

# Foundations of Software Engineering

26 lectures in 30 min (245 slides)

# Learning Goals

- Broad scope of software engineering
- Importance of nontechnical issues
- Overview key challenges
  
- Syllabus, introduction and team forming

# Smoking Section



**2016** [ [edit](#) ]

This list is up to date as of June 30, 2016. Indicated changes in market value are relative to the previous quarter.

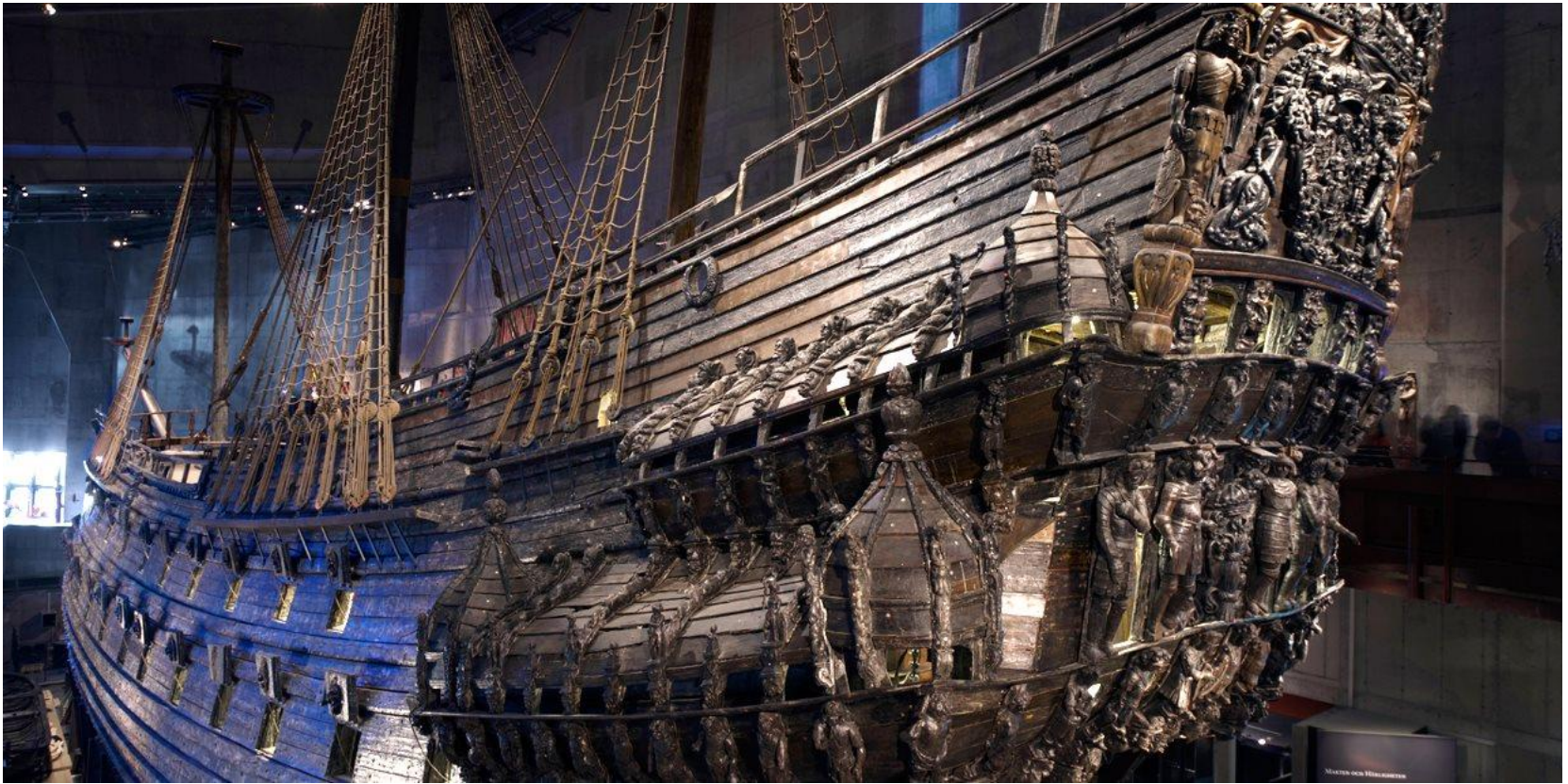
Rank	First quarter <sup>[8]</sup>		Second quarter		Third quarter		Fourth quarter	
1		Apple Inc ▲596,988.7		Apple Inc ▼515,590.0				
2		Alphabet ▼514,923.5		Alphabet ▼475,160.0				
3		Microsoft ▼434,130.1		Microsoft ▼399,710.0				
4		Amazon Inc. ▲356,119.4		Exxon Mobil ▲388,710.0				
5		Berkshire Hathaway ▲349,813.4		Berkshire Hathaway ▲356,810.0				
6		Exxon Mobil ▲346,616.5		Amazon Inc. ▼337,650.0				
7		Facebook ▲326,357.8		Johnson & Johnson ▲333,650.0				
8		Johnson & Johnson ▲300,604.4		Facebook ▲326,880.0				
9		General Electric ▼295,545.7		General Electric ▼289,480.0				
10		Wells Fargo ▼246,035.0		Wells Fargo ▼238,950.0				

003/45/7844



ISAT GeoStar 45  
23:15 EST 14 Aug. 2003

# Vasa



# Case Study 1: PeopleCars

- Scenario and question from prior final
- Read scenario and question
- Discuss answers with your neighbors
  
- Keep answers until last lecture

# Foundations of Software Engineering

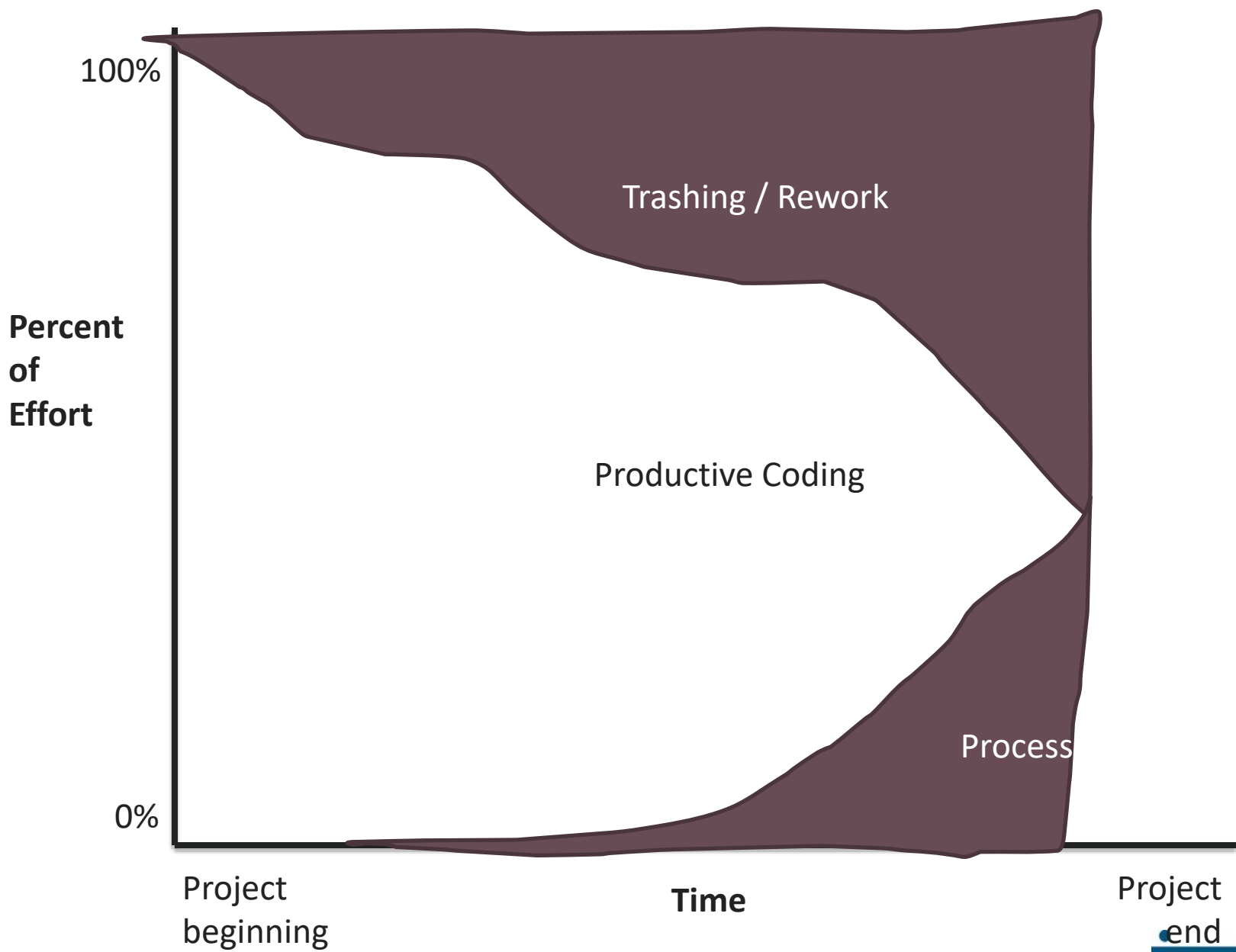
Part 2: Quick intro to process, teamwork,  
risk and scheduling

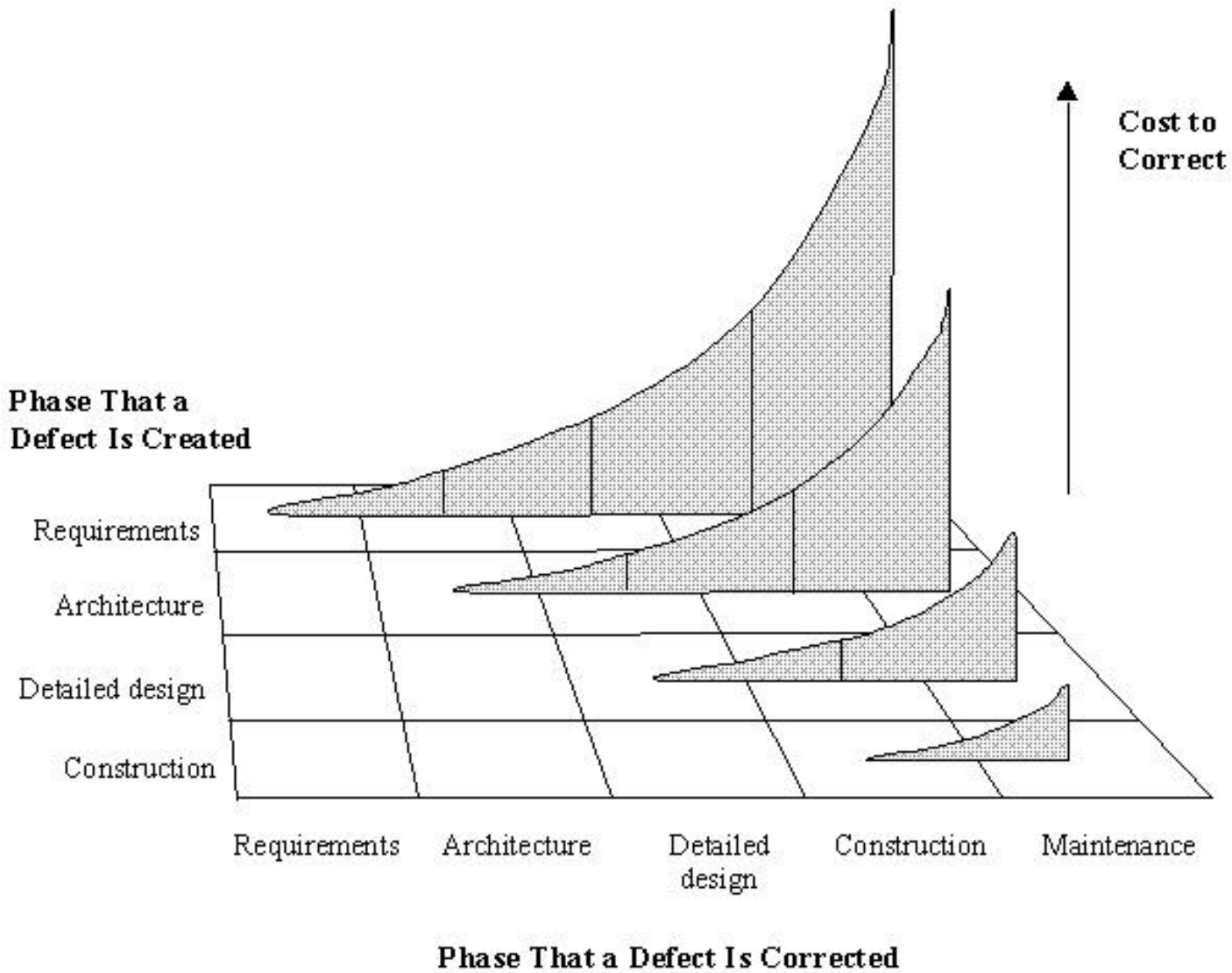
Christian Kästner



# Learning Goals

- Recognize the Importance of process
- Understand the difficulty of measuring progress
- Identify what why software development has project characteristics
- Use milestones for planning and progress measurement
- Ability to divide work and planning and replan it
- Model dependencies and schedule work with network plans and Gantt diagrams
- Identifying and managing risks





Copyright 1998 Steven C. McConnell. Reprinted with permission from *Software Project Survival Guide* (Microsoft Press, 1998).

π

# Accepting and Coping with Risks

- Selectively innovate to increase value
- Improve capability and competitiveness
  
- Focus risk where it is needed
- Rely on precedent and convention (experience)
- Use iteration and feedback
  - prototypes, spiral development, sprints

You are here: [Home](#) > [Shop](#)



Big  
75mm 67 grams

~~6.00 EUR~~

[Buy](#) [Detail](#)



Small  
60mm 53 grams

~~4.00 EUR~~

[Buy](#) [Detail](#)



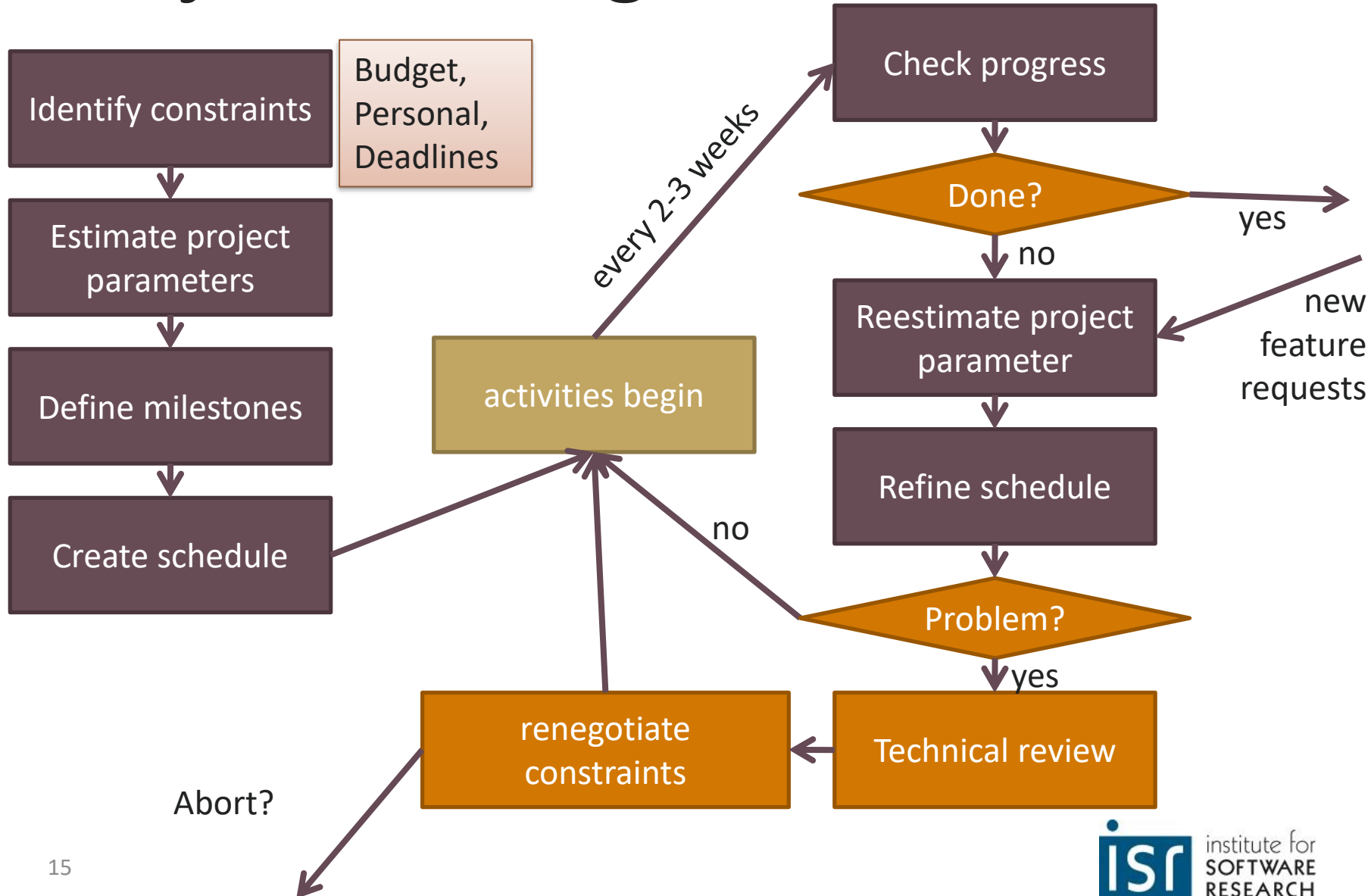
Viking Norwik

75mm 134g-402g | 134g 7€ | 201g 8€ | 268g 9€ | 335g 10€ | 402g 11€ |

~~8.00 EUR~~

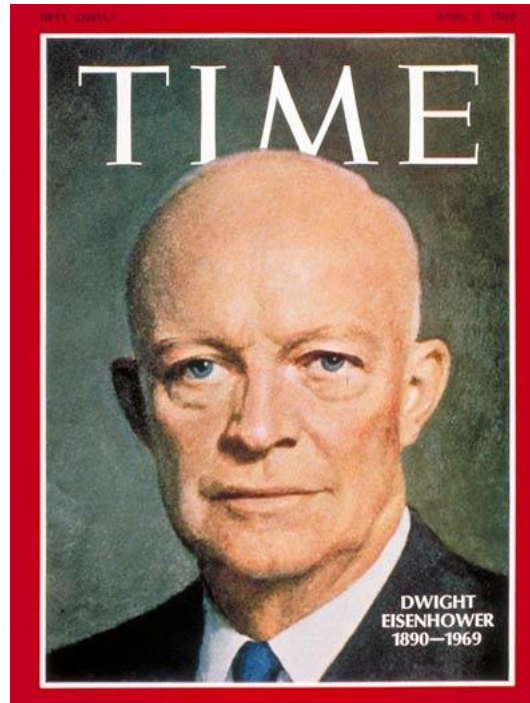
[Buy](#) [Detail](#)

