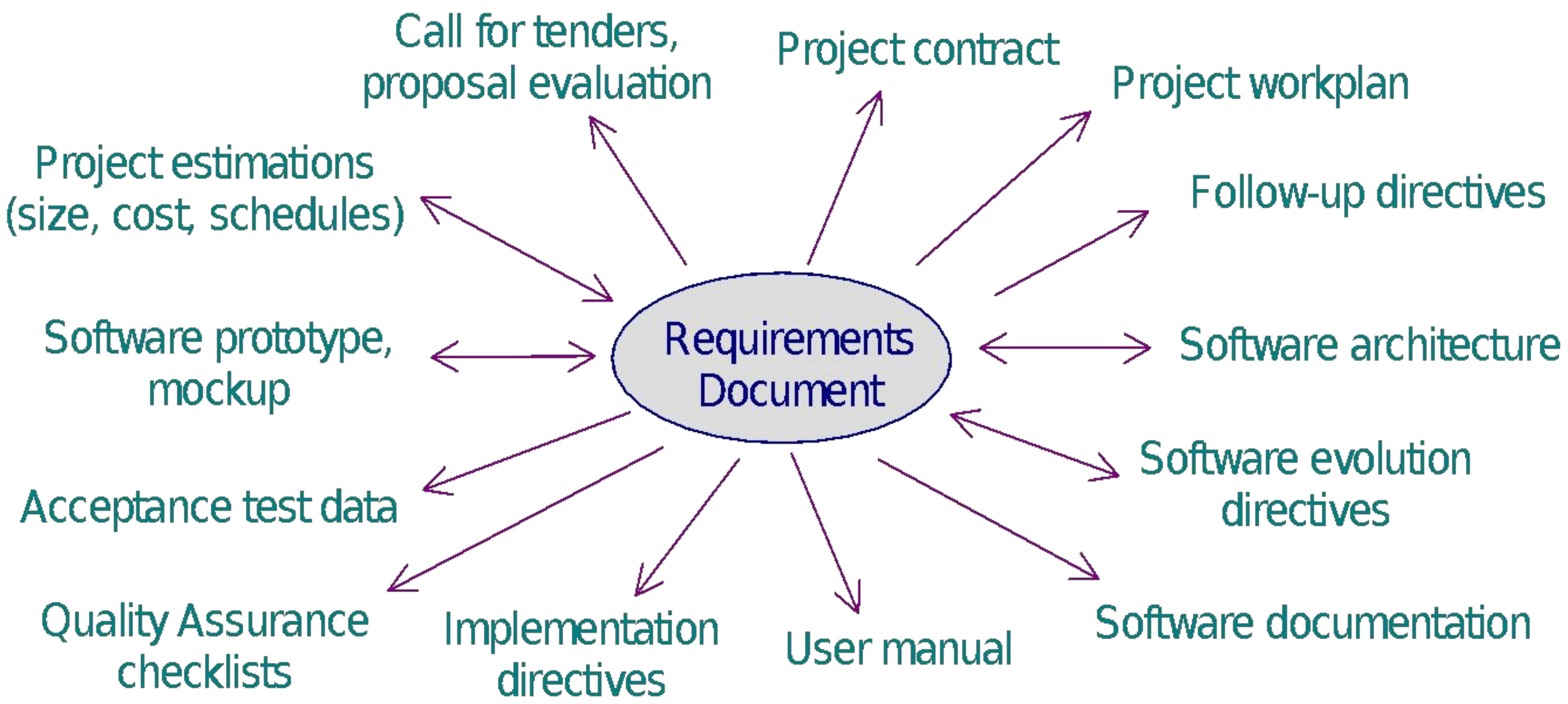
Key challenges in how to

- 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?

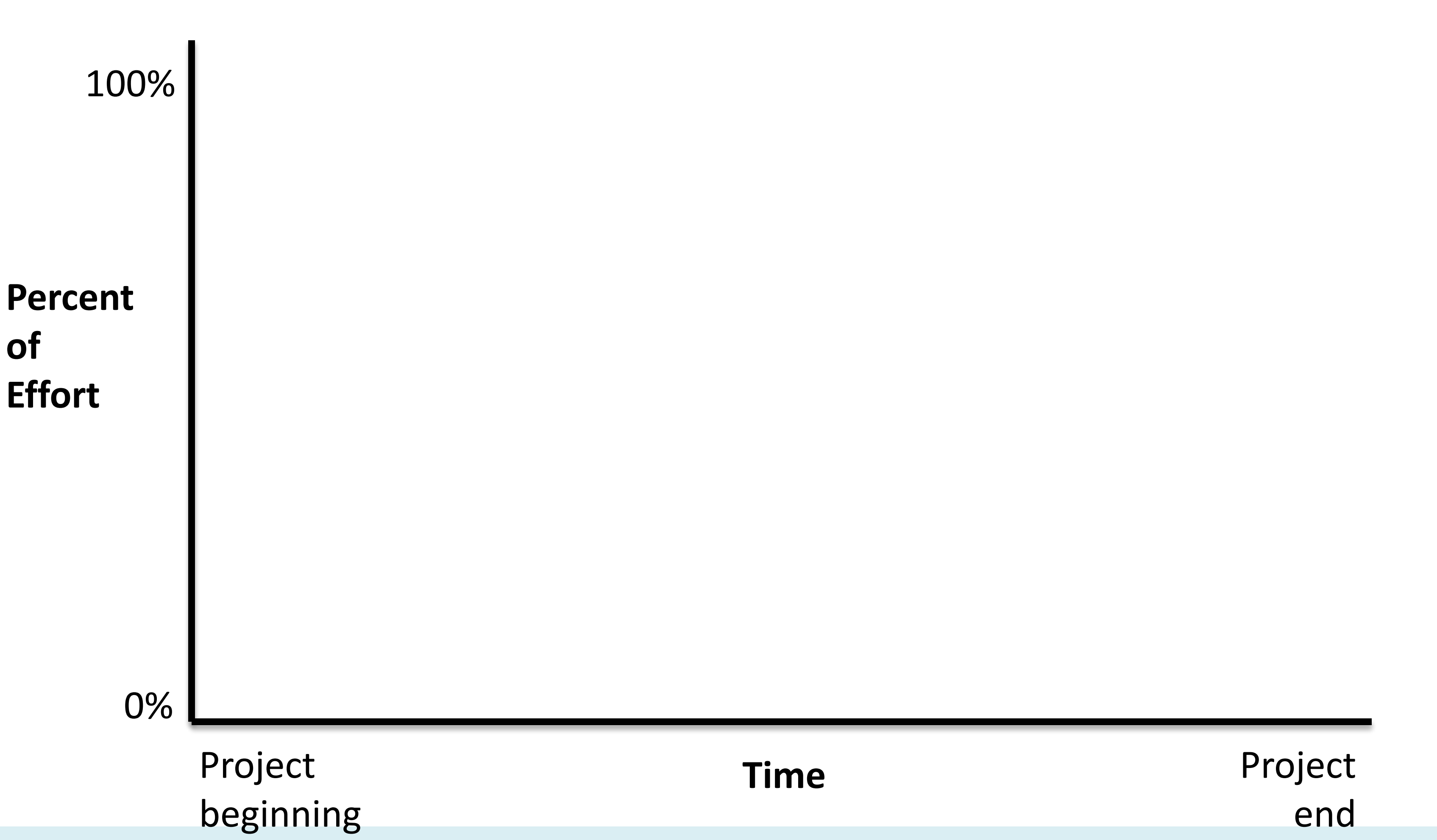
1. Discuss the software that needs to be written
2. Write some code
3. Test the code to identify the defects
4. Debug to find causes of defects
5. Fix the defects
6. If not done, return to step 1