# Project Planning



# Planning

- **Plans are worthless, but planning is everything.**





# HW1: Planning and Building PhD Application System

# Foundations of Software Engineering

Part 3: Healthcare.gov

Michael Hilton

# Foundations of Software Engineering

Lecture 5: Requirements are hard

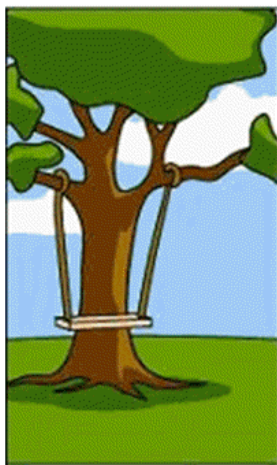
Michael Hilton

# Learning goals

- Explain the importance and challenges of requirements in software engineering.
- Explain how and why requirements articulate the relationship between a desired system and its environment. Identify assumptions.
- Distinguish between and give examples of: functional and quality requirements; informal statements and verifiable requirements.
- State quality requirements in measurable ways



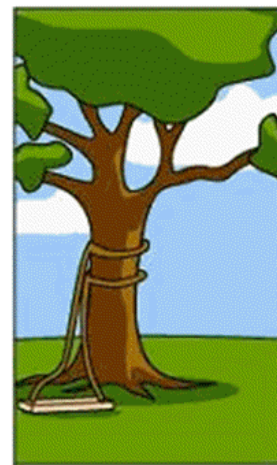
How the customer explained it



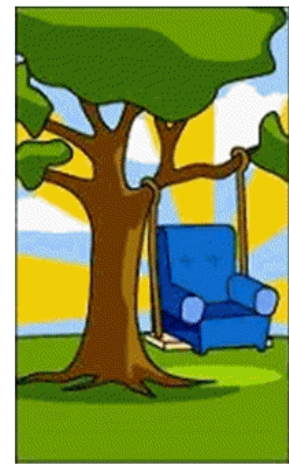
How the project leader understood it



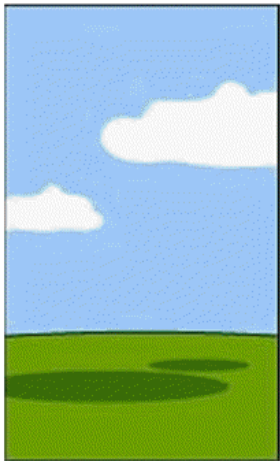
How the engineer designed it



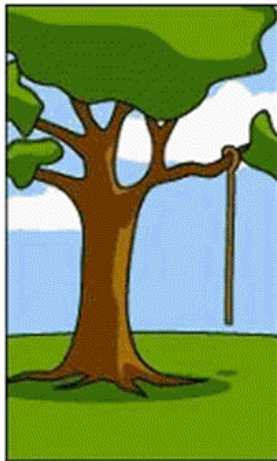
How the programmer wrote it



How the sales executive described it



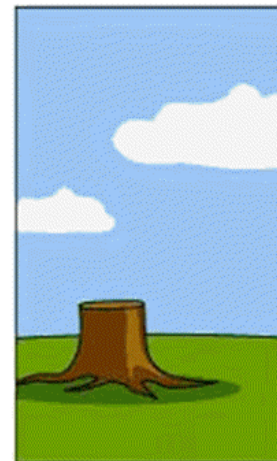
How the project was documented



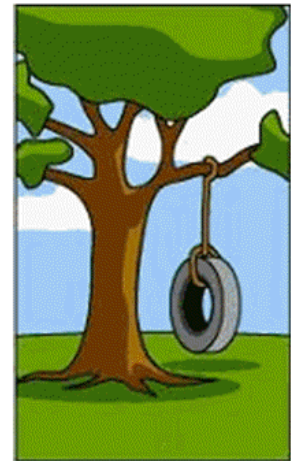
What operations installed



How the customer was billed



How the help desk supported it



What the customer really needed

# Communication problem

Goal: figure out what should be built.

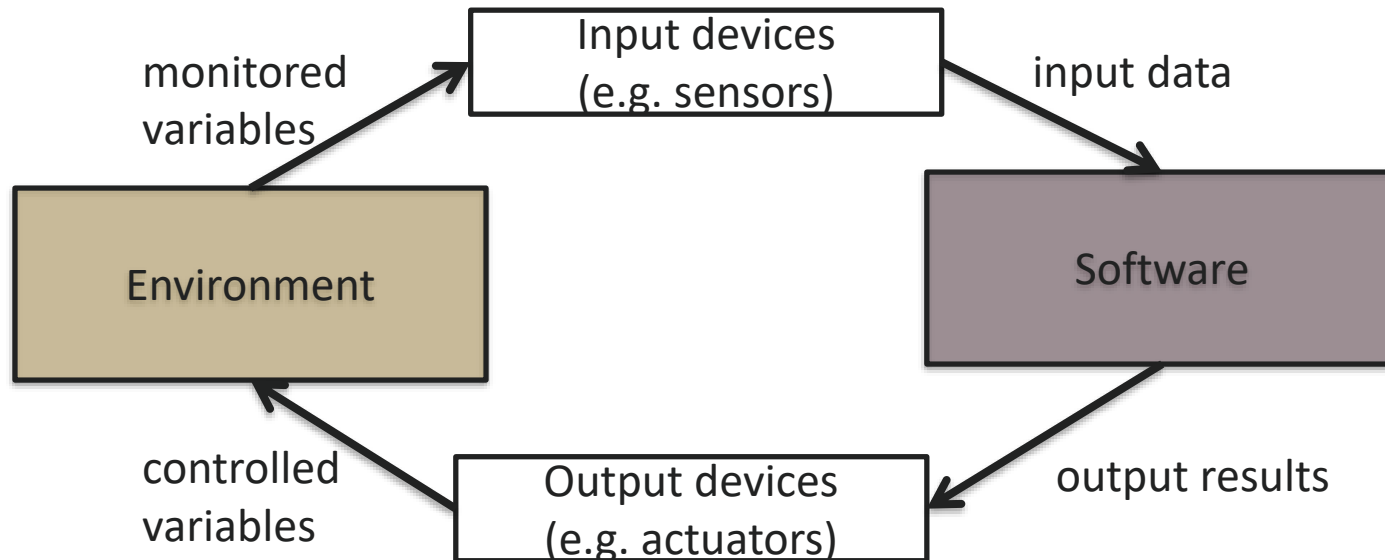
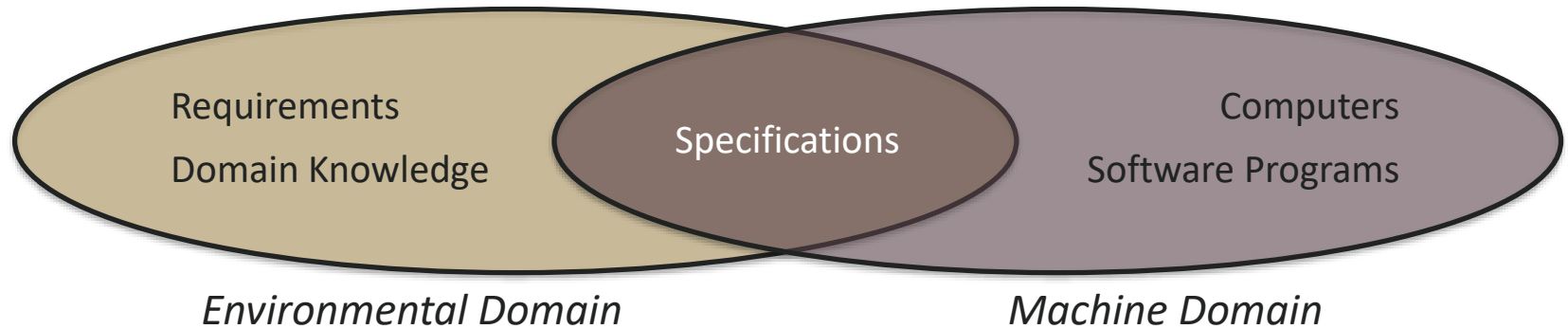
Express those ideas so that the correct thing is built.



# Four Kinds of Denial

- **Denial by prior knowledge** – we have done this before, so we know **what** is required
- **Denial by hacking** – our fascination with machines dominates our focus on the **how**
- **Denial by abstraction** – we pursue elegant models which obscure, remove or downplay the real world
- **Denial by vagueness** – imply (vaguely) that machine descriptions are actually those of the world

# Environment and the Machine



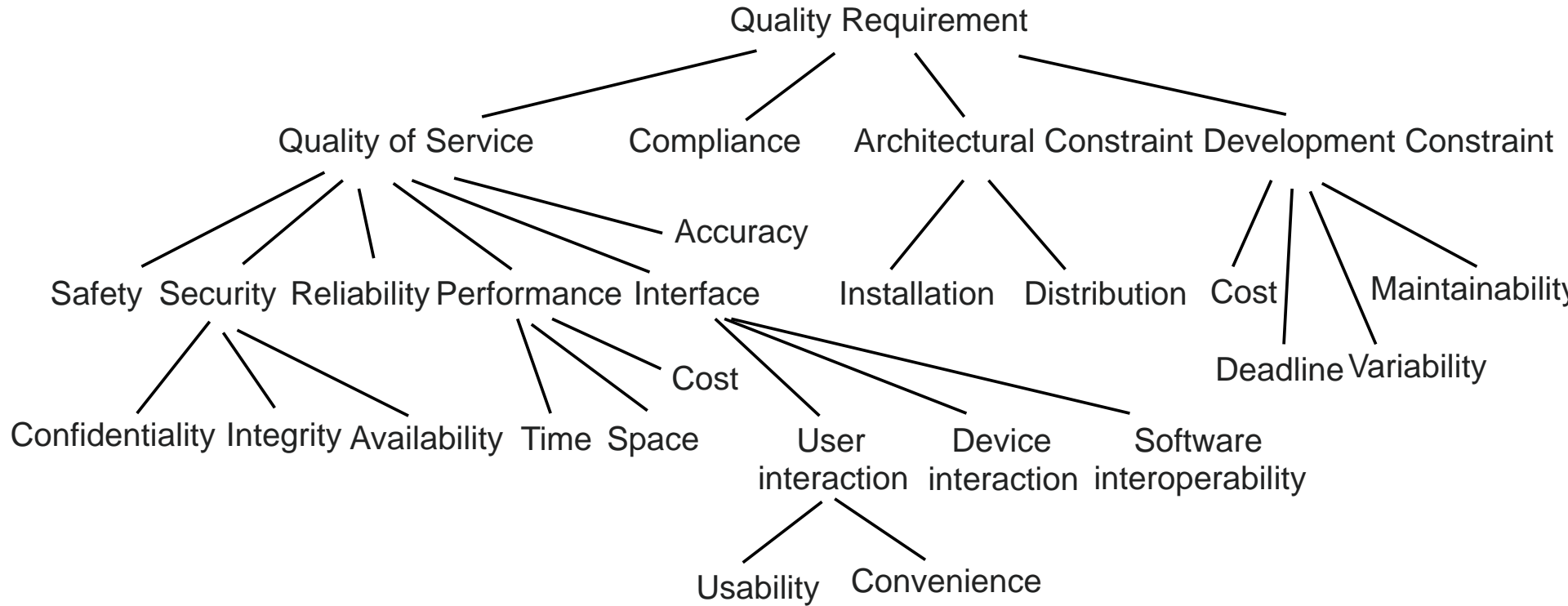


# Lufthansa Flight 2904



# Avoiding implementation bias

- Requirements describe what is observable at the environment-machine interface.
- *Indicative mood* describes the environment (as-is)
- *Optative mood* to describe the environment with the machine (to-be).



# Foundations of Software Engineering

Lecture 5: Measurement

Christian Kaestner

# Learning Goals

- Use measurements as a decision tool to reduce uncertainty
- Understand difficulty of measurement; discuss validity of measurements
- Examples of metrics for software qualities and process
- Understand limitations and dangers of decisions and incentives based on measurements



# Visual Studio since 2007

“Maintainability Index calculates an index value between 0 and 100 that represents the relative ease of maintaining the code. A **high value means better maintainability**. Color coded ratings can be used to quickly identify trouble spots in your code. A **green rating is between 20 and 100** and indicates that the code has good maintainability. A **yellow rating is between 10 and 19** and indicates that the code is moderately maintainable. A **red rating is a rating between 0 and 9** and indicates low maintainability.”