Software Process

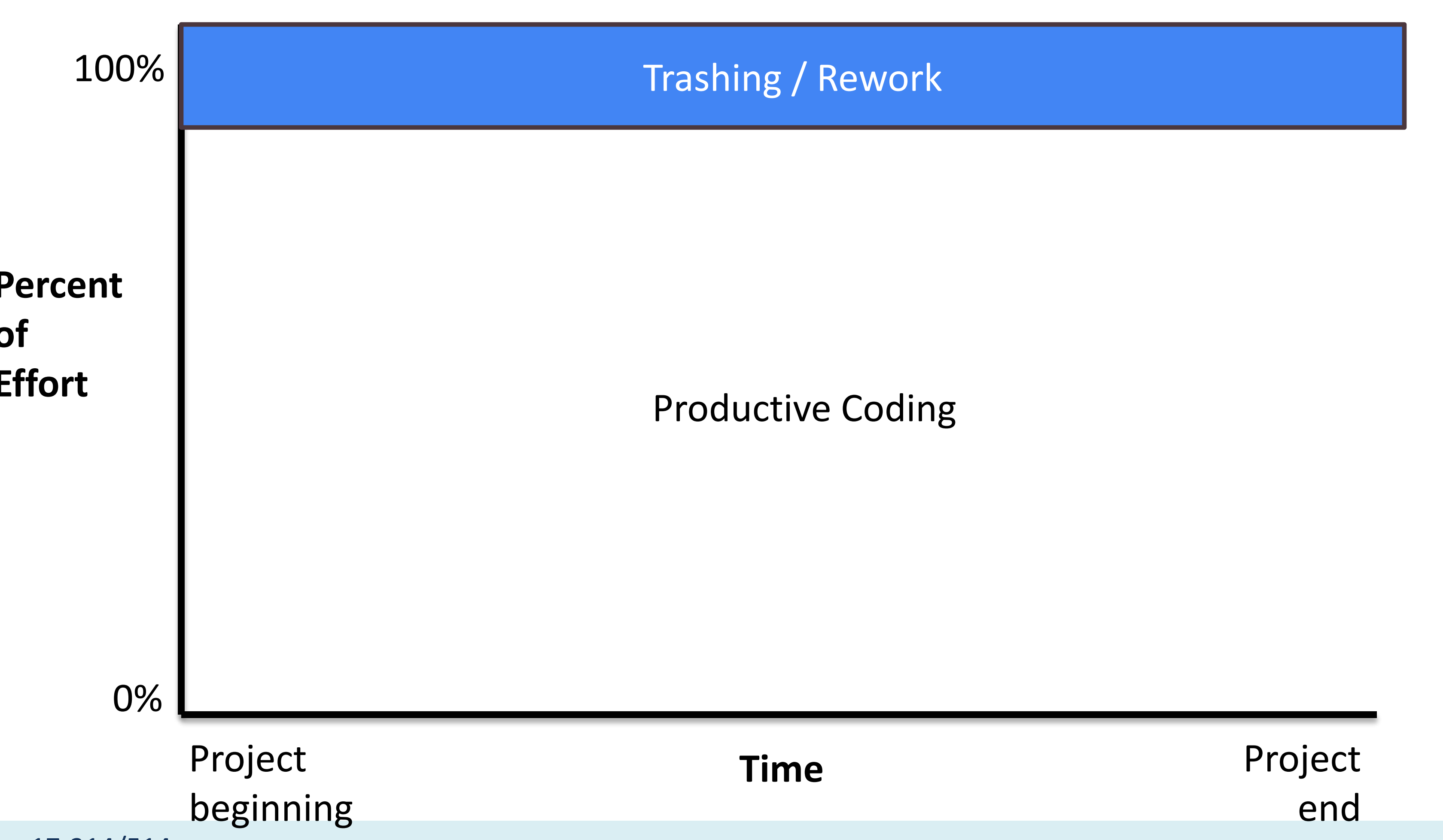
"The set of activities and associated results that produce a software product"

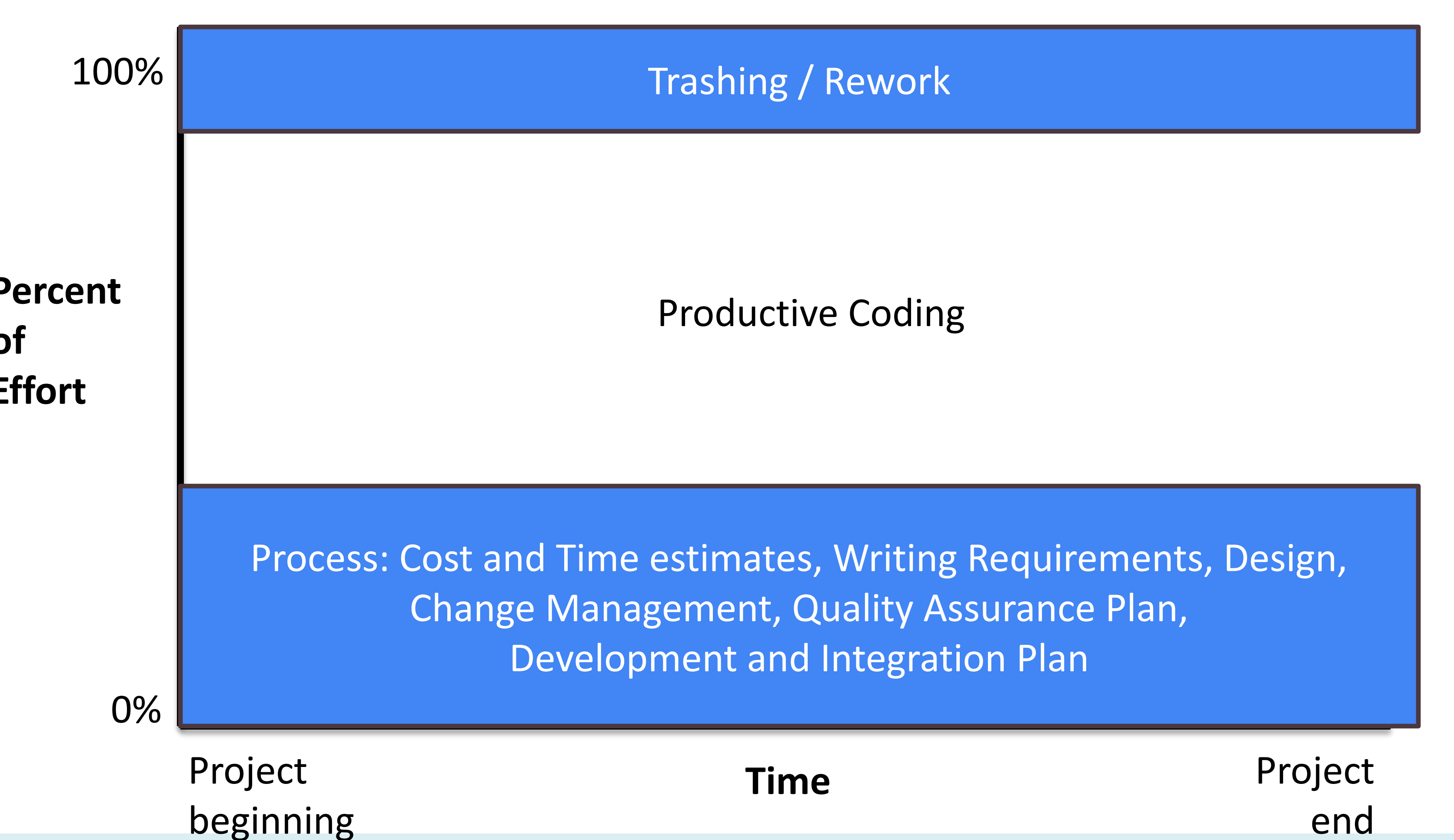
What makes a good process?

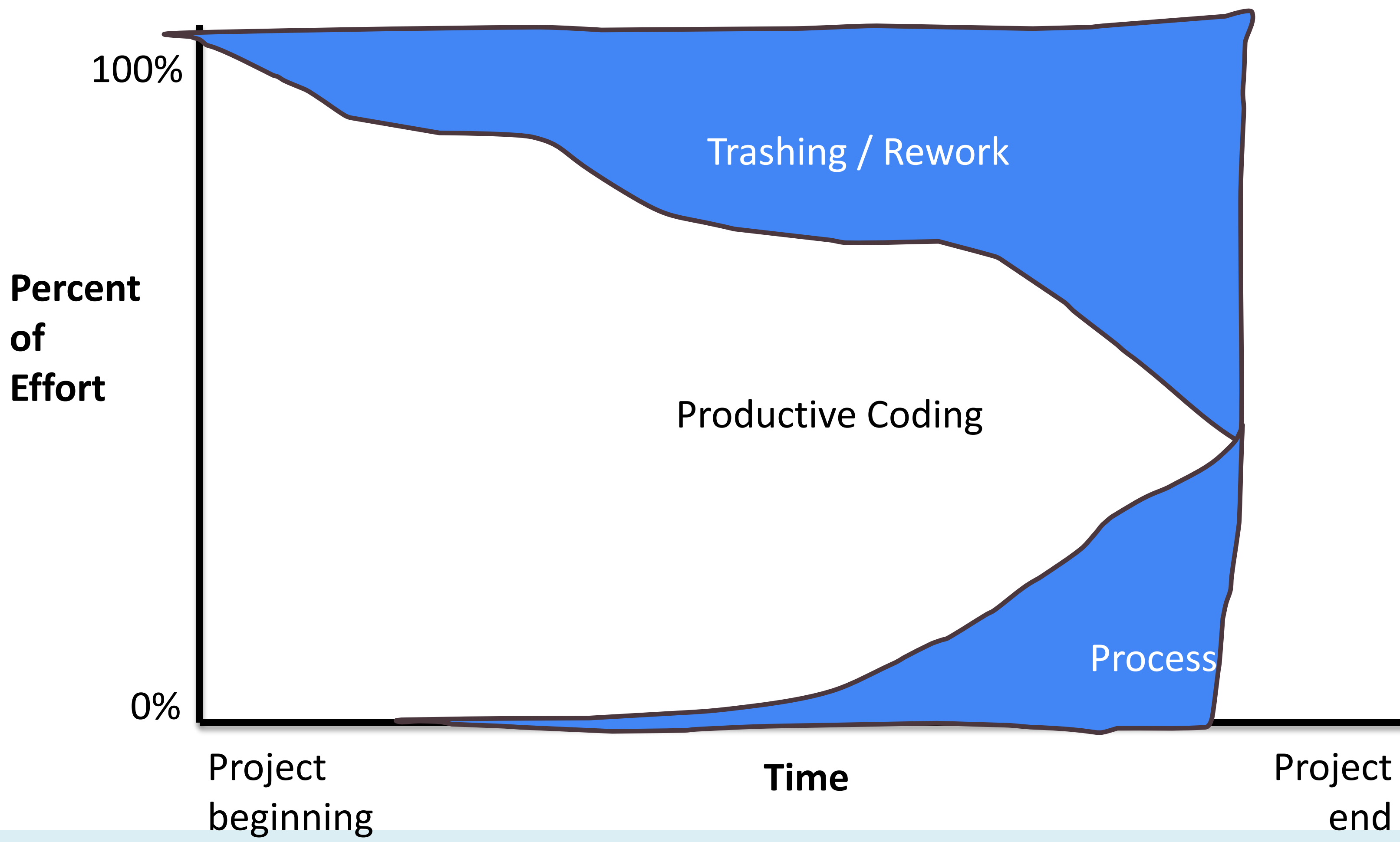
Sommerville, SE, ed. 8

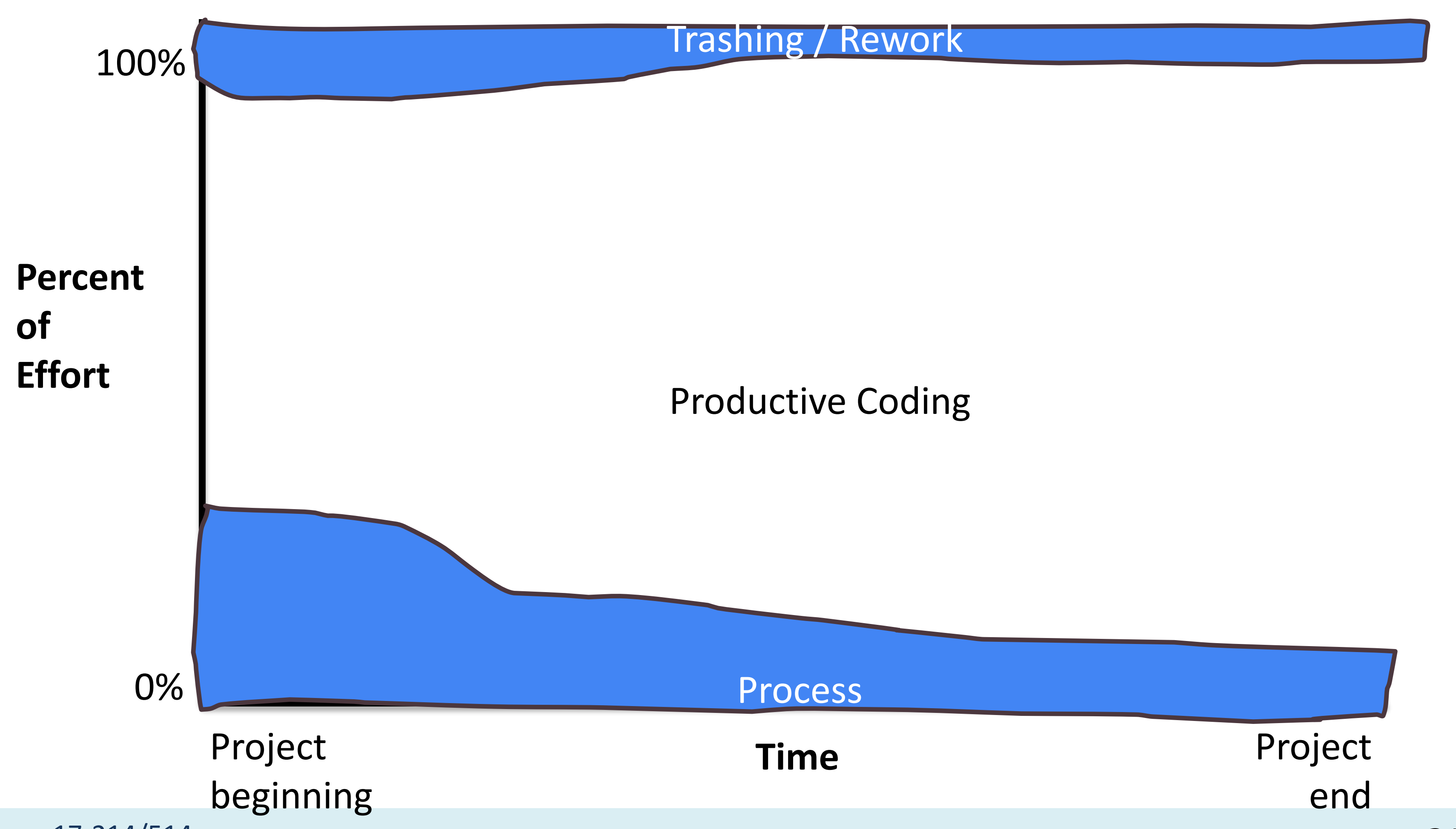


S3D


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

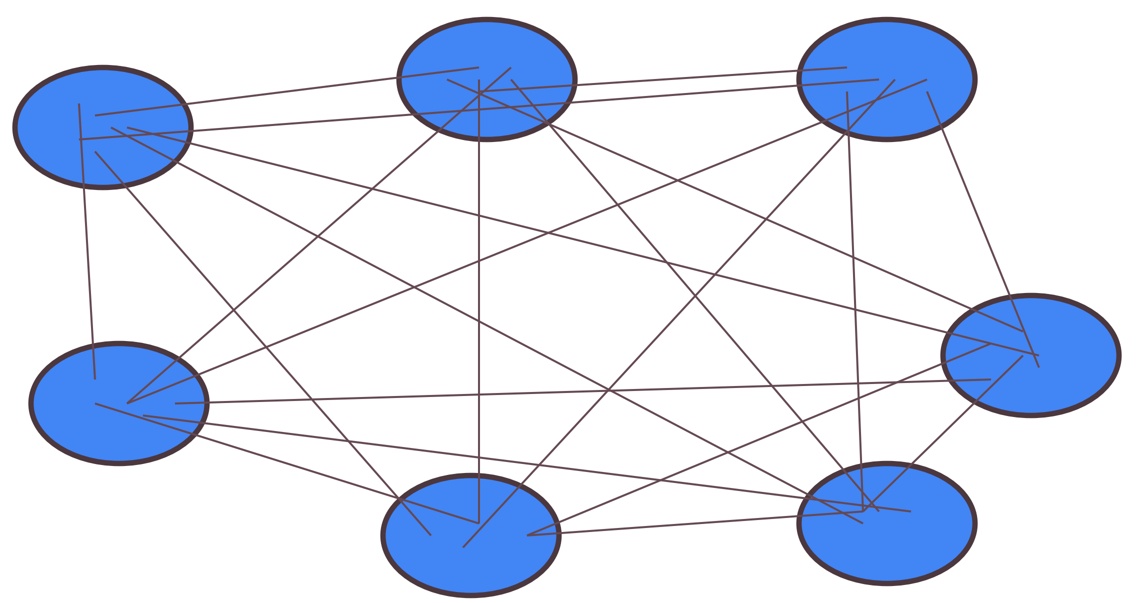

100%
Percent
of
Effort
Trashing / Rework
Productive Coding
Process
0%
Project
beginning
Time
Project
end