Hierarchy	Maintainability Index	Cyclomatic Complexity	Class Coupling	Depth of Inheritance	Lines of Code
- checkopenfile.exe	74	10	19	7	39
{ } checkopenfile	74	10	19	7	39
Form1	67	9	16	7	36
Program	81	1	3	1	3

# The Index

Maintainability Index =

$\text{MAX}(0, (171 -$

$5.2 * \log(\text{Halstead Volume}) -$

$0.23 * (\text{Cyclomatic Complexity}) -$

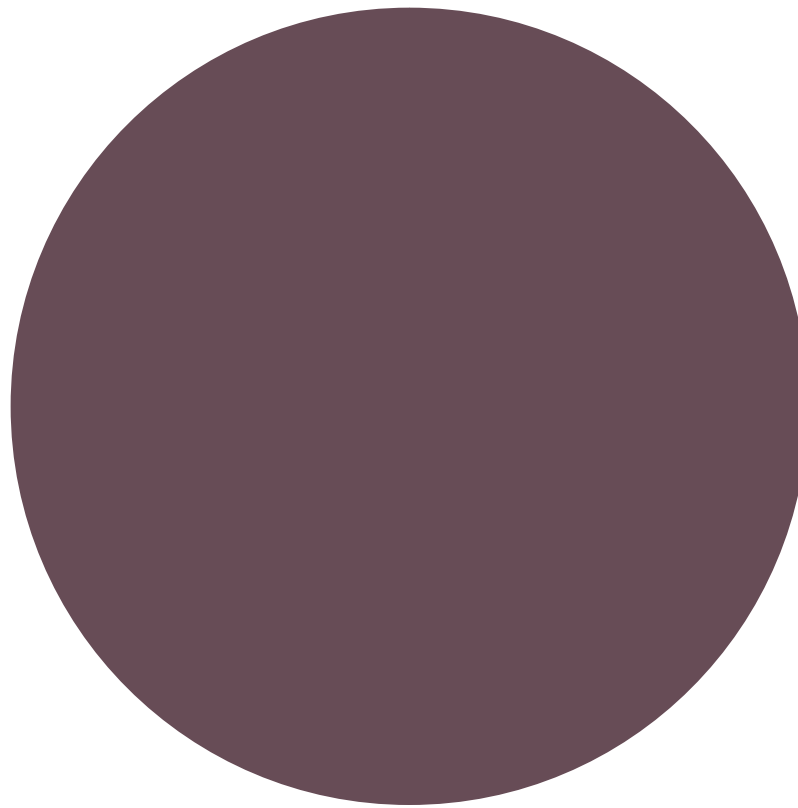
$16.2 * \log(\text{Lines of Code})$

$)*100 / 171)$



# About You

Could explain Cyclomatic Complexity



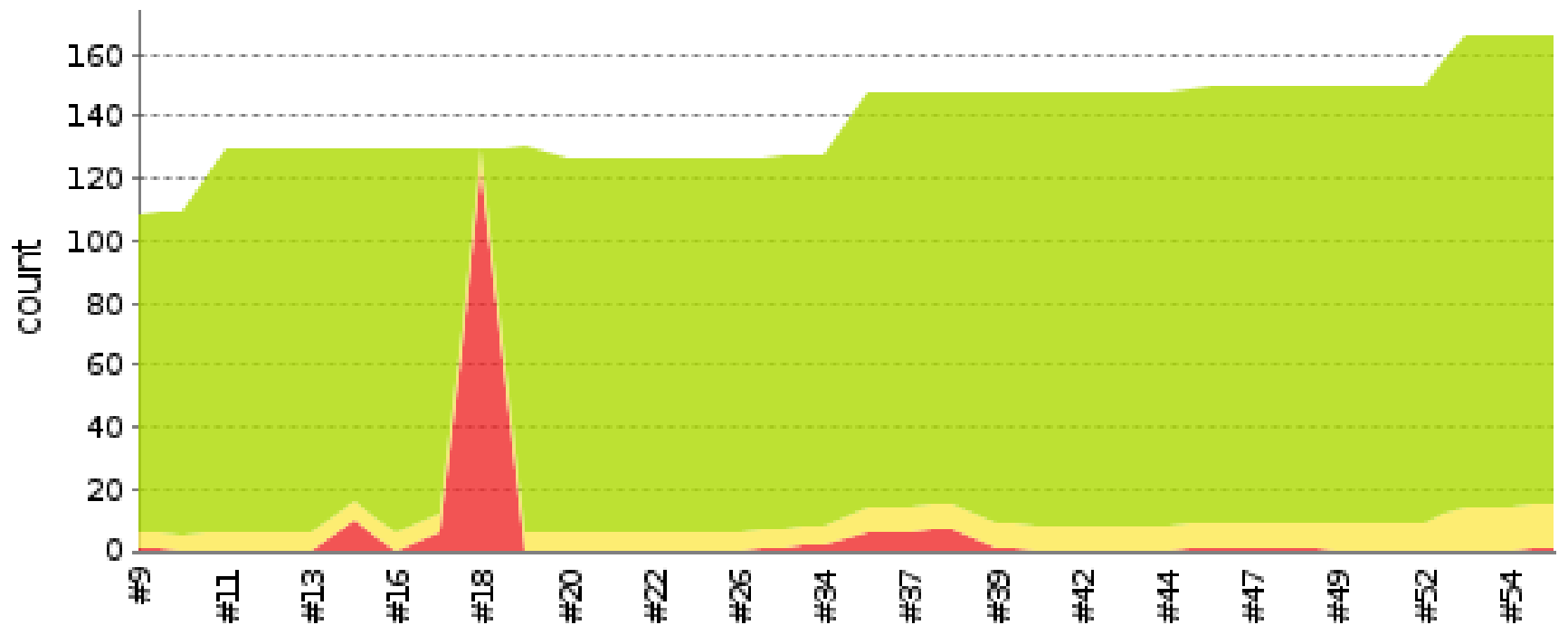
- No
- Vaguely
- Yes

# Everything is measurable

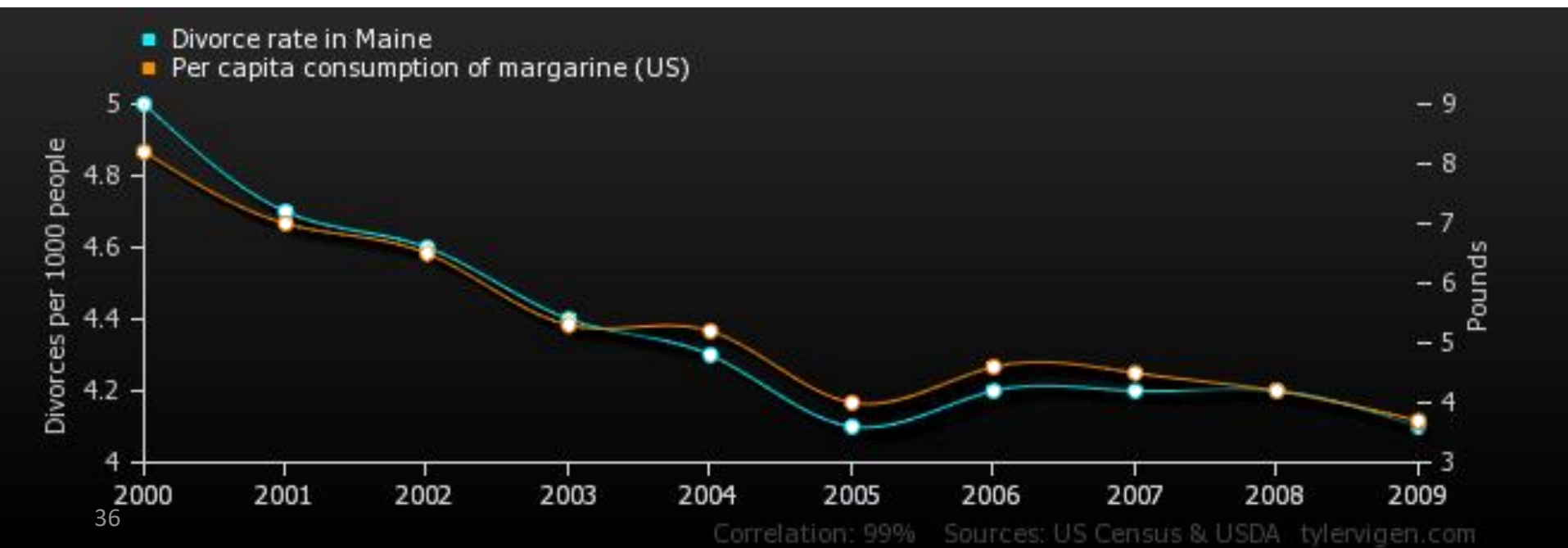
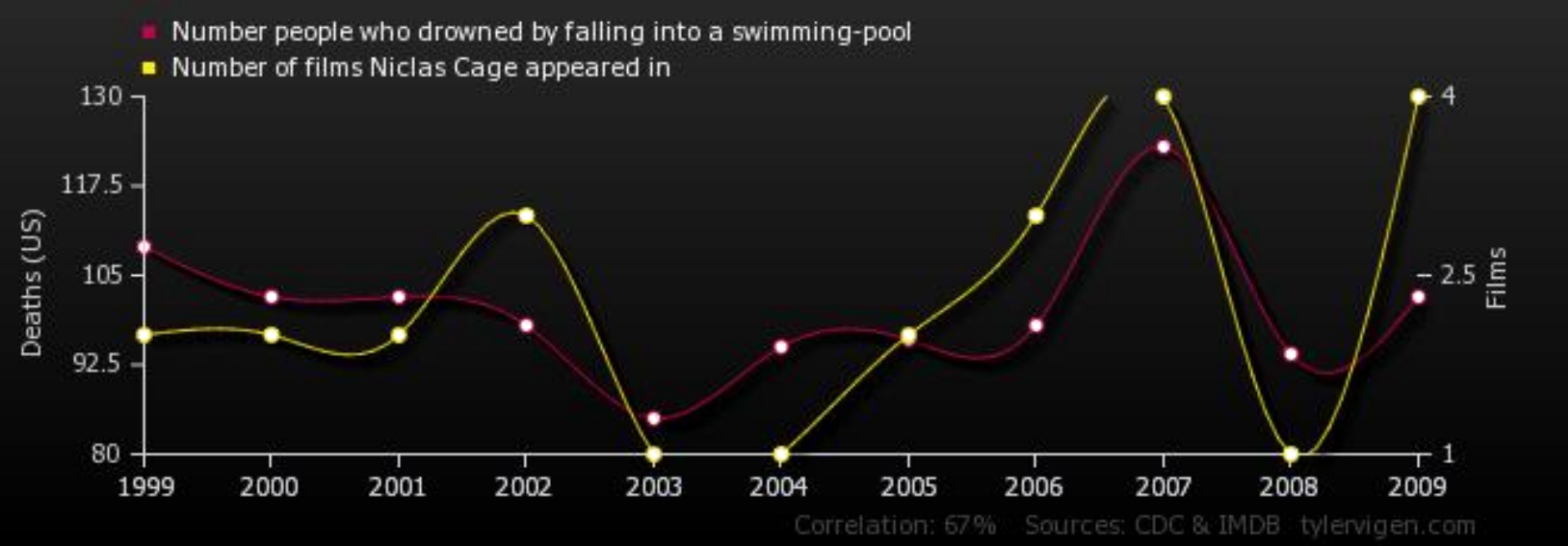
1. If X is something we care about, then X, by definition, must be detectable.
  - How could we care about things like “quality,” “risk,” “security,” or “public image” if these things were totally undetectable, directly or indirectly?
  - If we have reason to care about some unknown quantity, it is because we think it corresponds to desirable or undesirable results in some way.
2. If X is detectable, then it must be detectable in some amount.
  - If you can observe a thing at all, you can observe more of it or less of it
3. If we can observe it in some amount, then it must be measurable.

# Trend analyses

## Test Result Trend



[\(just show failures\)](#) [enlarge](#)

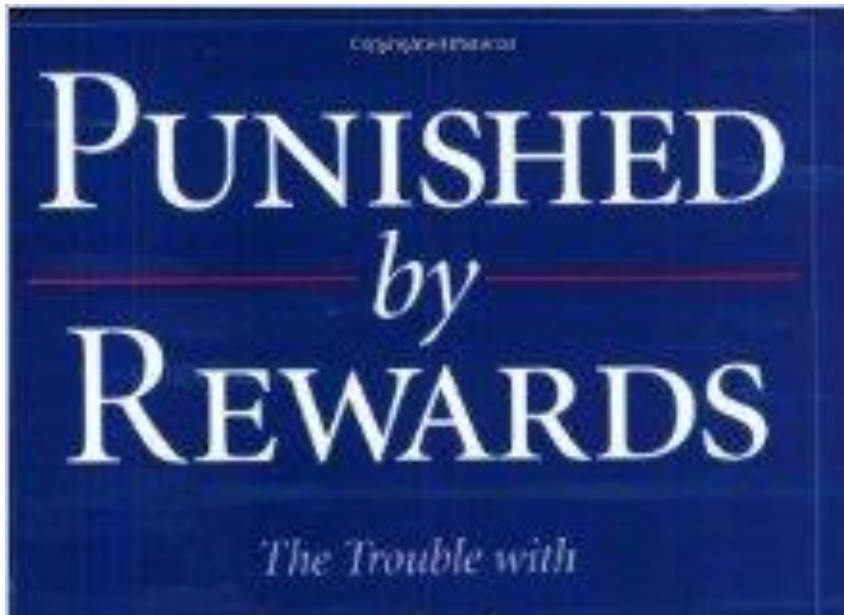




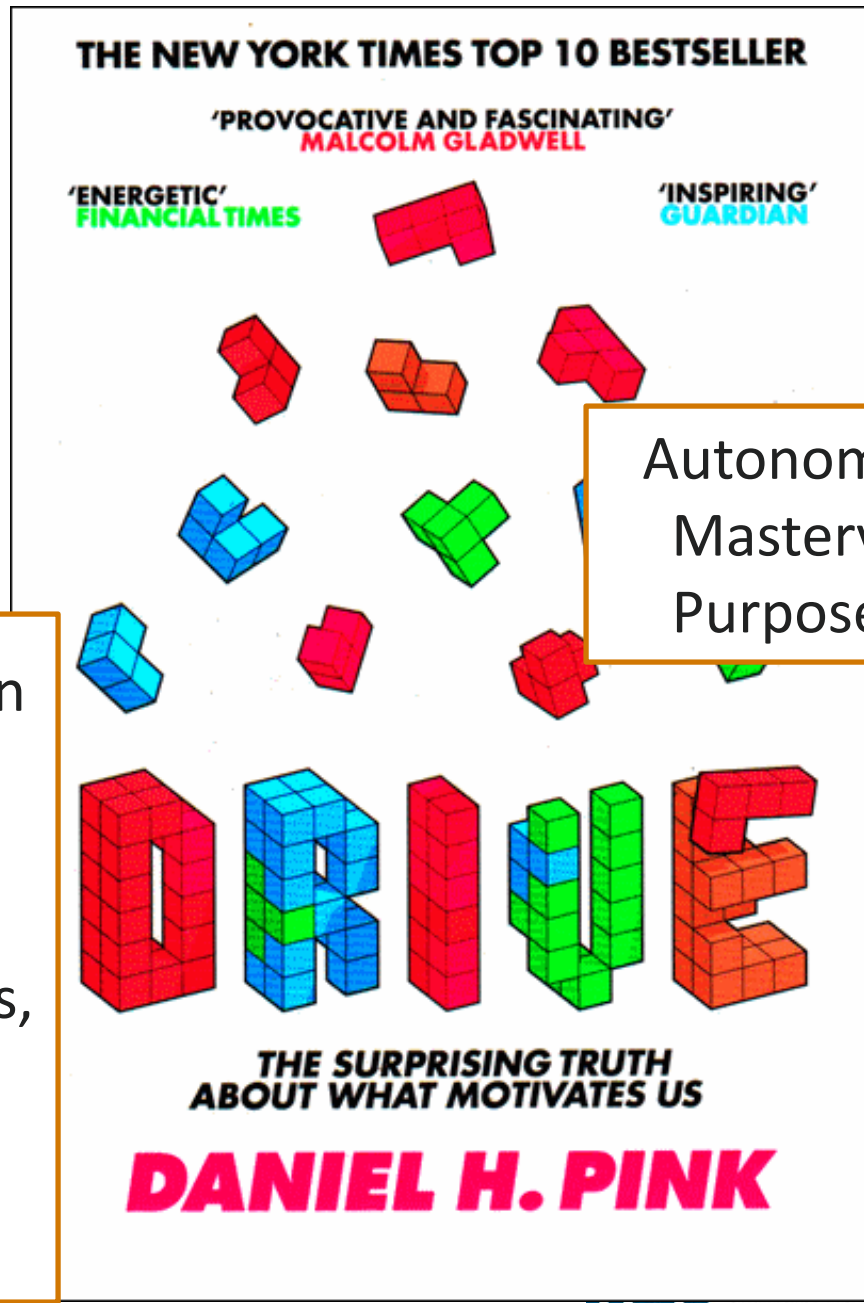
S. ADAMS E-mail: SCOTTADAMS@aol.com

W/C © 1995 United Feature Syndicate, Inc. (NYC)

<http://dilbert.com/strips/comic/1995-11-13/>



- Can extinguish intrinsic motivation
- Can diminish performance
- Can crush creativity
- Can crowd out good behavior
- Can encourage cheating, shortcuts, and unethical behavior
- Can become addictive
- Can foster short-term thinking



Autonomy  
Mastery  
Purpose

# Foundations of Software Engineering

## Lecture 6: Requirements Solicitation and Documentation

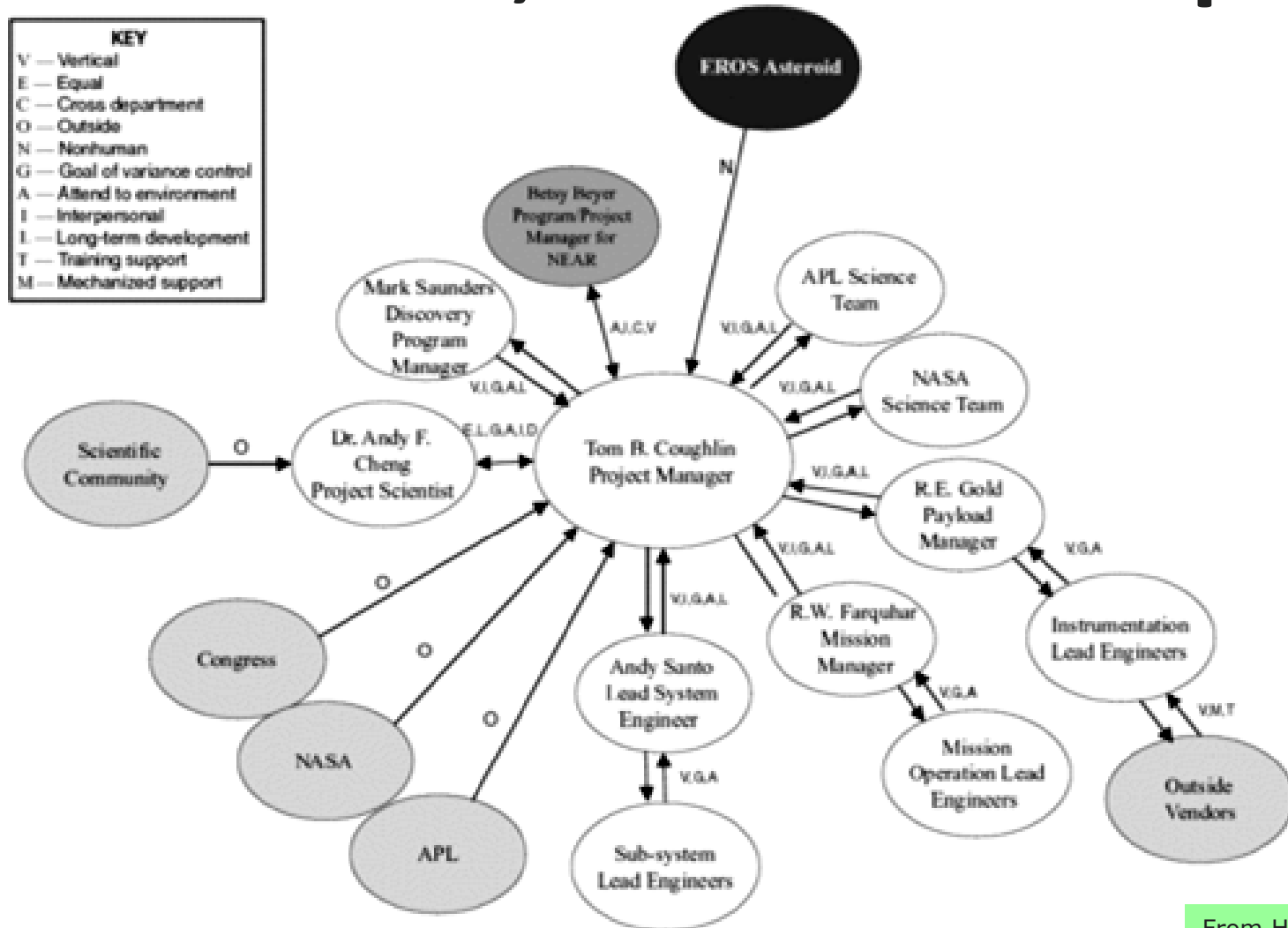
Christian Kästner

# Learning goals

- Basic proficiency in executing effective requirements interviews
- Understand tradeoffs of different documentation strategies
- Document requirements using use cases and user stories
- Recognize and resolve conflicts with priorities



# Stakeholders, a NASA example



From HSI NAP 11893

# Interviews



# Studying Artifacts (Content Analysis)

- Learn about the domain
  - Books, articles, wikipedia
- Learn about the system to be replaced
  - How does it work? What are the problems?  
Manuals? Bug reports?
- Learn about the organization
- Knowledge reuse from other systems?

# Abby Jones<sup>1</sup>



## 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 information processing style* when she needs to more information. So, instead of acting upon the first option that seems promising, she gathers information comprehensively to try to form a complete understanding of the problem before trying to solve it. Thus, her style is "burst-y"; first she reads a lot, then she acts on it in a batch of activity.
- **Learning: by Process vs. by Tinkering:** When learning new technology, Abby leans toward process-oriented learning, e.g., tutorials, step-by-step processes, wizards, online how-to videos, etc. She doesn't particularly like learning by tinkering with software (i.e., just trying out new features or commands to see what they do), but when she does tinker, it has positive effects on her understanding of the software.

<sup>1</sup>Abby represents users with motivations/attitudes and information/learning styles similar to hers. For data on females and males similar to and different from Abby, see <http://eusesconsortium.org/gender/gender.php>

# Handling inconsistencies

- Terminology, designation, structure:  
Build glossary, domain model
- Weak, strong conflicts: Negotiation required
  - Cause: different objectives of stakeholders  
=> resolve outside of requirements
  - Cause: quality tradeoffs => explore preferences

Examples?

# High- vs low- fidelity mockups



# Software Requirements Specification (SRS)

- Formal requirements document
- Several standards exists
- Often basis for contracts

## Table of Contents

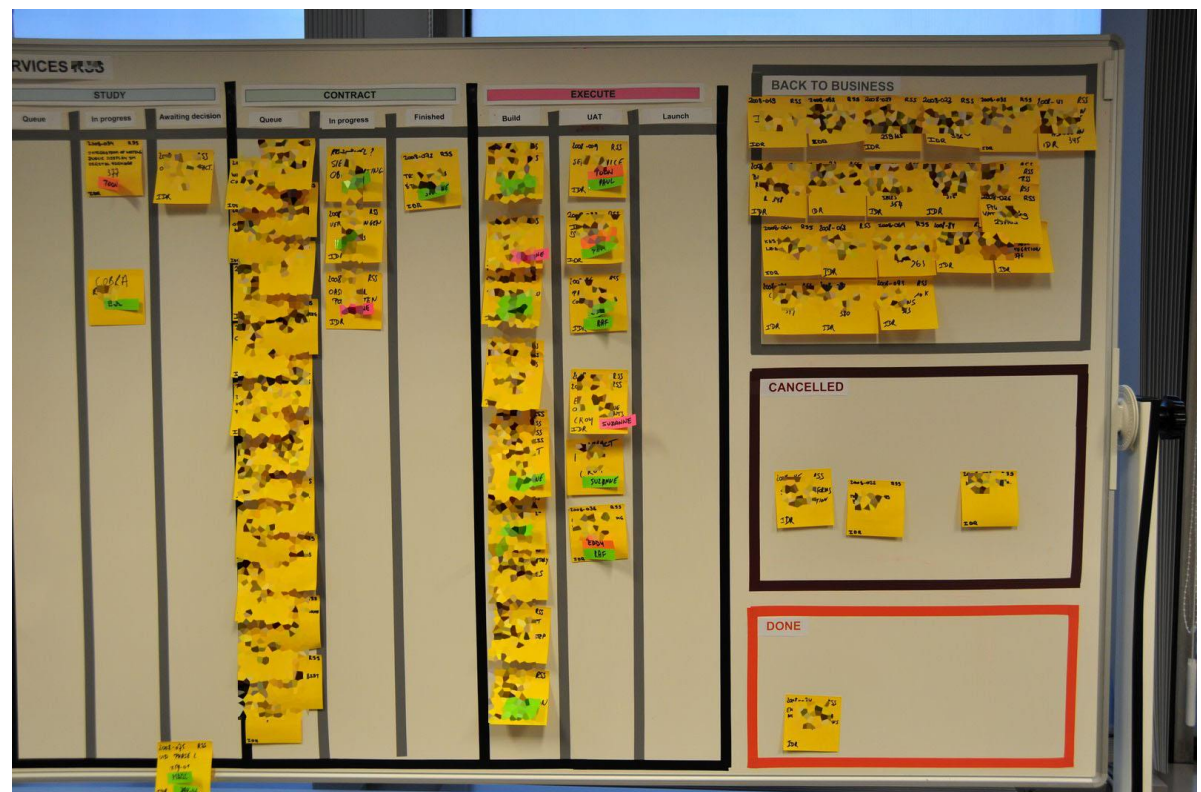
1	Introduction.....	4
1.1	Purpose.....	4
1.2	Document Conventions.....	4
1.3	Project Scope.....	4
1.4	References.....	4
2	System Description.....	4
3	Functional Requirements.....	4
3.1	System Features.....	4
3.1.1	System Feature 1.....	5
3.1.2	System Feature 2.....	5
3.2	Use Cases.....	5
3.2.1	Use Case Diagrams.....	5
3.2.2	Use Case 1.....	5
3.2.3	Use Case 2.....	5
3.3	Entity Relationship Diagrams.....	5
3.4	Data Dictionary.....	6
3.4.1	Entity 1.....	6
3.4.2	Entity 2.....	6
4	External Interface Requirements.....	6
5	Technical Requirements (Non functional).....	6
5.1	Performance.....	6
5.2	Scalability.....	6
5.3	Security.....	6
5.4	Maintainability.....	6
5.5	Usability.....	6
5.6	Multi lingual Support.....	6
5.7	Auditing and Logging.....	6
5.8	Availability.....	6
6	Open Issues.....	7

Use Case Name	(Title)
Scope	System under design
Level	User level, subprocess level
Primary actor	(actors can be primary, supporting, or offstage)
Stakeholders, interests	Important! A use case should include everything necessary to satisfy the stakeholders' interests.
Preconditions	What must always be true before a scenario begins. Not tested; assumed. Don't fill with pointless noise.
Success guarantees.	Aka post conditions
Main success scenario	Basic flow, "happy path", typical flow. Defer all conditions to the extensions. Records steps: interaction between actors, a validation, a state change by the system.
Extensions	Aka alternate flows. Usually the majority of the text. Sometimes branches off into another use case.
Special requirements	Where the non-functional/quality requirements live.
Technology and data variations list	Unavoidable technology constraints; try to keep to I/O technologies.
Frequency of occurrence	
Miscellaneous	



# Use of User Stories

- Keep a board of user stories, group them into “epics”



# Foundations of Software Engineering









Lecture 7: User stories and Risk

Michael Hilton

# Learning goals

- Document requirements as user stories
- Evaluate the quality of a user story
- Understand risk and its role in requirements, specifically how it can be identified, analyzed, and then mitigated/handled in system design.

# Requirements should be

1. Correct  According to both the engineer and the customer
2. Consistent  In that there are no conflicting requirements. Quality requirements are particularly dangerous.
3. Unambiguous  Ambiguous: multiple readers can walk away with different but valid interpretations.
4. Complete  Covers all required behavior and output for all inputs under all constraints.
5. Feasible  Can it be done at all? Again, quality/non-functional reqs are particularly vulnerable.
6. Relevant 
7. Testable  Acceptance tests and metrics are possible/obvious.
8. Traceable  Organized, uniquely labeled.

# Bird Risks



# How to evaluate user story?

Follow the INVEST  
guidelines for good  
user stories!



one|80  
SERVICES



# Interview

Josh Gardner!

BS in Computer Science from University of Buffalo

Developer at SPAWAR 4 years

Server Lead at Mobiquity Inc. 5 years

Mobiquity is a software services company, meaning we sell our skills in building software (mobile apps, web apps and now Alexa skills) and building cloud infrastructure to other companies. That covers the whole range of activities, visual design, project management, and dragging what they actually want out of them ('gathering requirements'), and then building the system.

My personal role has become a combination of actually writing nodejs code, and managing a pack of fellow server devs on one of our large health care projects. Previously I was a rank and file developer for a few consumer services type apps, and then was a full stack lead on a smaller Healthcare app (Novartis Heart Partner). Software services is interesting in that you have to frequently deal with different customers and different types of work (both technically and managerially) so I have sometimes a fairly different view of the business than folks who work in a more product company type setup, where the vision can often extend years in advance.



# HW2: Requirements Collection



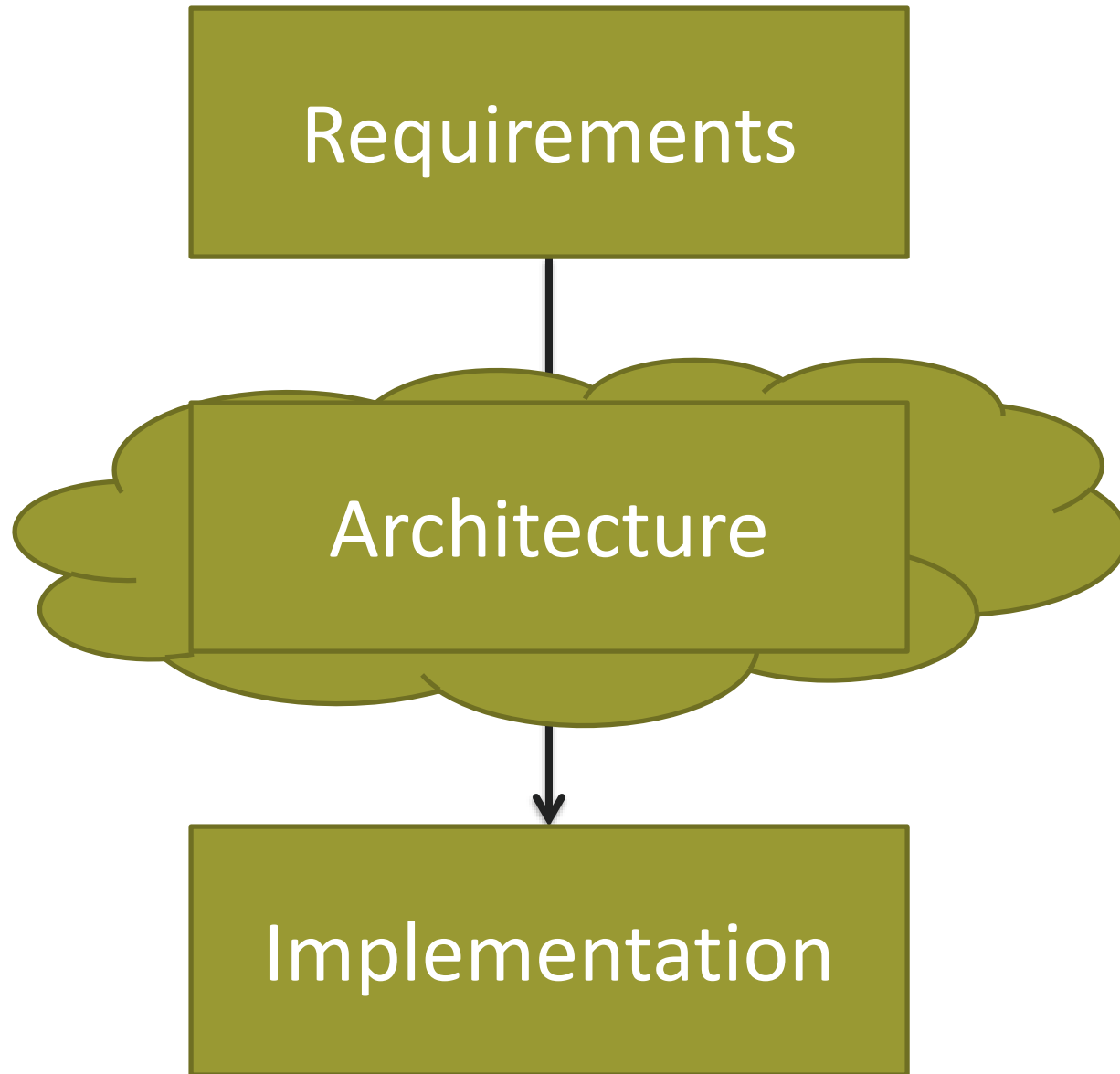
# Foundations of Software Engineering

Lecture 8: Introduction to Software  
Architecture and Documentation

Michael Hilton

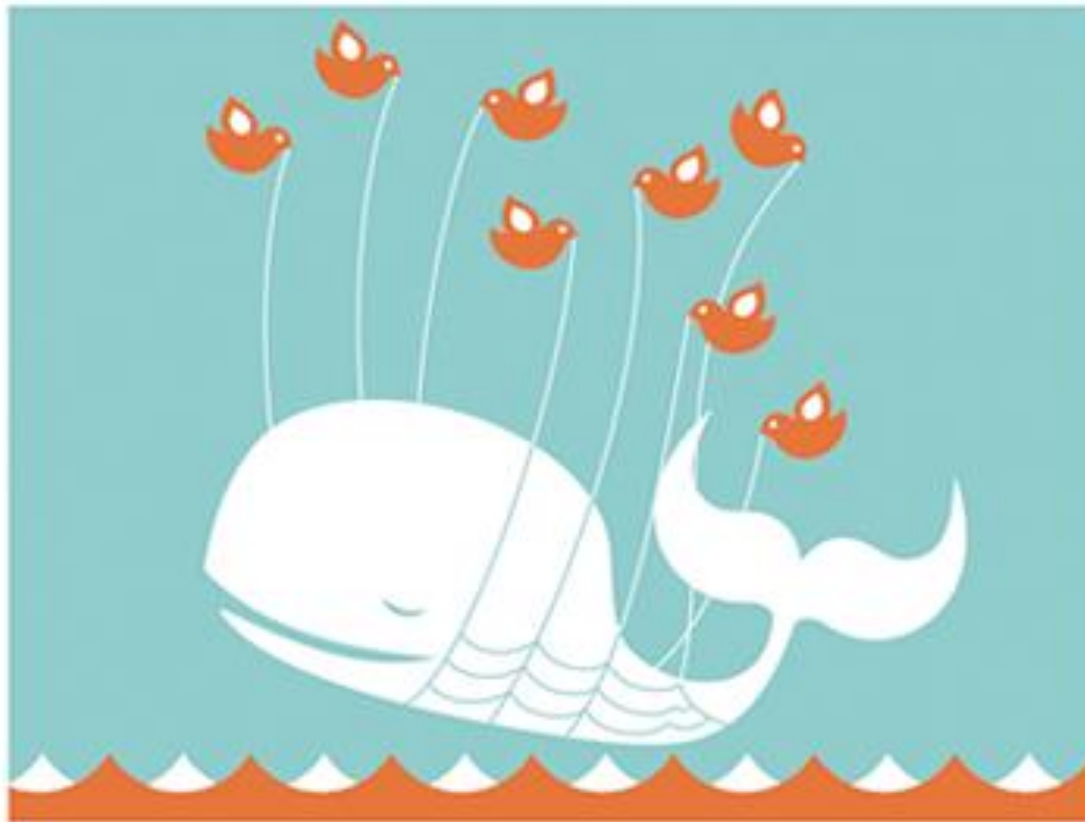
# Learning Goals

- Understand the abstraction level of architectural reasoning
- Approach software architecture with quality attributes in mind
- Distinguish software architecture from (object-oriented) software design
- Use notation and views to describe the architecture suitable to the purpose
- Document architectures clearly, without ambiguity
- Understand the benefits and challenges of traceability.