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

Example process issues

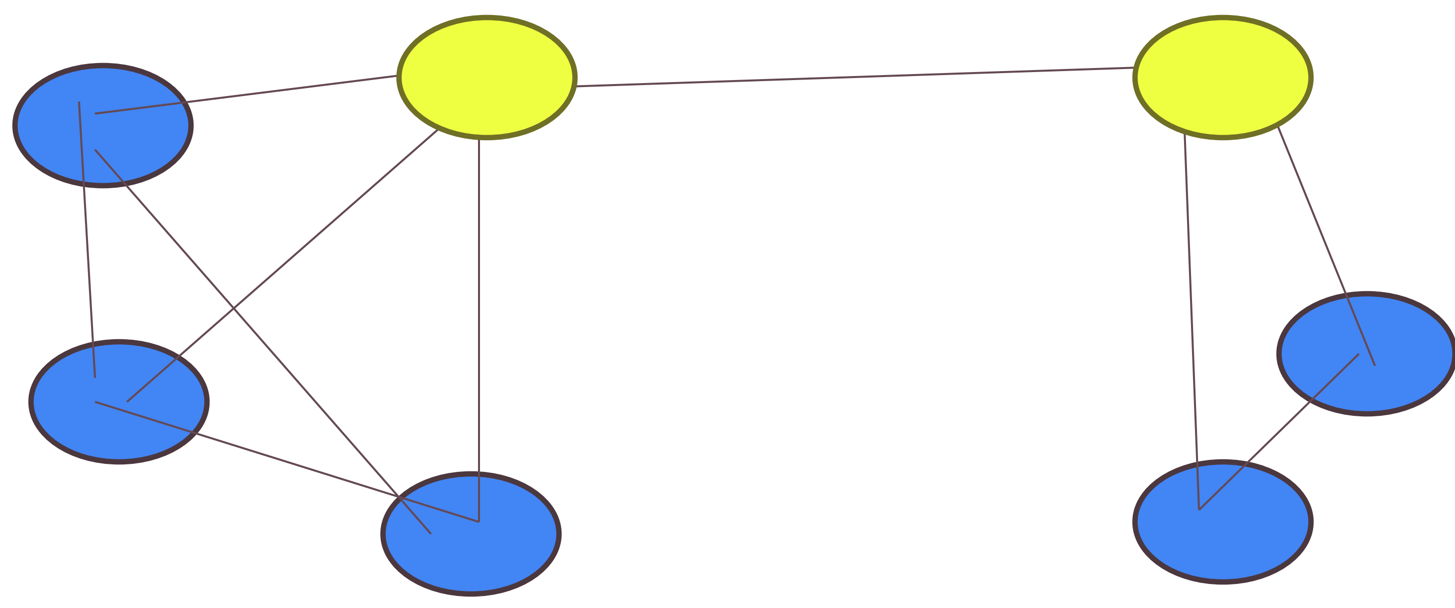
- Change Control: Mid-project informal agreement to changes suggested by customer or manager. Project scope expands 25-50%
- Quality Assurance: Late detection of requirements and design issues. Test-debug-reimplement cycle limits development of new features. Release with known defects.
- Defect Tracking: 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

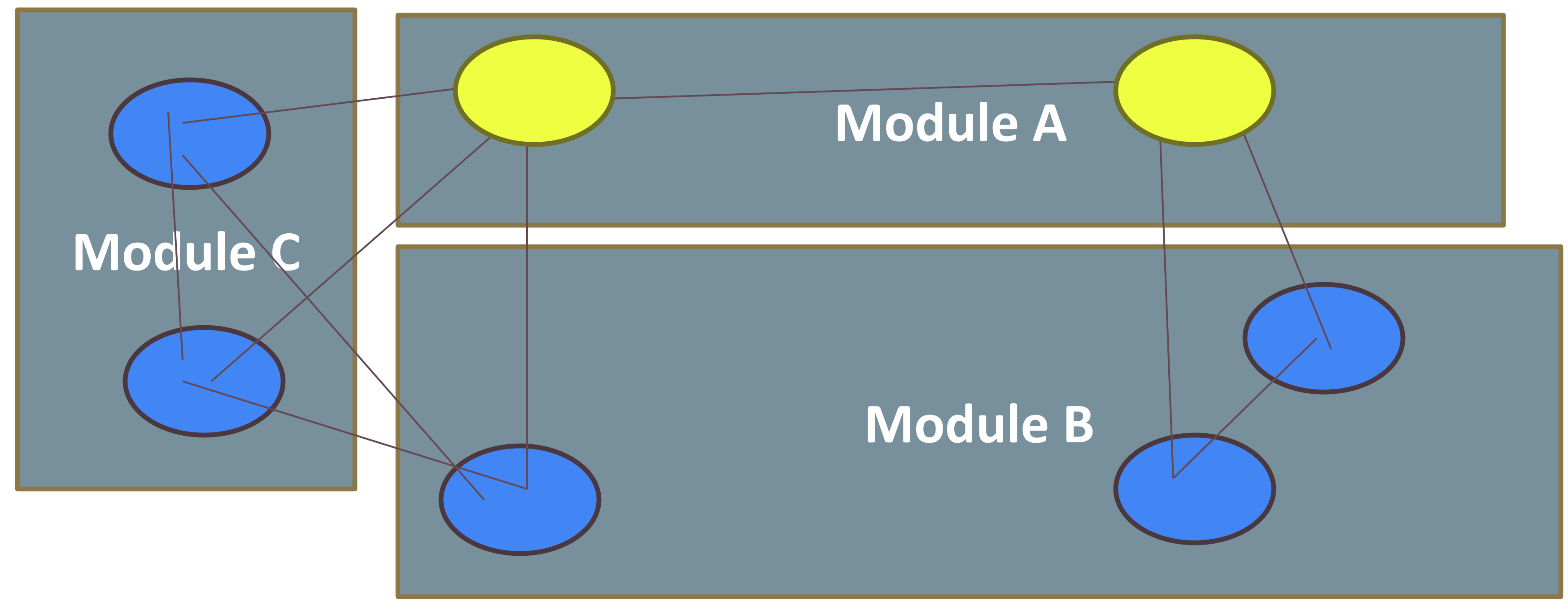
Conway’s Law

“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

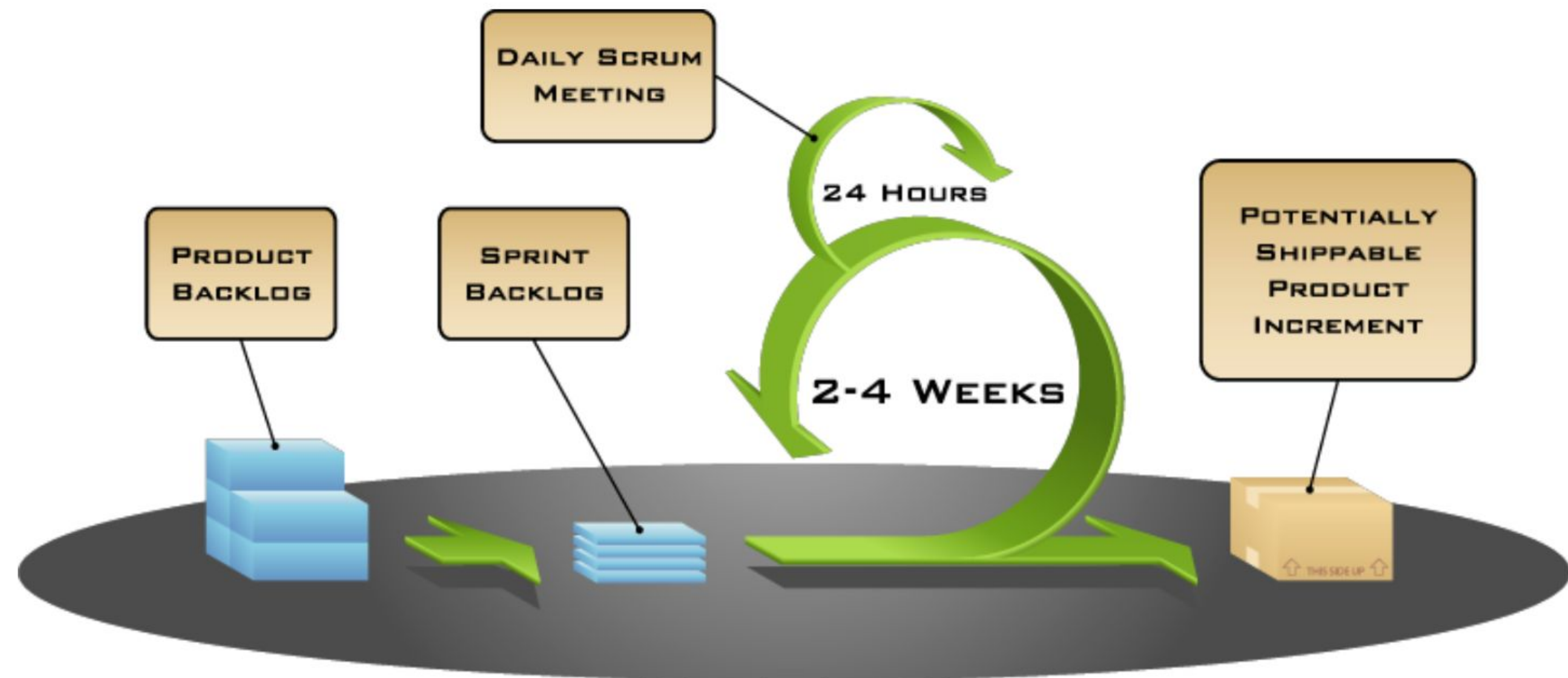


The Manifesto for Agile Software Development (2001)