## Twitter is over capacity.

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



# Foundations of Software Engineering

Lecture 9: Architecture Documentation,  
Patterns, and Tactics

Christian Kaestner

# Learning Goals

- Use notation and views to describe the architecture suitable to the purpose
- Document architectures clearly, without ambiguity
- Understand the benefits and challenges of traceability.
- Understand key parts of architectural process
- Use architectural styles and tactics for design decisions
- Make justified architectural decisions for new systems and within existing systems

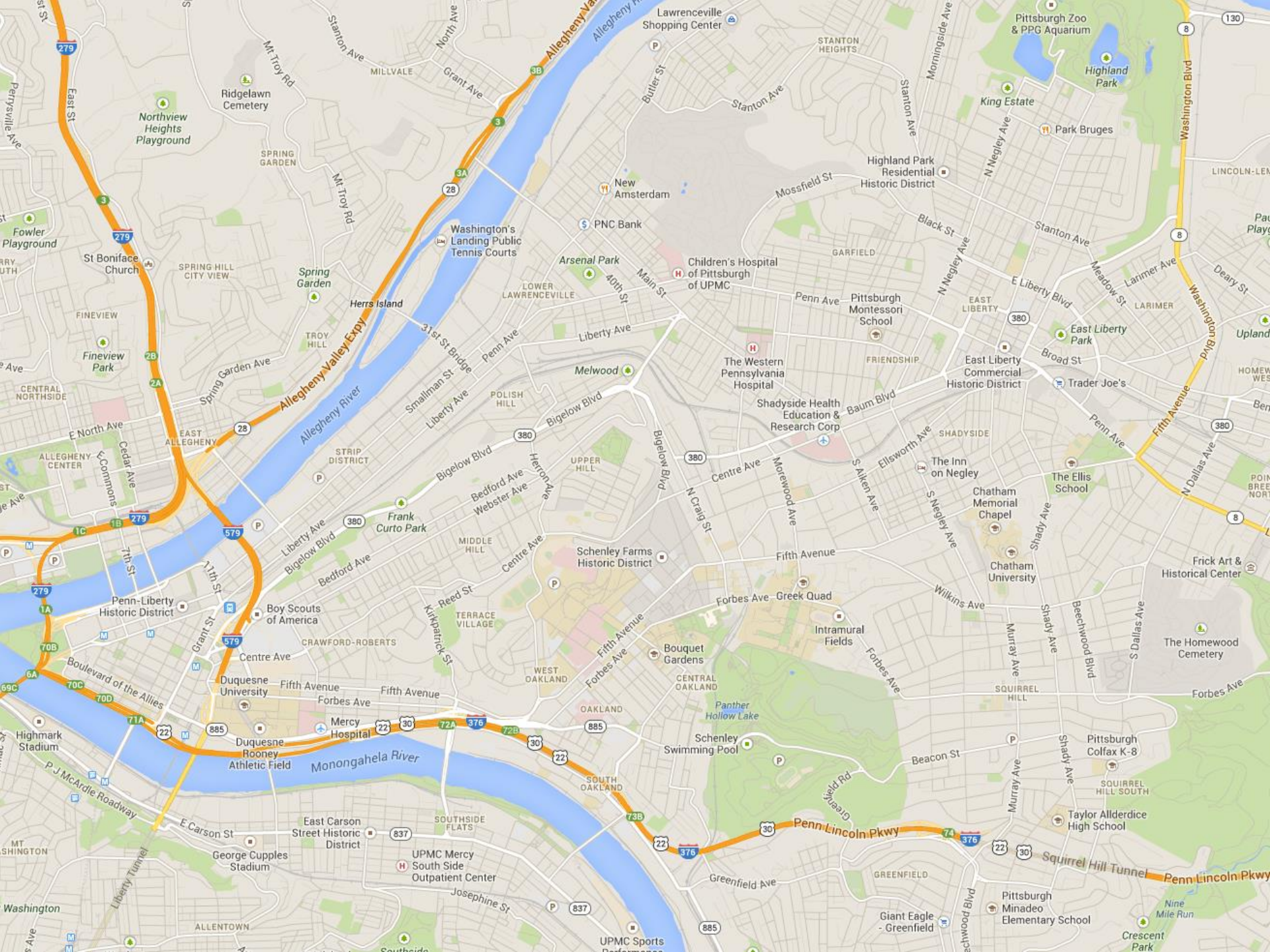
# Design vs. Architecture

## Design Questions

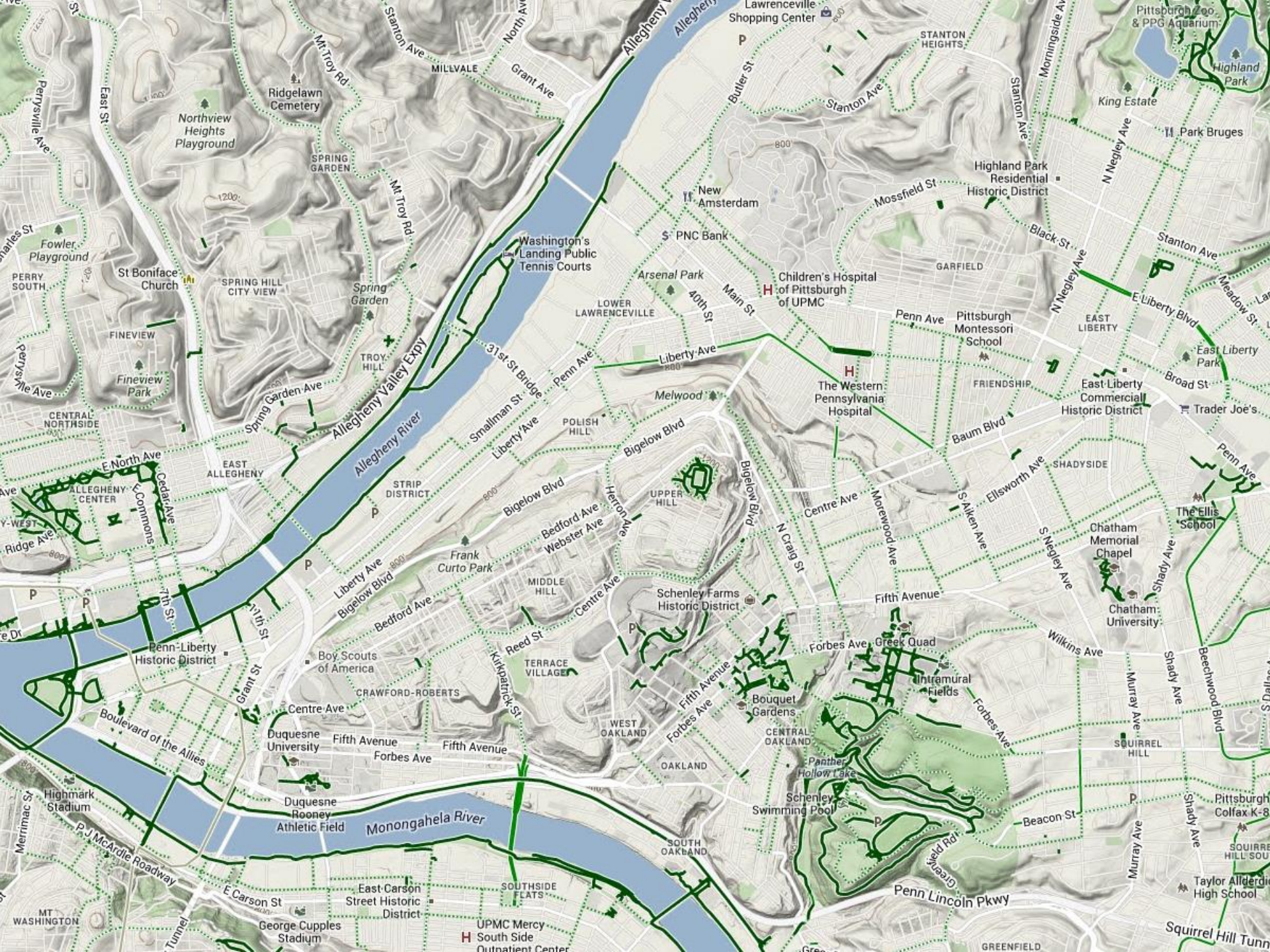
- How do I add a menu item in Eclipse?
- How can I make it easy to add menu items in Eclipse?
- What lock protects this data?
- How does Google rank pages?
- What encoder should I use for secure communication?
- What is the interface between objects?

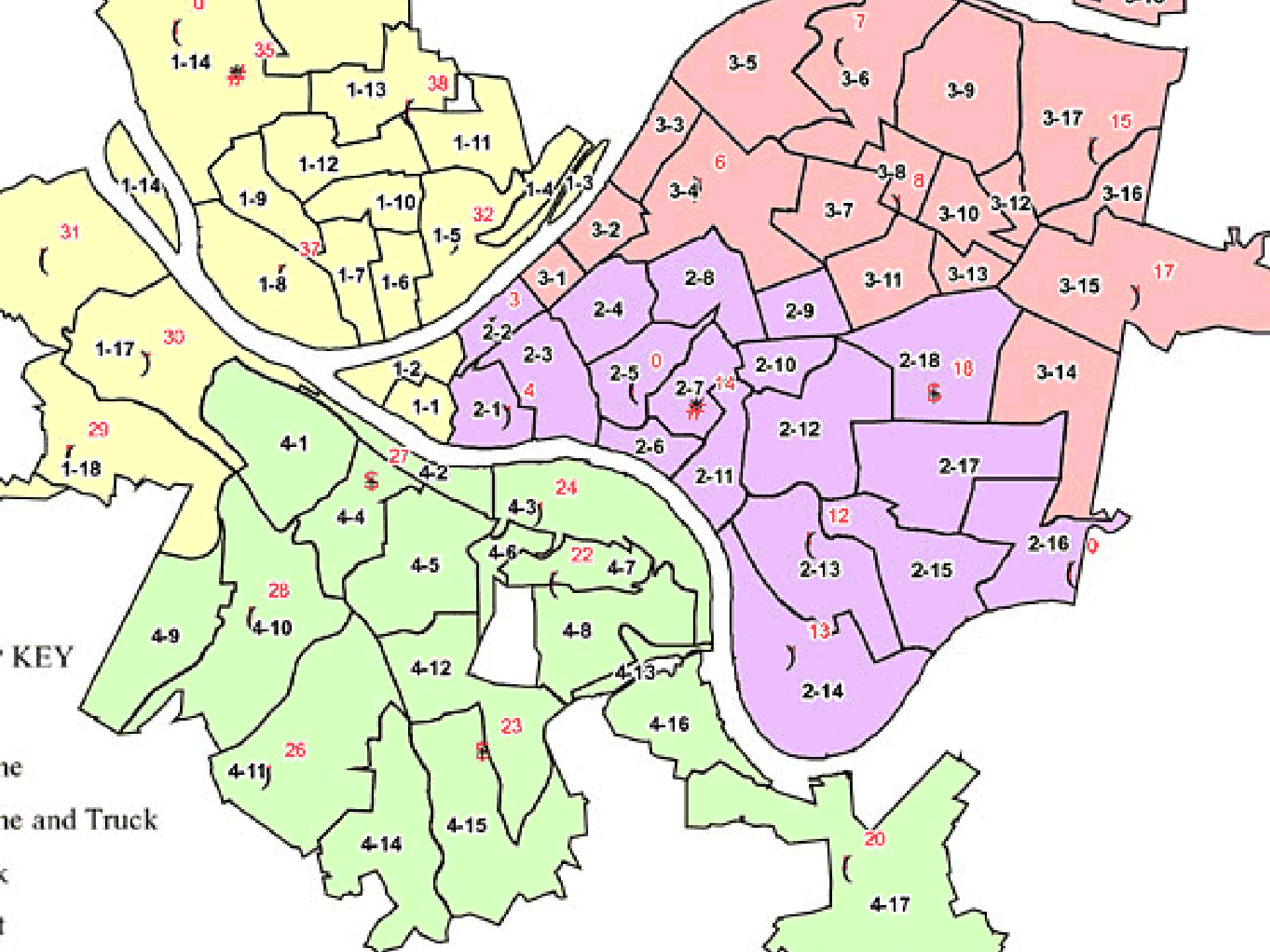
## Architectural Questions

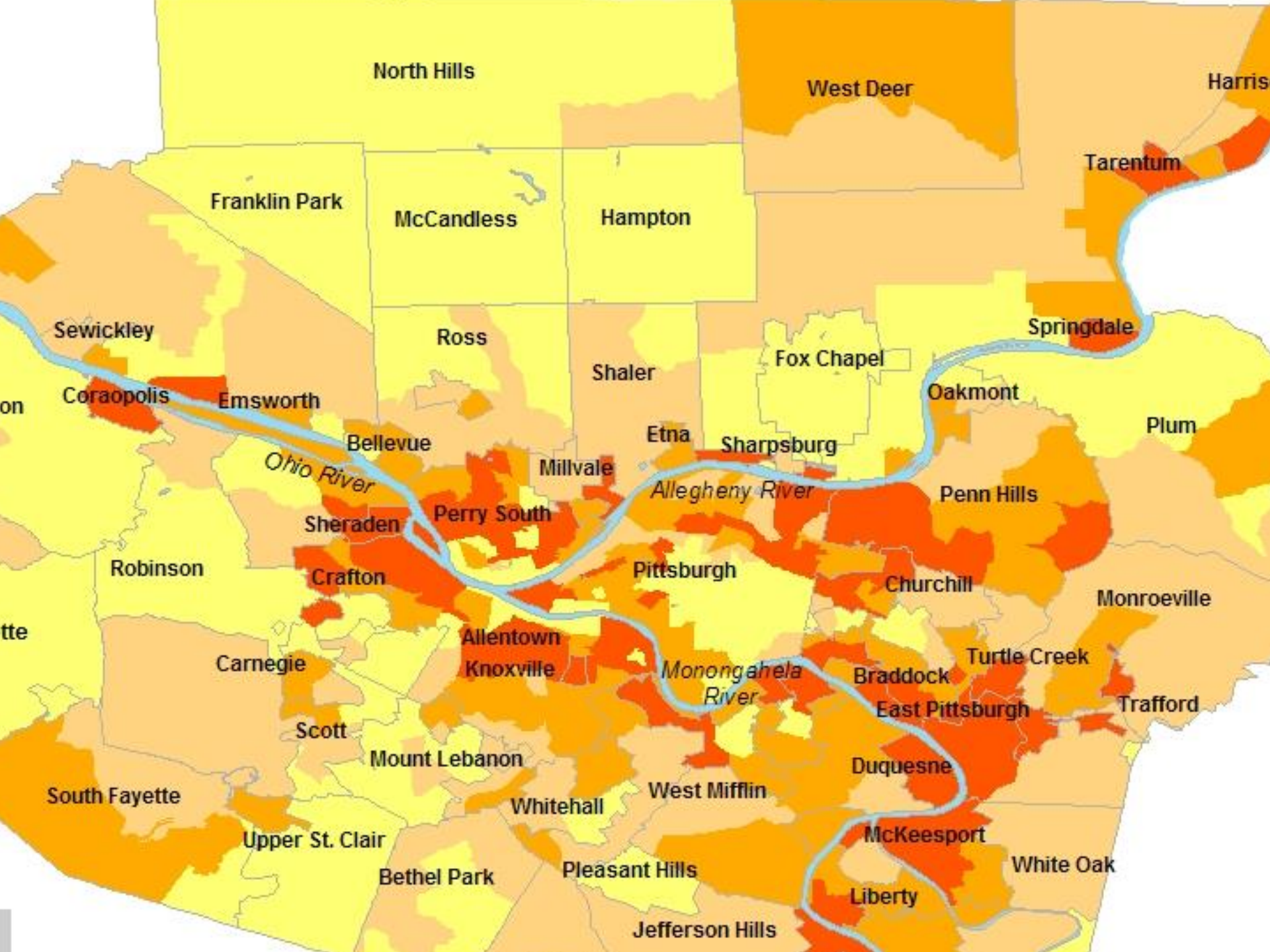
- How do I extend Eclipse with a plugin?
- What threads exist and how do they coordinate?
- How does Google scale to billions of hits per day?
- Where should I put my firewalls?
- What is the interface between subsystems?

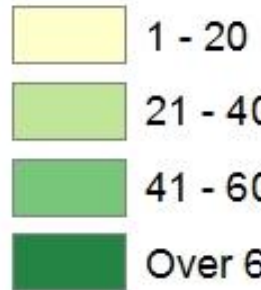
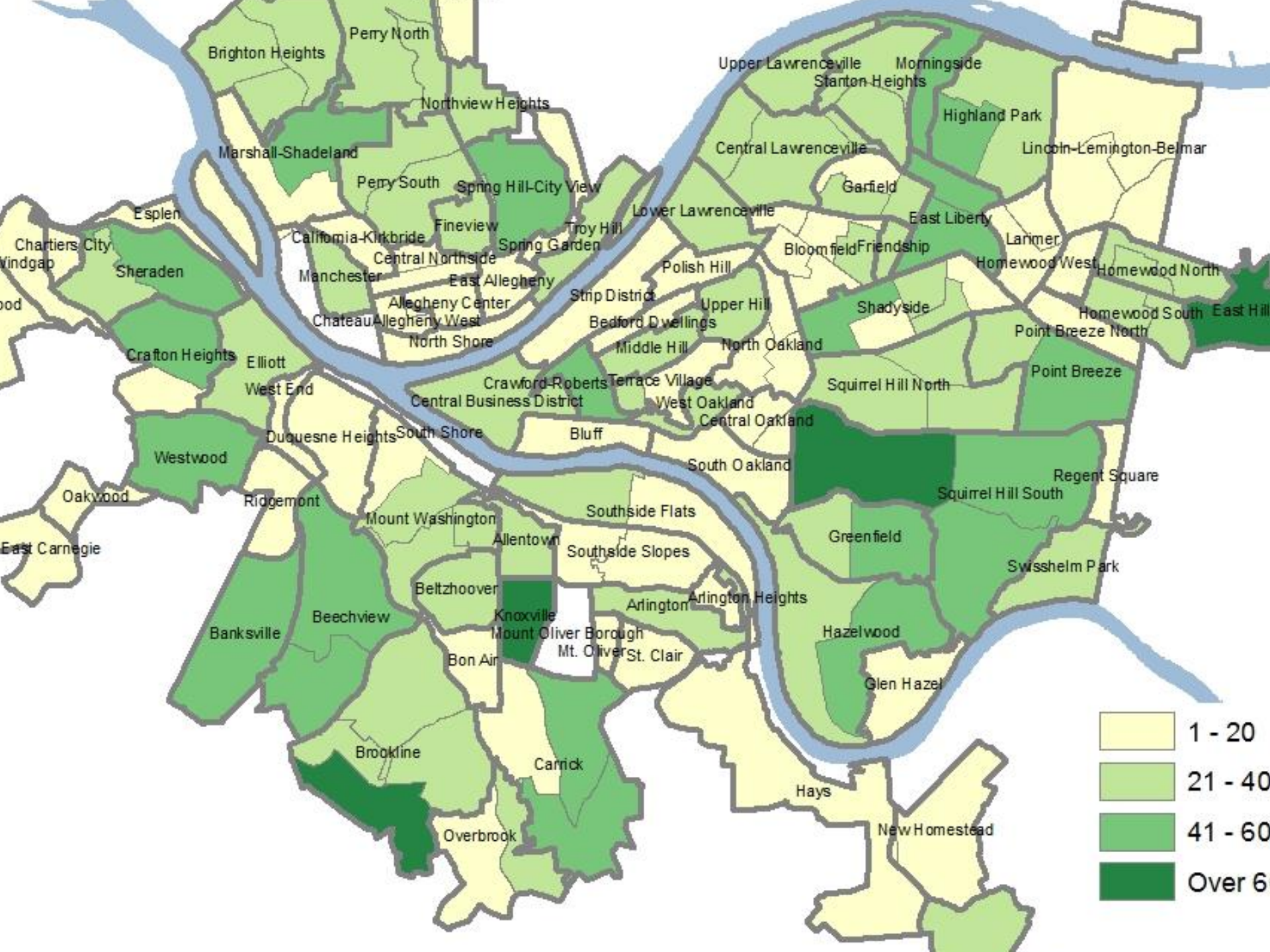




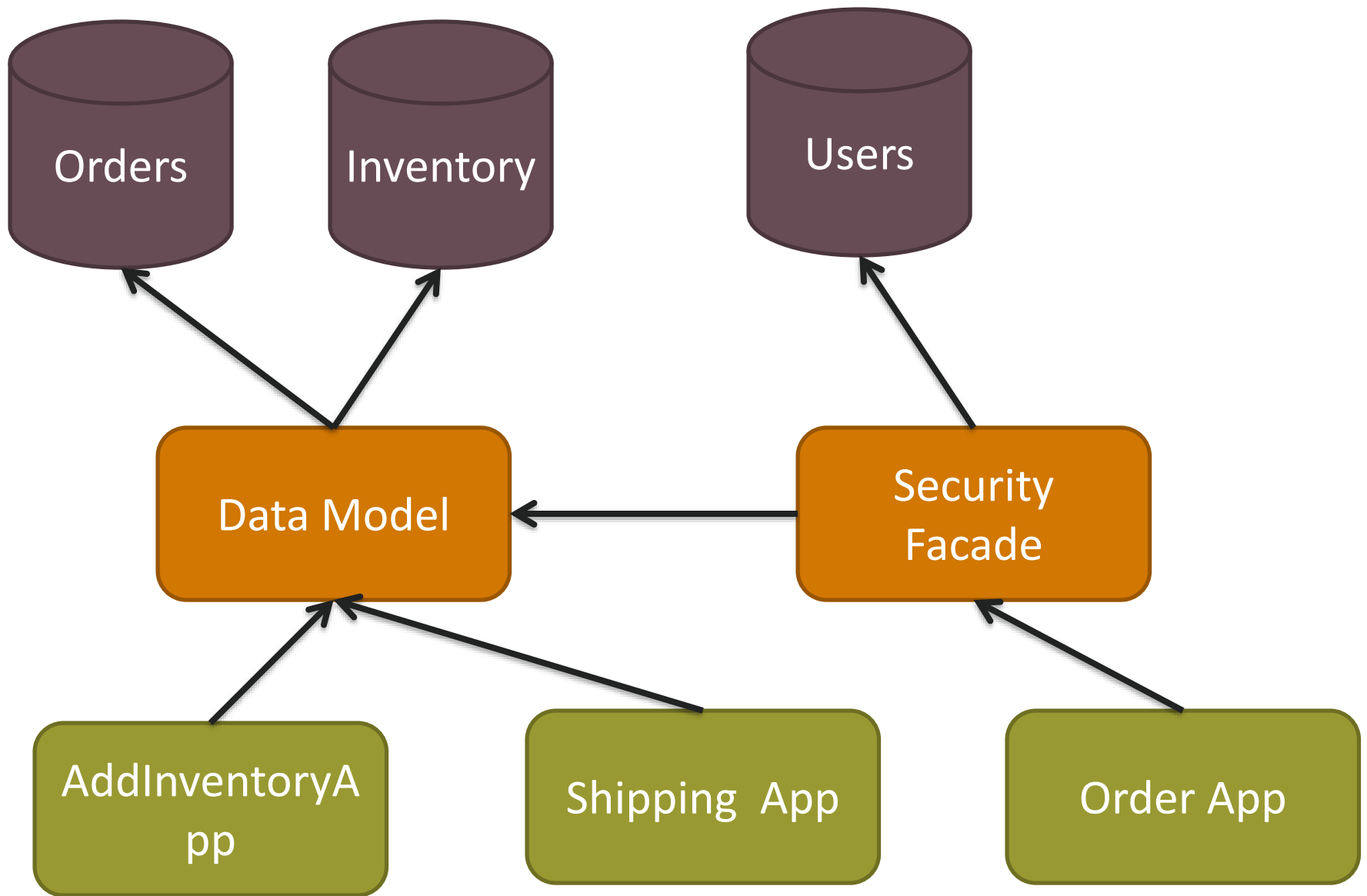












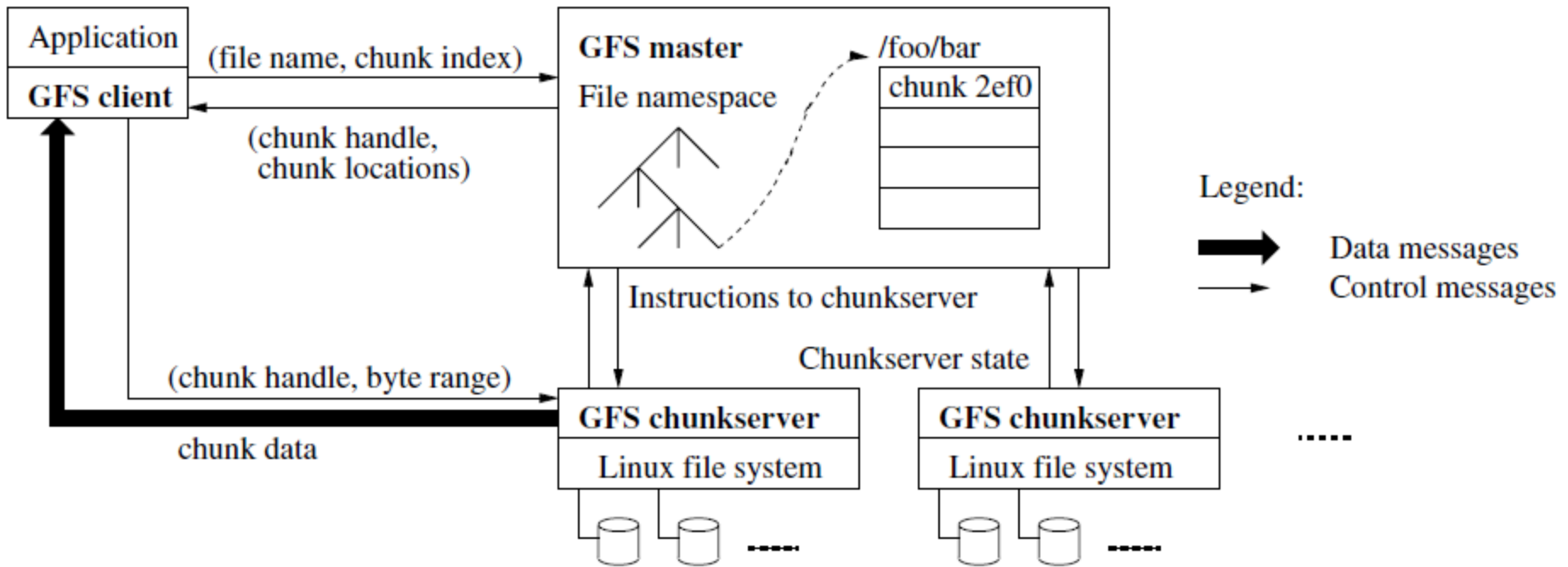
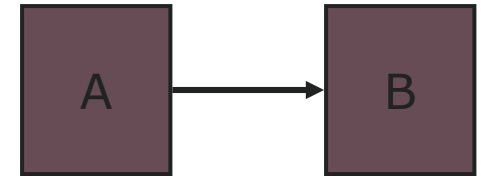


Figure 1: GFS Architecture

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

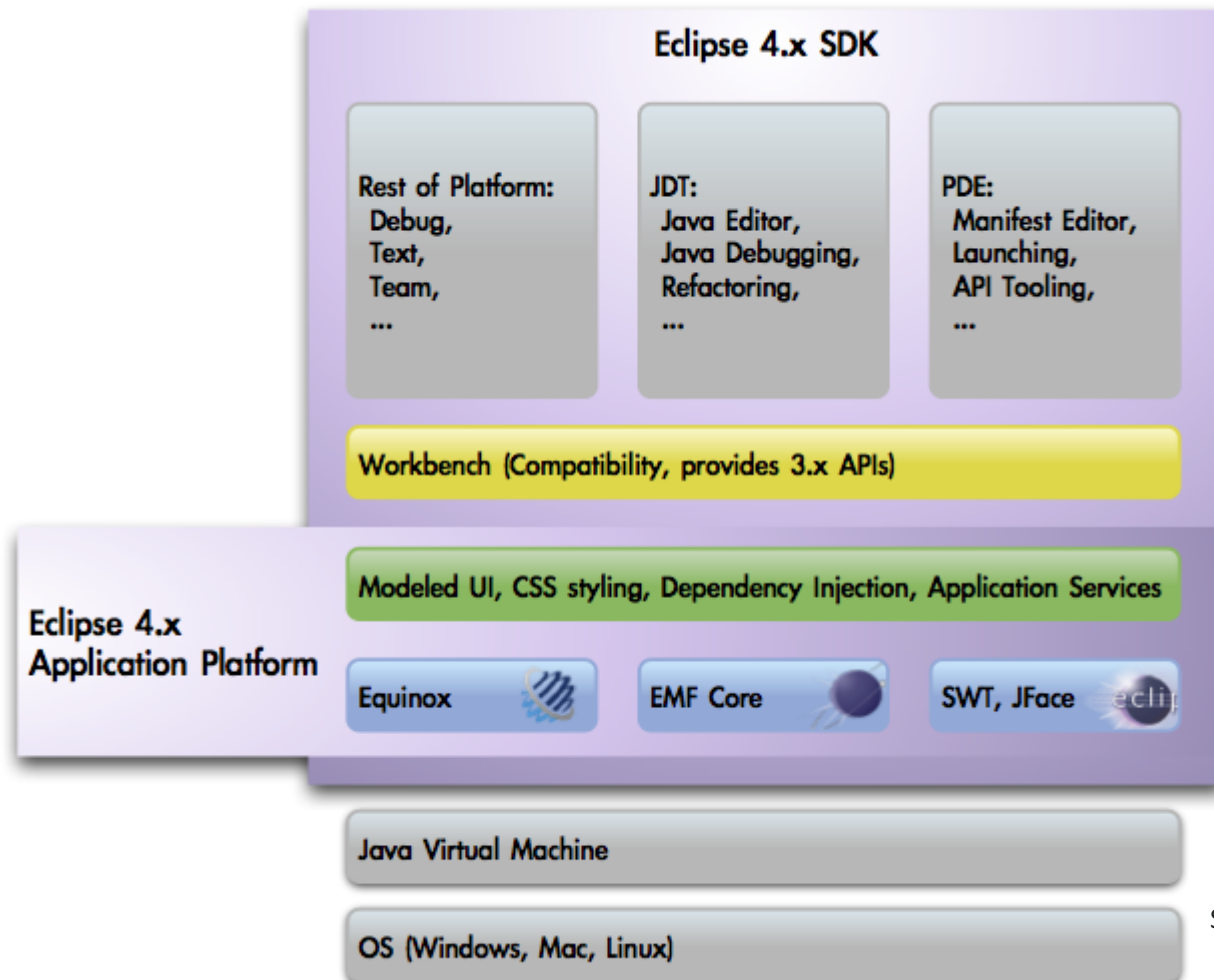
# What could the arrow mean?

- Many possibilities
  - A passes control to B
  - A passes data to B
  - A gets a value from B
  - A streams data to B
  - A sends a message to B
  - A creates B
  - A occurs before B
  - B gets its electricity from A
  - ...