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



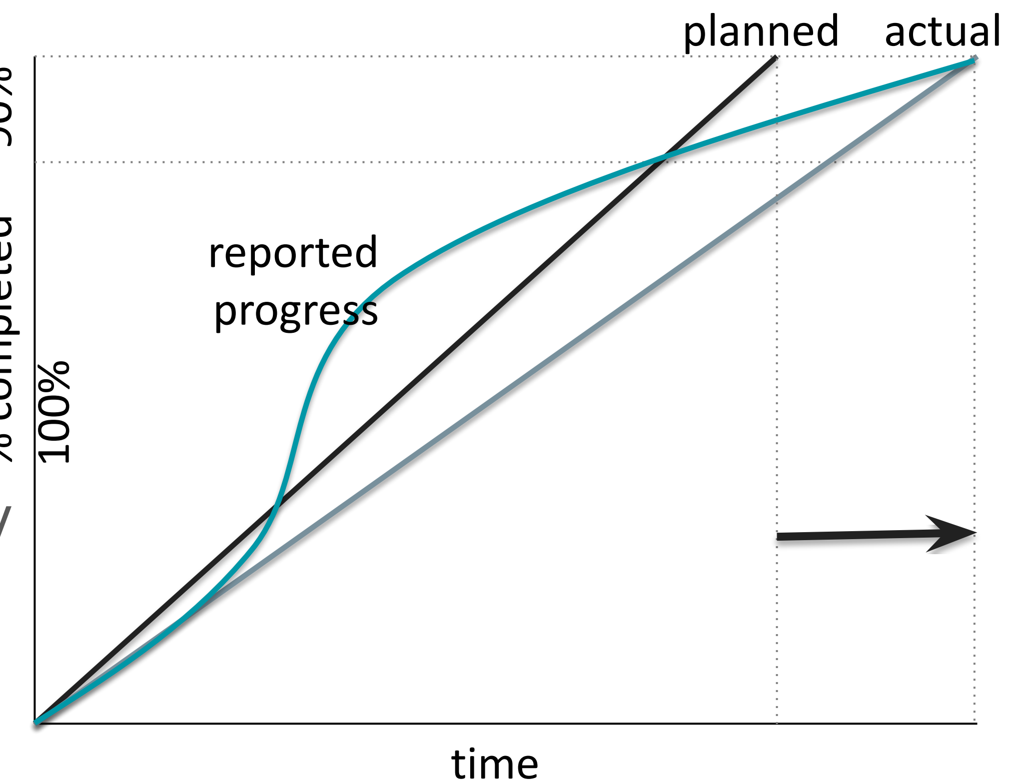
Planning

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

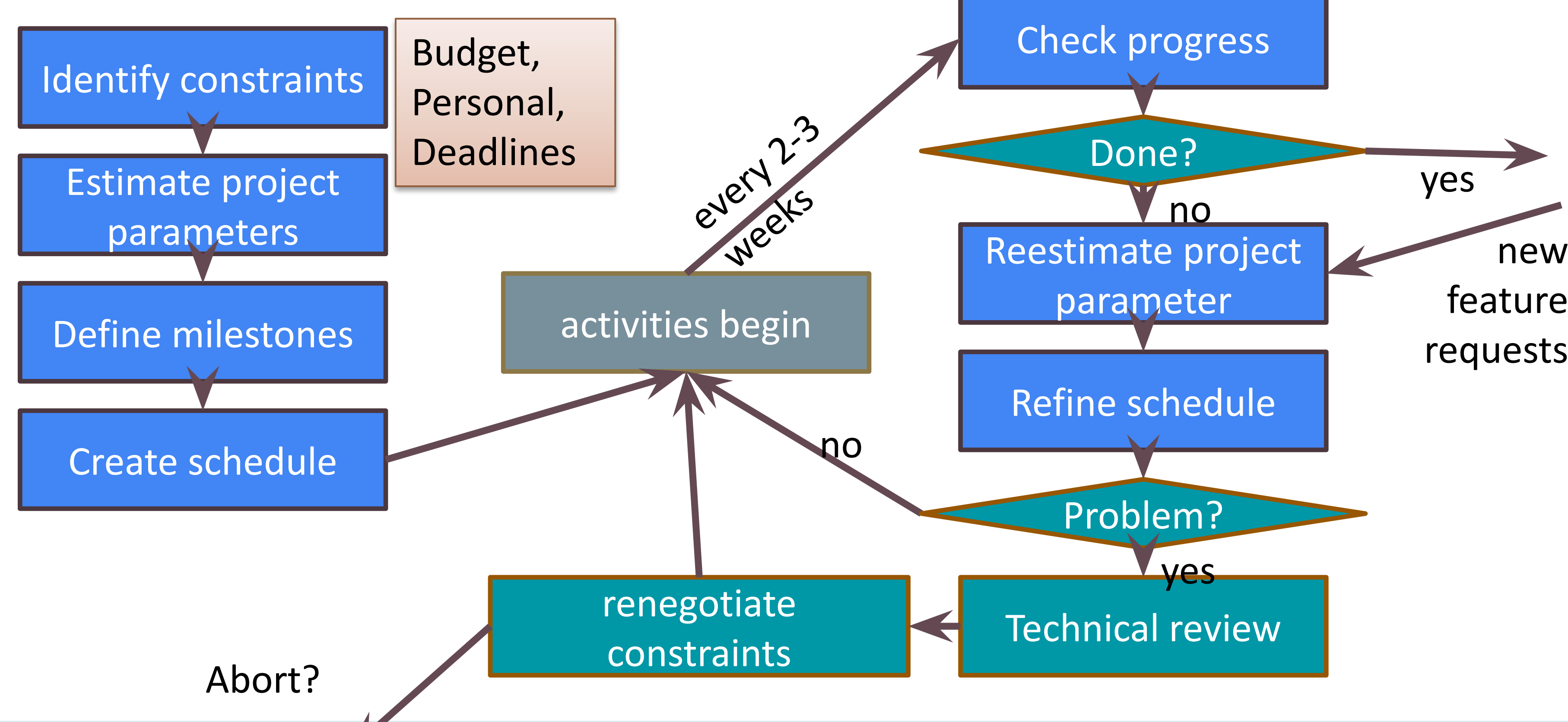
- Last 10% of work -> 40% of time (or 20/80)
- Make progress measureable
- Avoid depending entirely on developer estimations



Measuring Progress?