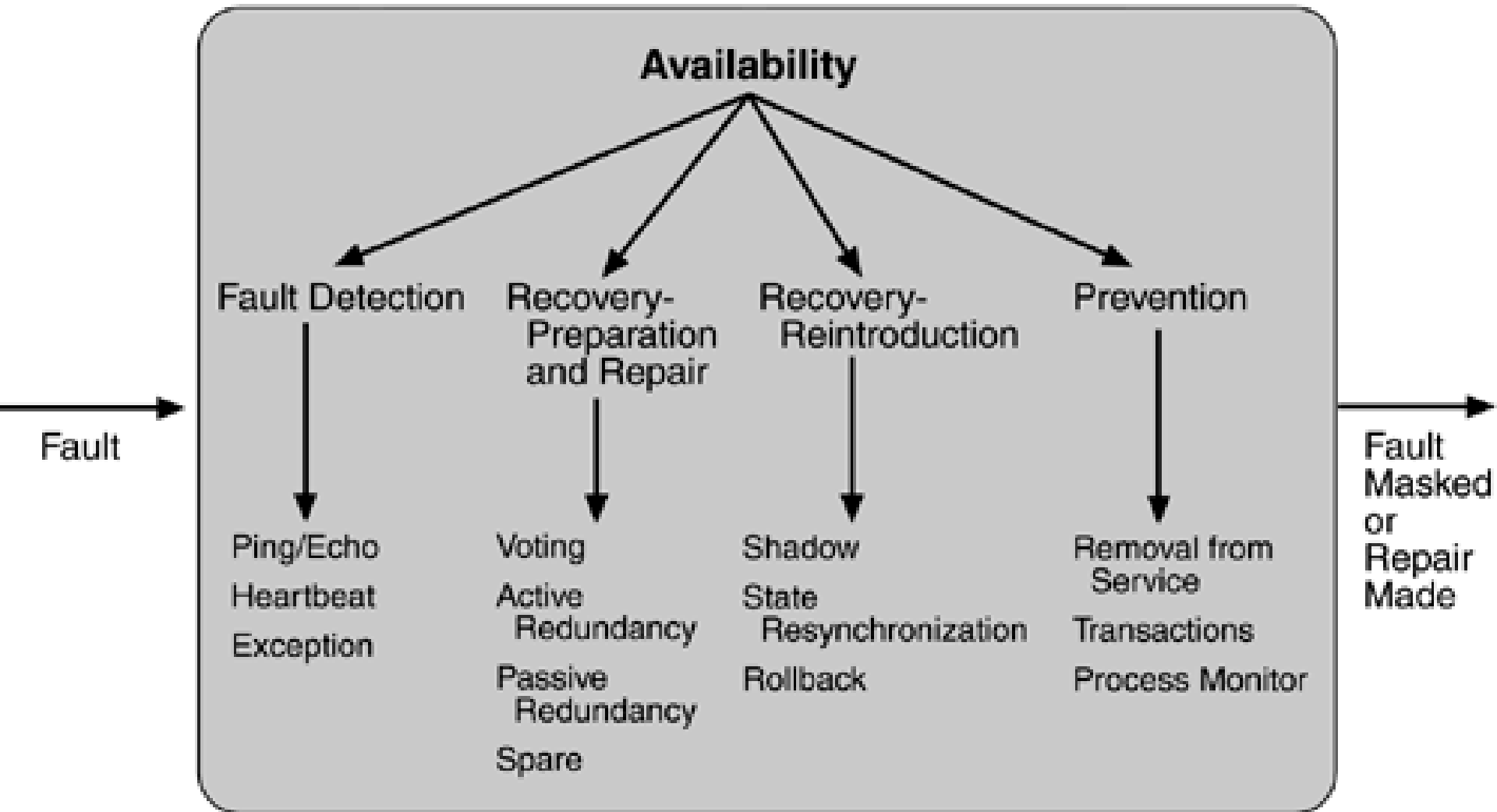




# Layered system



Source: eclipse.org



# Foundations of Software Engineering

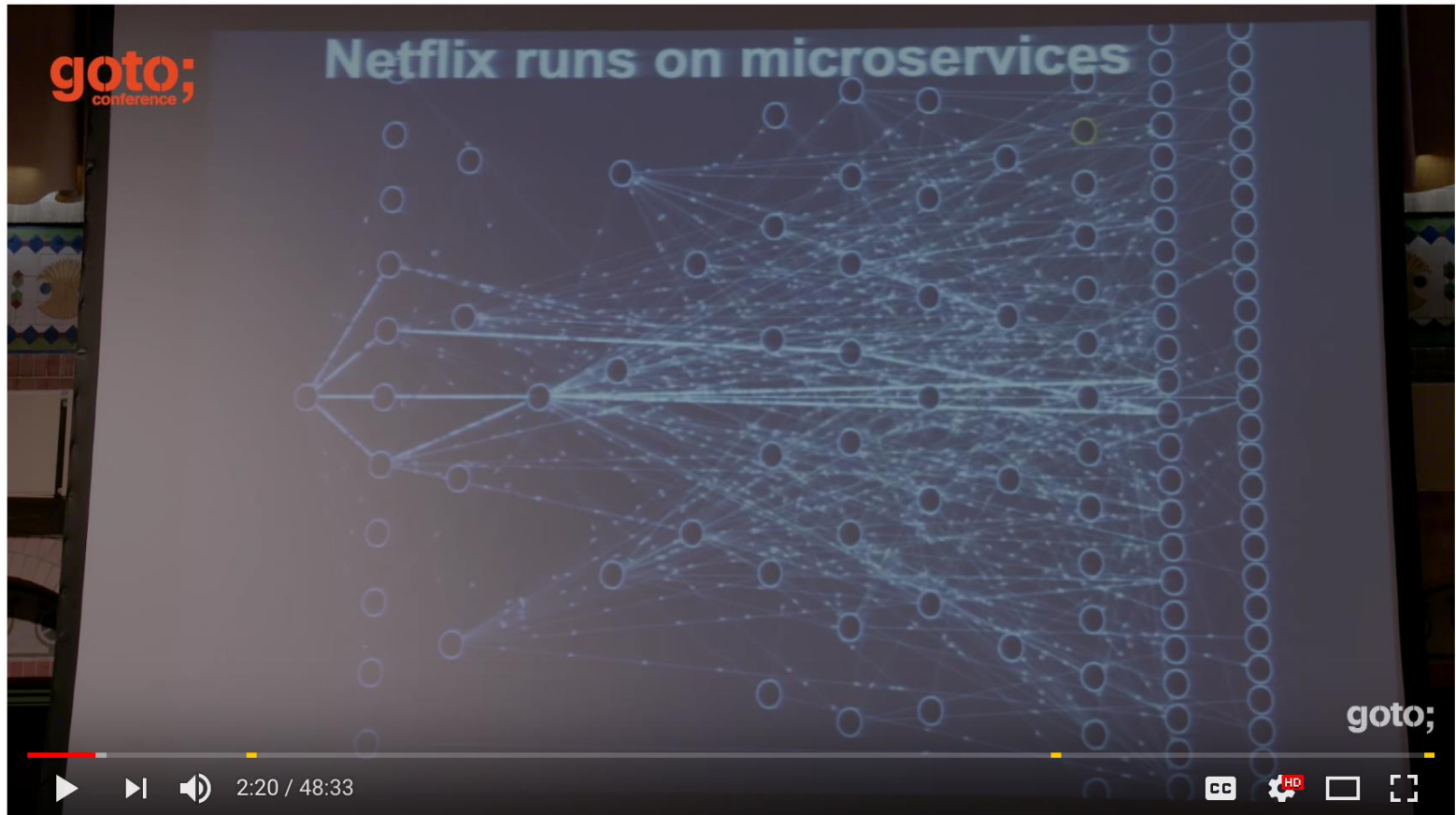
Architecture – Styles and Hypes

Michael Hilton

# Learning Goals

- Recognize architectural styles and their implications
- Reason about system structures and their tradeoffs with architectural views and styles
- Reason about tradeoffs of Microservice architectures
- Understand the key ideas of DevOps
- Appreciate challenge of architecture in practice



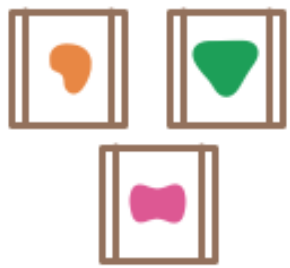


# Netflix Discussion

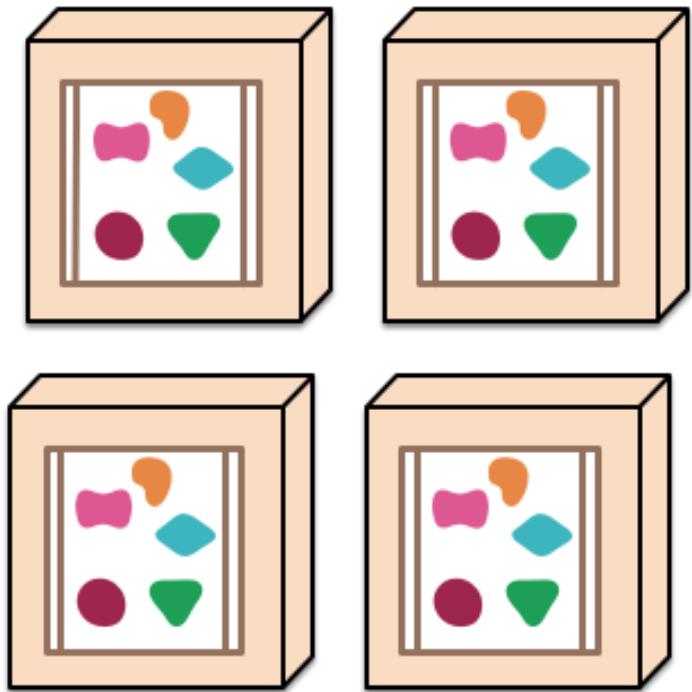
*A monolithic application puts all its functionality into a single process...*



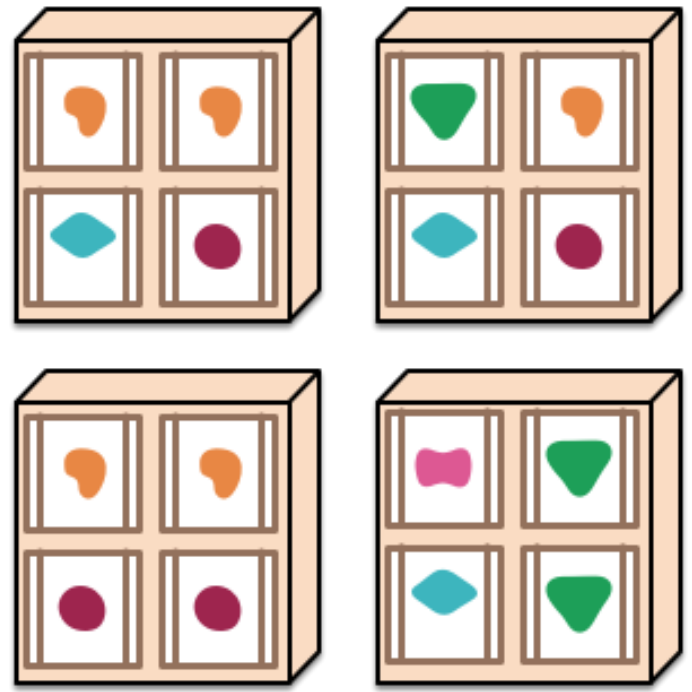
*A microservices architecture puts each element of functionality into a separate service...*



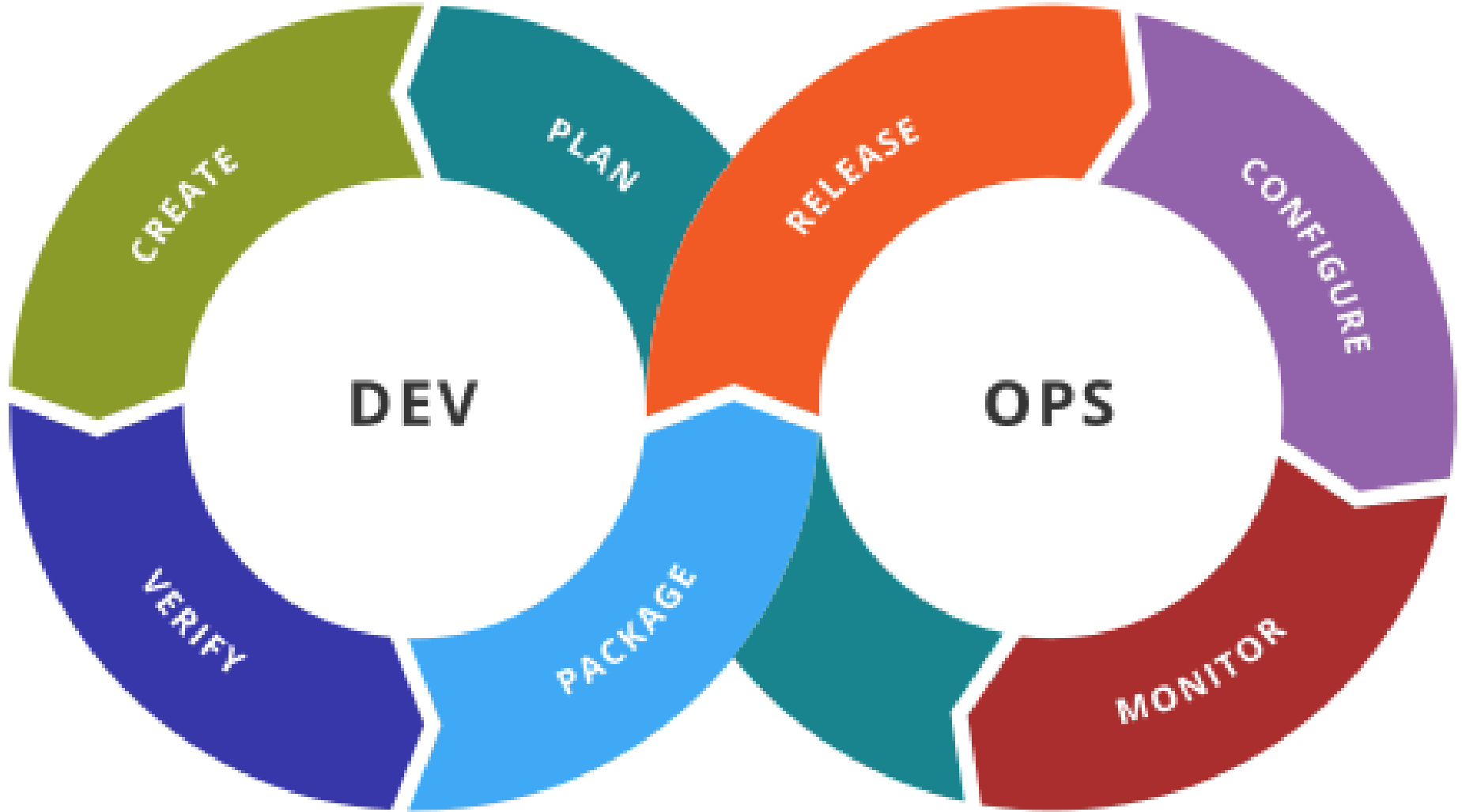
*... and scales by replicating the monolith on multiple servers*



*... and scales by distributing these services across servers, replicating as needed.*



source: <http://martinfowler.com/articles/microservices.html>





# Continuous Integration /Deployment

- Release several times per day
- Incremental rollout, quick rollback



## Travis CI





- Lightweight virtualization
- Sub-second boot time
- Sharable virtual images with full setup incl. configuration settings
- Used in development and deployment
- Separate docker images for separate services (web server, business logic, database, ...)

# HW3: Architecture

# Foundations of Software Engineering

Lecture 12 – Intro to QA, Testing  
Christian Kaestner

# Learning goals

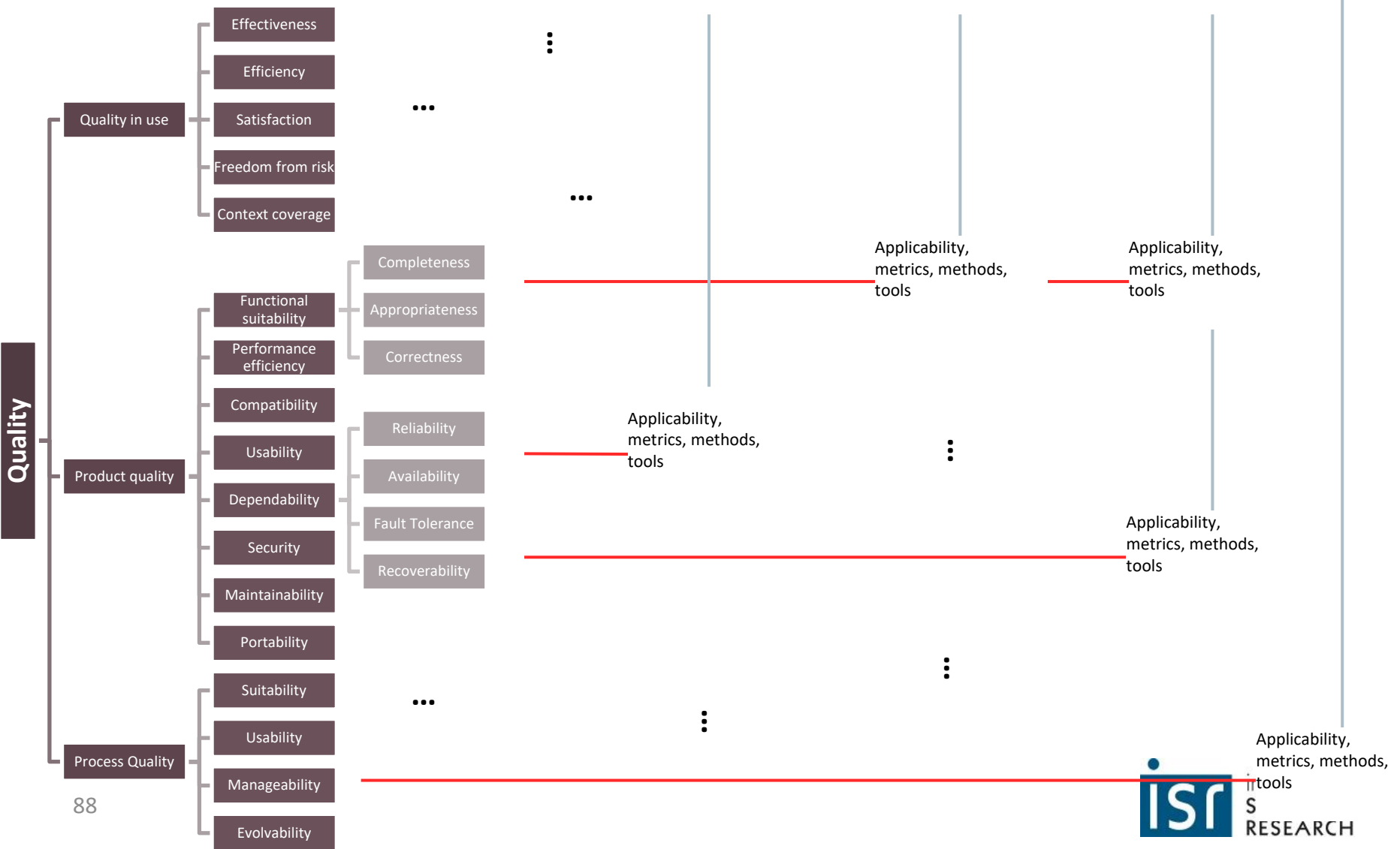
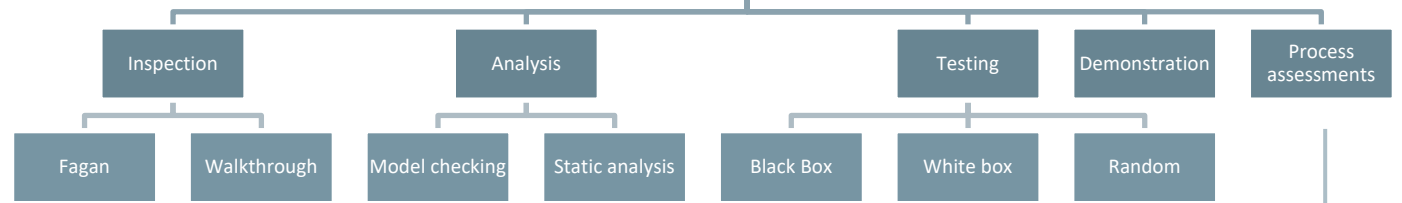
- Define software analysis
- Distinguish validation and verification
- Understand a range of QA techniques
- Apply testing and test automation for functional correctness
- Understand opportunities and challenges for testing quality attributes; enumerate testing strategies to help evaluate the following quality attributes: usability, reliability, security, robustness (both general and architectural), performance, integration.
- Discuss the limitations of testing

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

# Validation vs Verification

- **Verification:** Does the system meet its specification?
  - i.e. did we build the system correctly?
- **Verification:** are there flaws in design or code?
  - i.e. are there incorrect design or implementation decisions?
- Validation: Does the system meet the needs of users?
  - i.e. did we build the right system?
- Validation: are there flaws in the specification?
  - i.e., did we do requirements capture incorrectly?