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

Project Planning

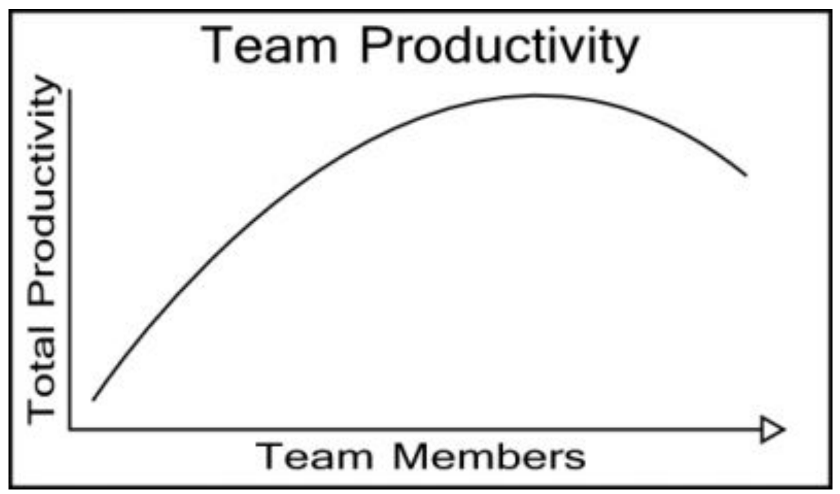


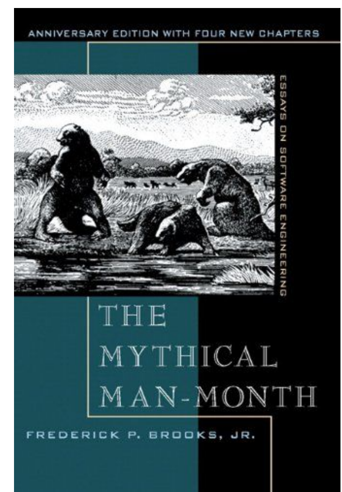
Reasons for Missed Deadlines

- 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

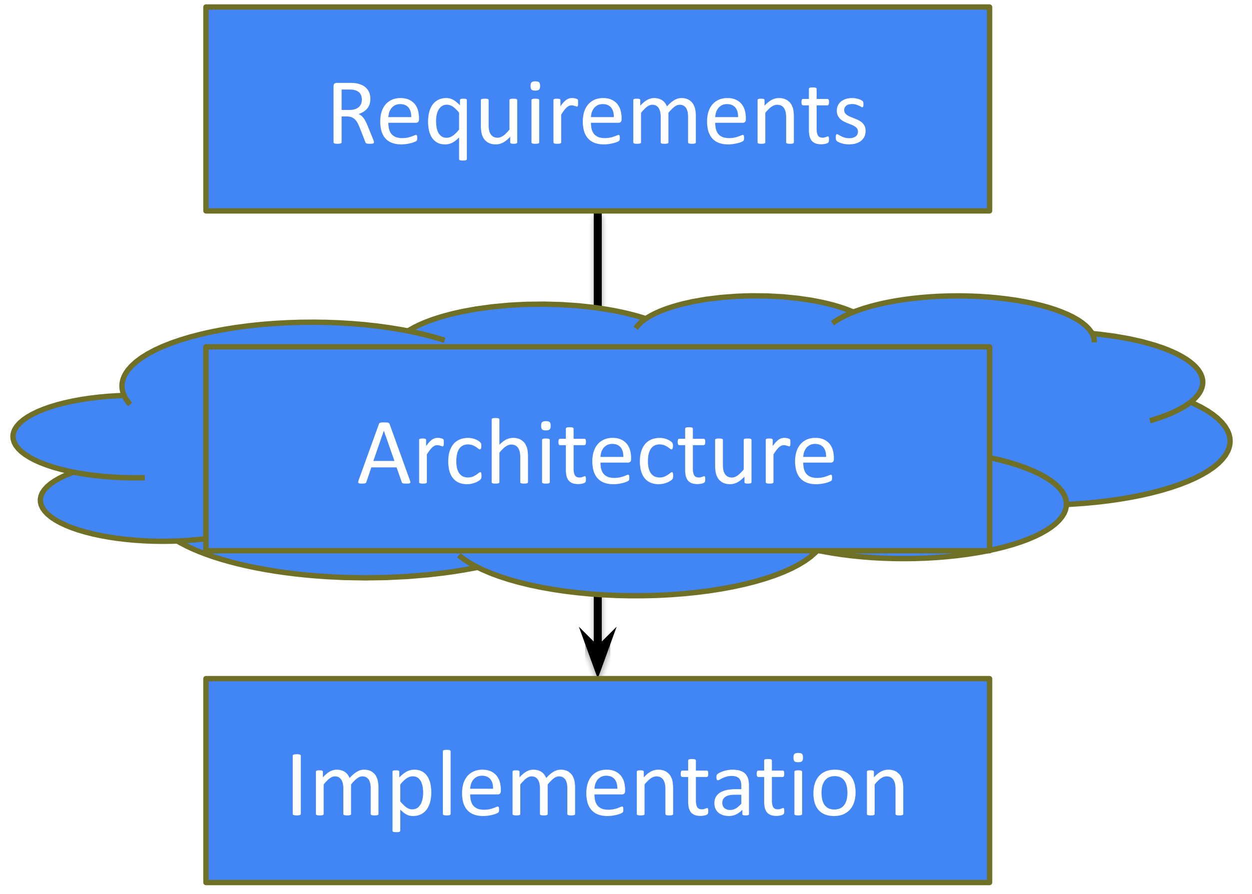
- Brook's law: Adding people to a late software project makes it later.





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

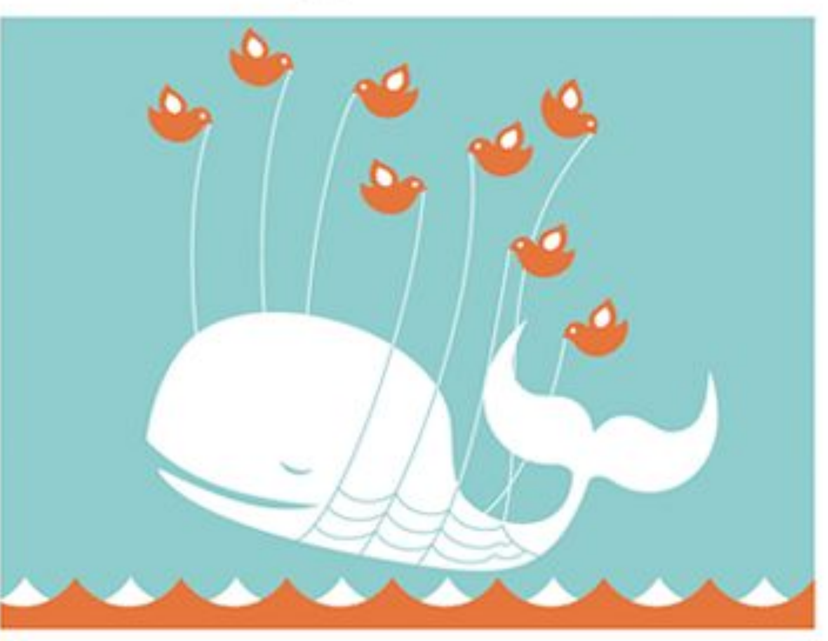
- How do I extend Eclipse with a plugin?
- What threads exist and how do they coordinate?
- How does Google scale to billions of hits per day?
- Where should I put my firewalls?
- What is the interface between subsystems?

Case Study:
Architecture Changes at Twitter

Home Public Timeline Help

Twitter is over capacity.

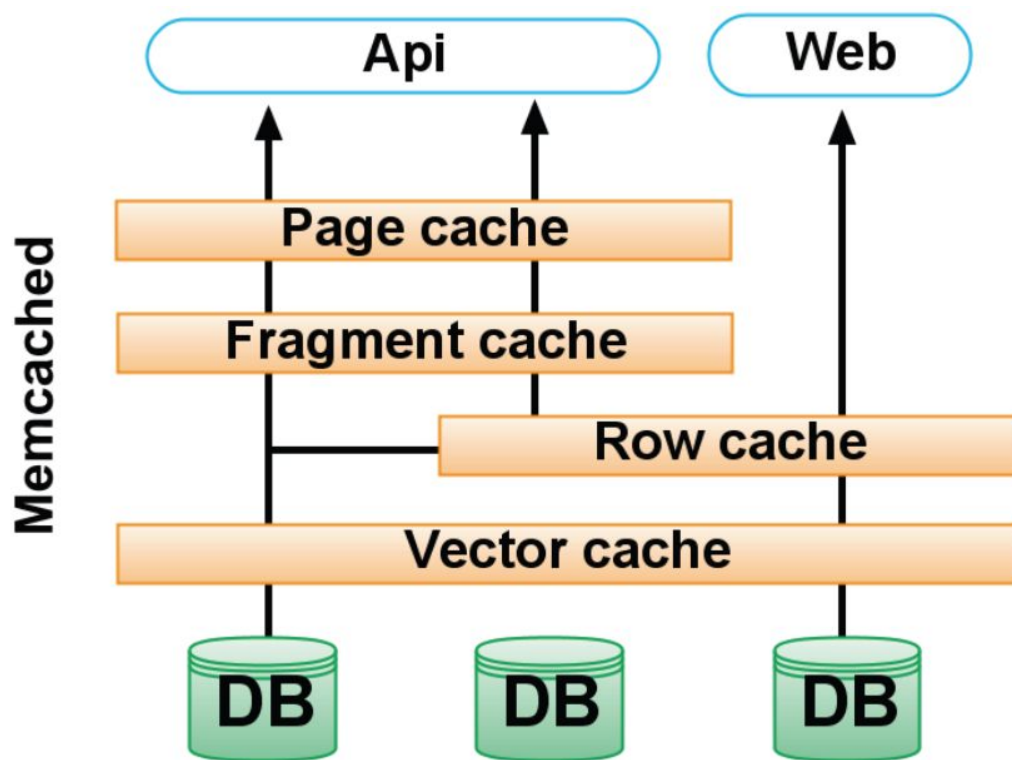
Too many tweets! Please wait a moment and try again.



© 2009 Twitter About Us Contact Blog Status API Help Jobs TOS Privacy

SOUTH
AFRICA
2010
FIFA
WORLD CUP

Caching



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."

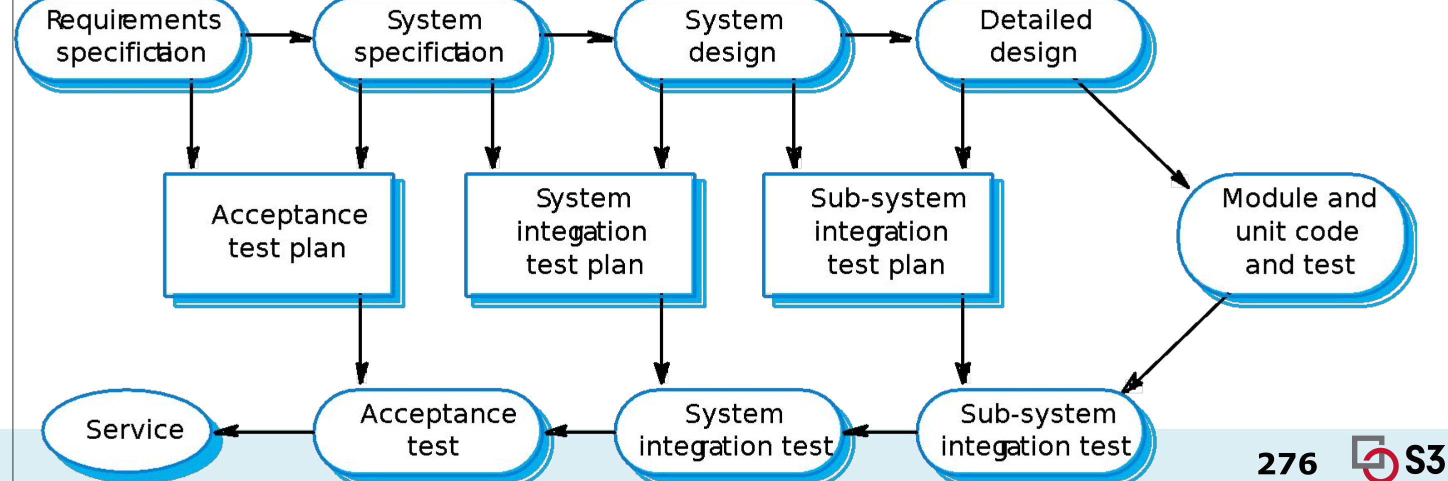
Was the original architect wrong?

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.”

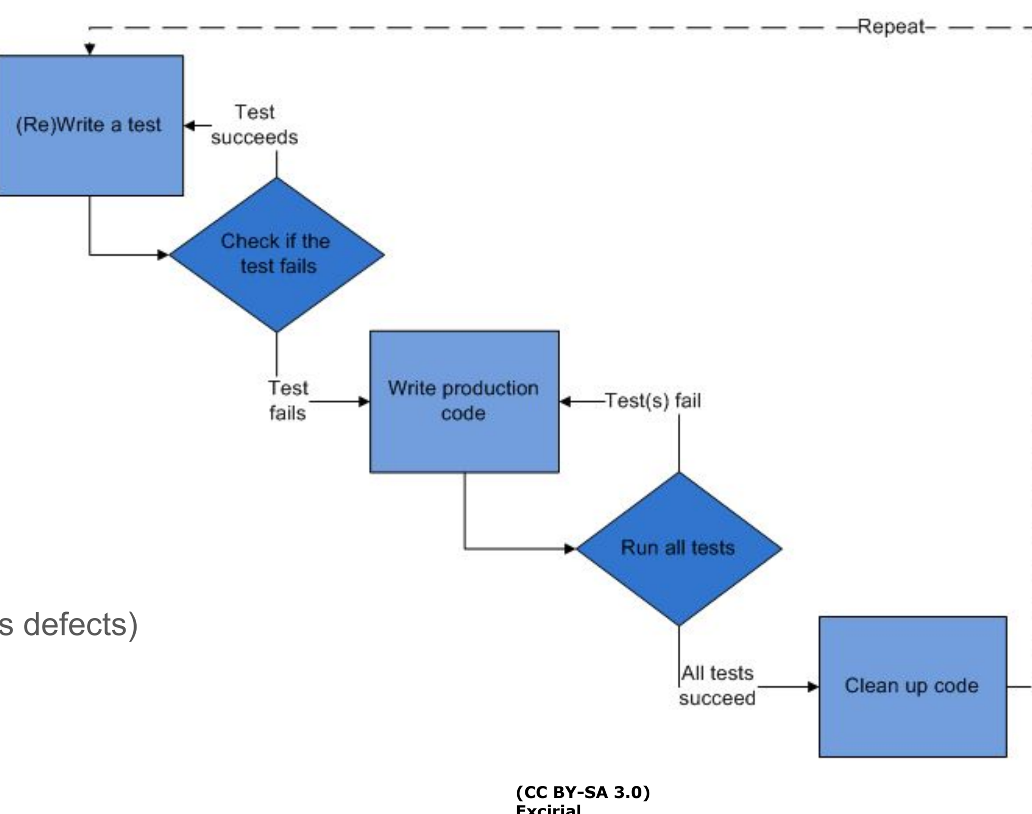
Time estimates (in hours):

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

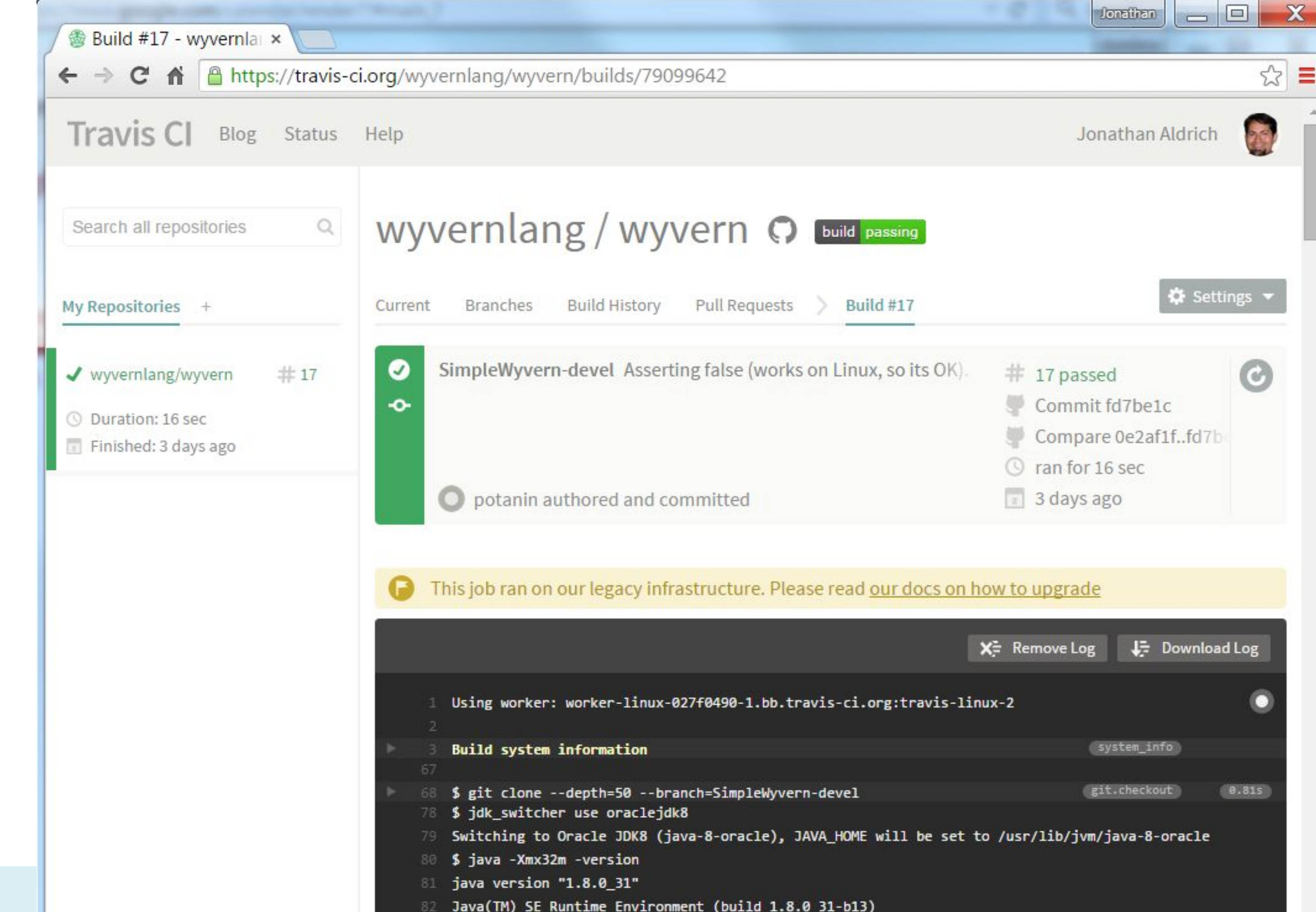
How to get developers to write tests?

Test Driven Development

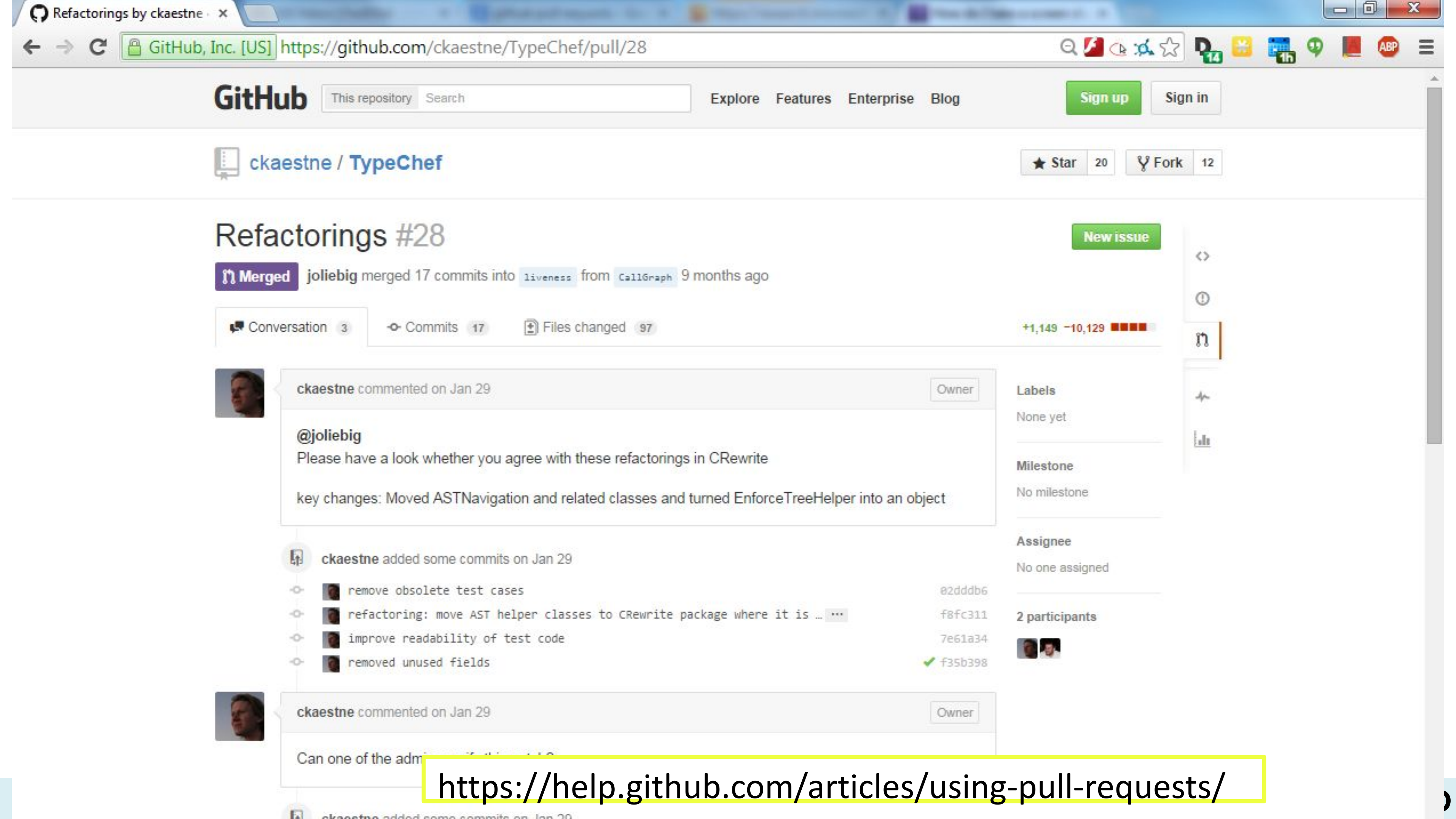
- Tests first!
- Popular agile technique
- Write tests as specifications before code
- Never write code without a failing test
- Claims:
  - Design approach toward testable design
  - Think about interfaces first
  - Avoid writing unneeded code
  - Higher product quality (e.g. better code, less defects)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

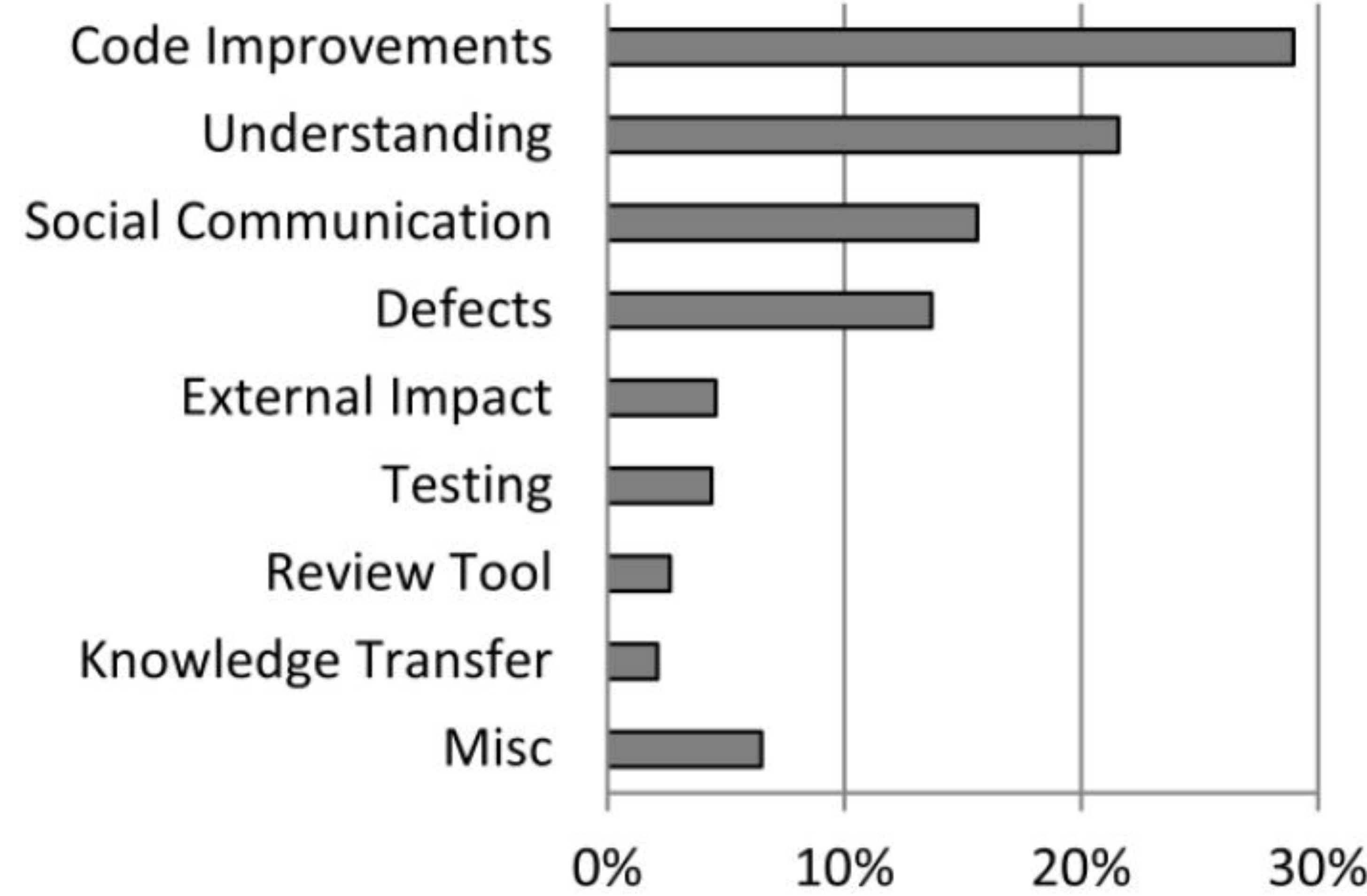


(CC BY-SA 3.0)
Excirial


Build #17 - wyvernla
https://travis-ci.org/wyvernlang/wyvern/builds/79099642
Travis CI
Blog
Status
Help
Jonathan Aldrich
Search all repositories
My Repositories
wyvernlang/wyvern
17
Duration: 16 sec
Finished: 3 days ago
wyvernlang / wyvern
build passing
Current
Branches
Build History
Pull Requests
Build #17
Settings
SimpleWyvern-devel Asserting false (works on Linux, so its OK).
17 passed
Commit fd7be1c
ran for 16 sec
3 days ago
potanin authored and committed
This job ran on our legacy infrastructure. Please read our docs on how to upgrade
Remove Log
Download Log
Using worker: worker-linux-027f0490-1.bb.travis-ci.org:travis-linux-2
Build system information
$ git clone --depth=50 --branch=SimpleWyvern-devel
$ jdk_switcher use oraclejdk8
Switching to Oracle JDK8 (java-8-oracle), JAVA_HOME will be set to /usr/lib/jvm/java-8-oracle
$ java -Xmx32m -version
java version "1.8.0_31"
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
Labels
None yet
Milestone
No milestone
Assignee
No one assigned
2 participants
ckaestne added some commits on Jan 29
remove obsolete test cases
e2dddb6
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
https://help.github.com/articles/using-pull-requests/



S3D

How to get developers to use static analysis?

```
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

Suggested fix attached: show

Not useful

```
  }

  public String getString() {
```

Are code reviews worth it?

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

There are many other courses in SE at CMU, consider taking them!