# Verification





	Error exists	No error exists
Error Reported	True positive (correct analysis result)	False positive
No Error Reported	False negative	True negative (correct analysis result)

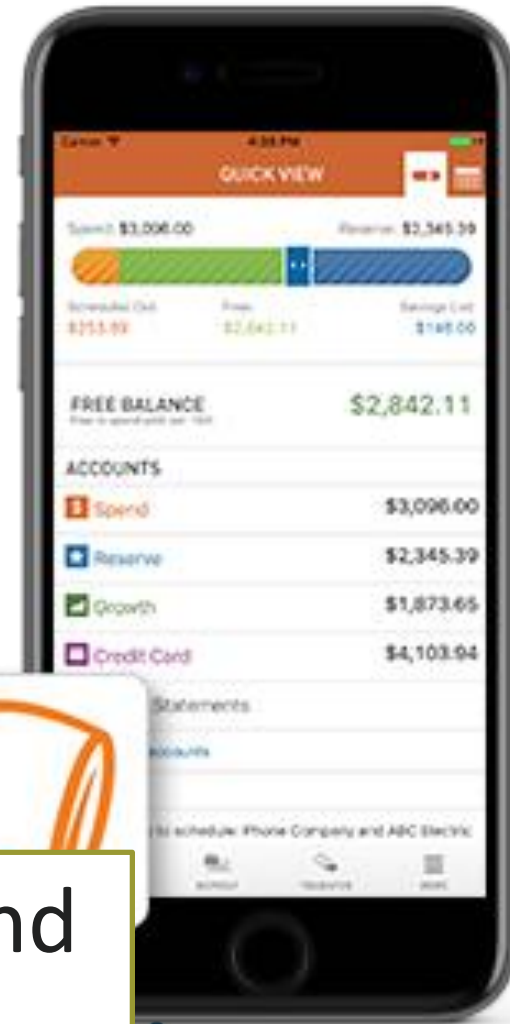
### Sound Analysis:

reports all defects  
-> no false negatives  
typically overapproximated

### Complete Analysis:

every reported defect is an actual defect  
-> no false positives  
typically underapproximated

# Brief Case Discussion



What qualities are important and how can you assure them?

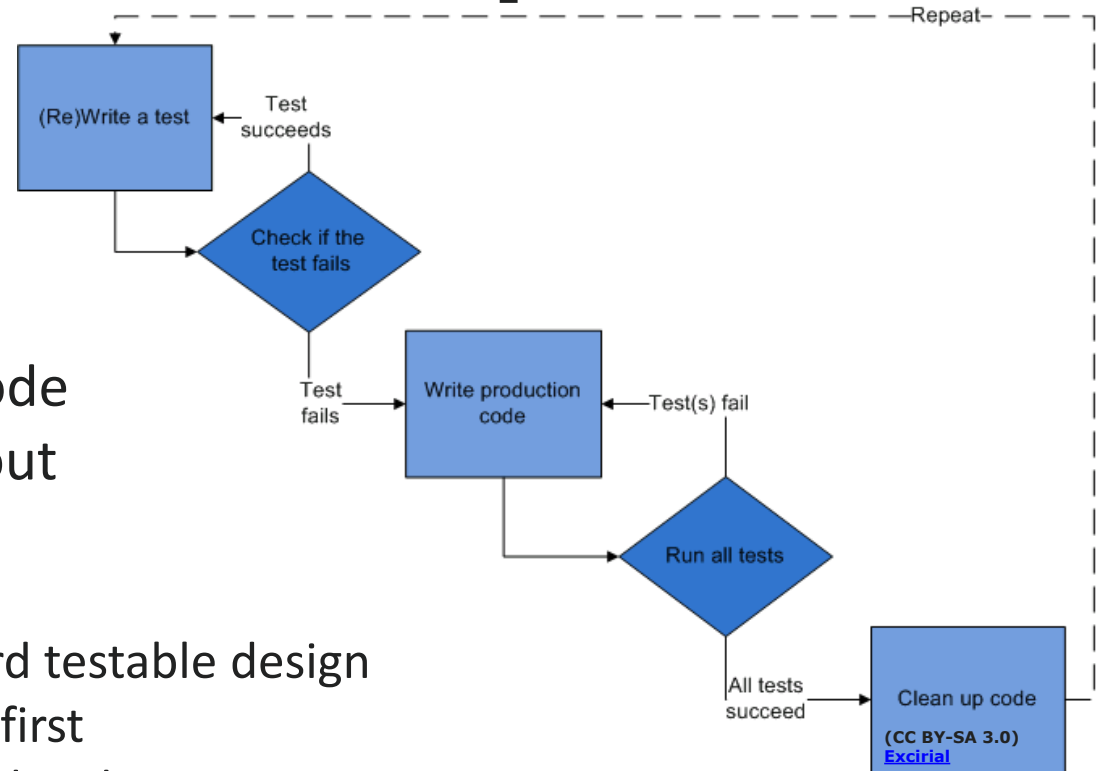
# Who's to blame?

```
Algorithms.shortestDistance(graph,  
    "Tom", "Anne");
```

> `ArrayOutOfBoundsException`

# 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



### Packages

[All](#)

- [net.sourceforge.cobertura.ant](#)
- [net.sourceforge.cobertura.check](#)
- [net.sourceforge.cobertura.coveragedata](#)
- [net.sourceforge.cobertura.instrument](#)
- [net.sourceforge.cobertura.merge](#)
- [net.sourceforge.cobertura.reporting](#)
- [net.sourceforge.cobertura.reporting.html](#)
- [net.sourceforge.cobertura.reporting.html.files](#)
- [net.sourceforge.cobertura.reporting.xml](#)
- [net.sourceforge.cobertura.util](#)

### All Packages

#### Classes

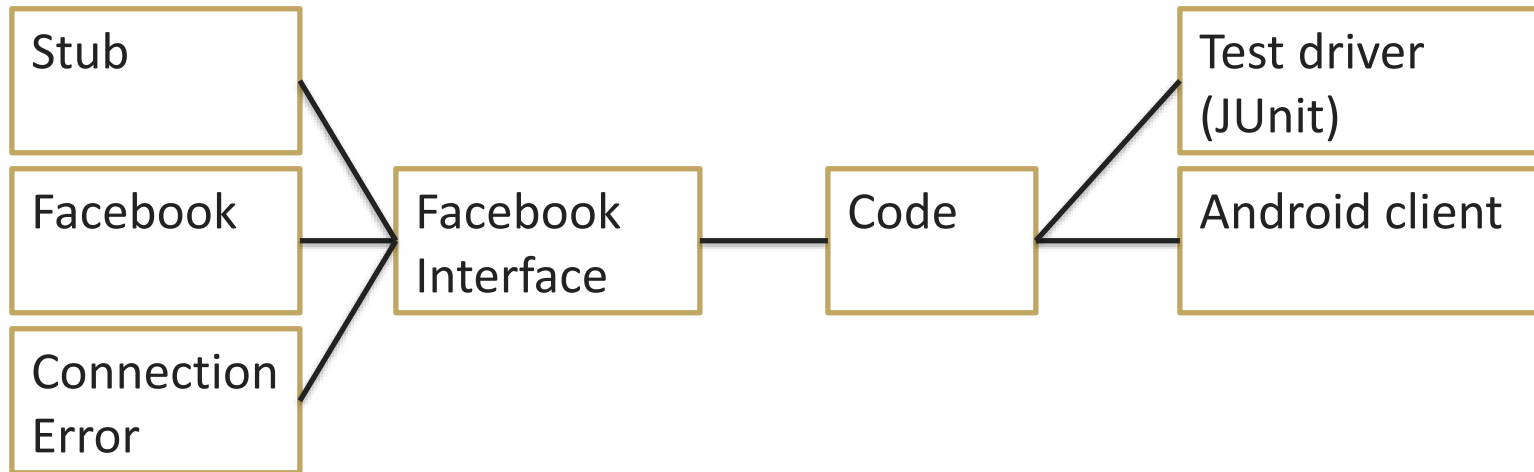
- [AntUtil \(88%\)](#)
- [Archive \(100%\)](#)
- [ArchiveUtil \(80%\)](#)
- [BranchCoverageData \(N/A\)](#)
- [CheckTask \(0%\)](#)
- [ClassData \(N/A\)](#)
- [ClassInstrumenter \(94%\)](#)
- [ClassPattern \(100%\)](#)
- [CoberturaFile \(73%\)](#)
- [CommandLineBuilder \(96%\)](#)
- [CommonMatchingTask \(88%\)](#)
- [ComplexityCalculator \(100%\)](#)
- [ConfigurationUtil \(50%\)](#)
- [CopyFiles \(87%\)](#)
- [CoverageData \(N/A\)](#)
- [CoverageDataContainer \(N/A\)](#)
- [CoverageDataFileHandler \(N/A\)](#)
- [CoverageRate \(0%\)](#)
- [ExcludeClasses \(100%\)](#)
- [FileFinder \(96%\)](#)
- [FileLocker \(0%\)](#)
- [FirstPassMethodInstrumenter \(100%\)](#)
- [HTMLReport \(94%\)](#)
- [HasBeenInstrumented \(N/A\)](#)
- [Header \(80%\)](#)

## Coverage Report - All Packages

Package ^	# Classes	Line Coverage		Branch Coverage		Compl
<b>All Packages</b>	55	75%		64%		
<a href="#">net.sourceforge.cobertura.ant</a>	11	52%		43%		
<a href="#">net.sourceforge.cobertura.check</a>	3	0%		0%		
<a href="#">net.sourceforge.cobertura.coveragedata</a>	13	N/A		N/A		
<a href="#">net.sourceforge.cobertura.instrument</a>	10	90%		75%		
<a href="#">net.sourceforge.cobertura.merge</a>	1	86%		88%		
<a href="#">net.sourceforge.cobertura.reporting</a>	3	87%		80%		
<a href="#">net.sourceforge.cobertura.reporting.html</a>	4	91%		77%		
<a href="#">net.sourceforge.cobertura.reporting.html.files</a>	1	87%		62%		
<a href="#">net.sourceforge.cobertura.reporting.xml</a>	1	100%		95%		
<a href="#">net.sourceforge.cobertura.util</a>	9	60%		69%		
<a href="#">someotherpackage</a>	1	83%		N/A		

Report generated by [Cobertura](#) 1.9 on 6/9/07 12:37 AM.

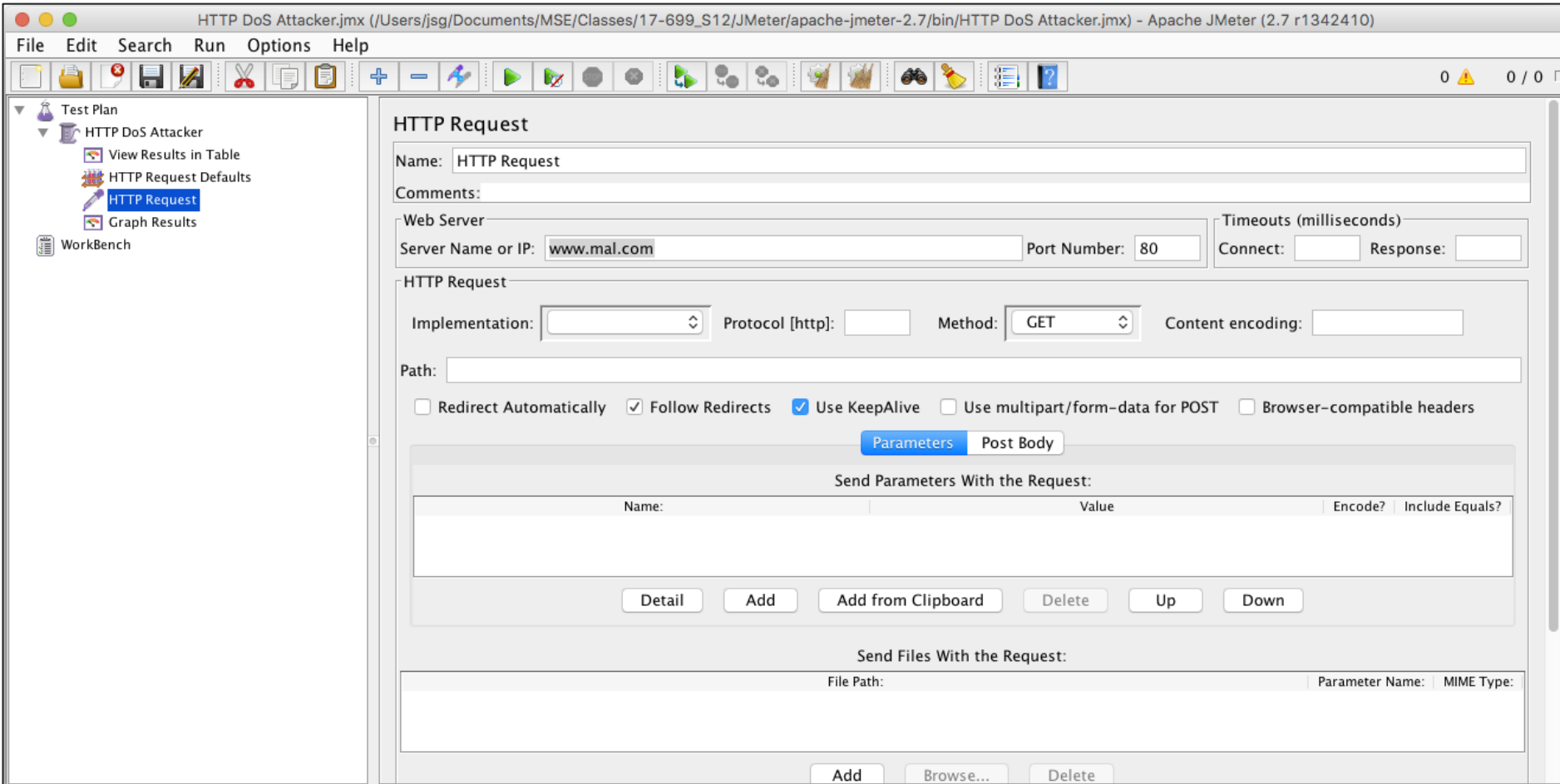
# Testing with Stubs



```
class ConnectionError implements FacebookInterface {  
    List<Node> getPersons(String name) {  
        throw new HttpConnectionException();  
    }  
}
```

```
@Test void testConnectionError() {  
    assert getFriends(new ConnectionError()) == null;  
}
```

# Performance testing tools: JMeter



<http://jmeter.apache.org>

# AB testing

- Act now! Sale ends soon!



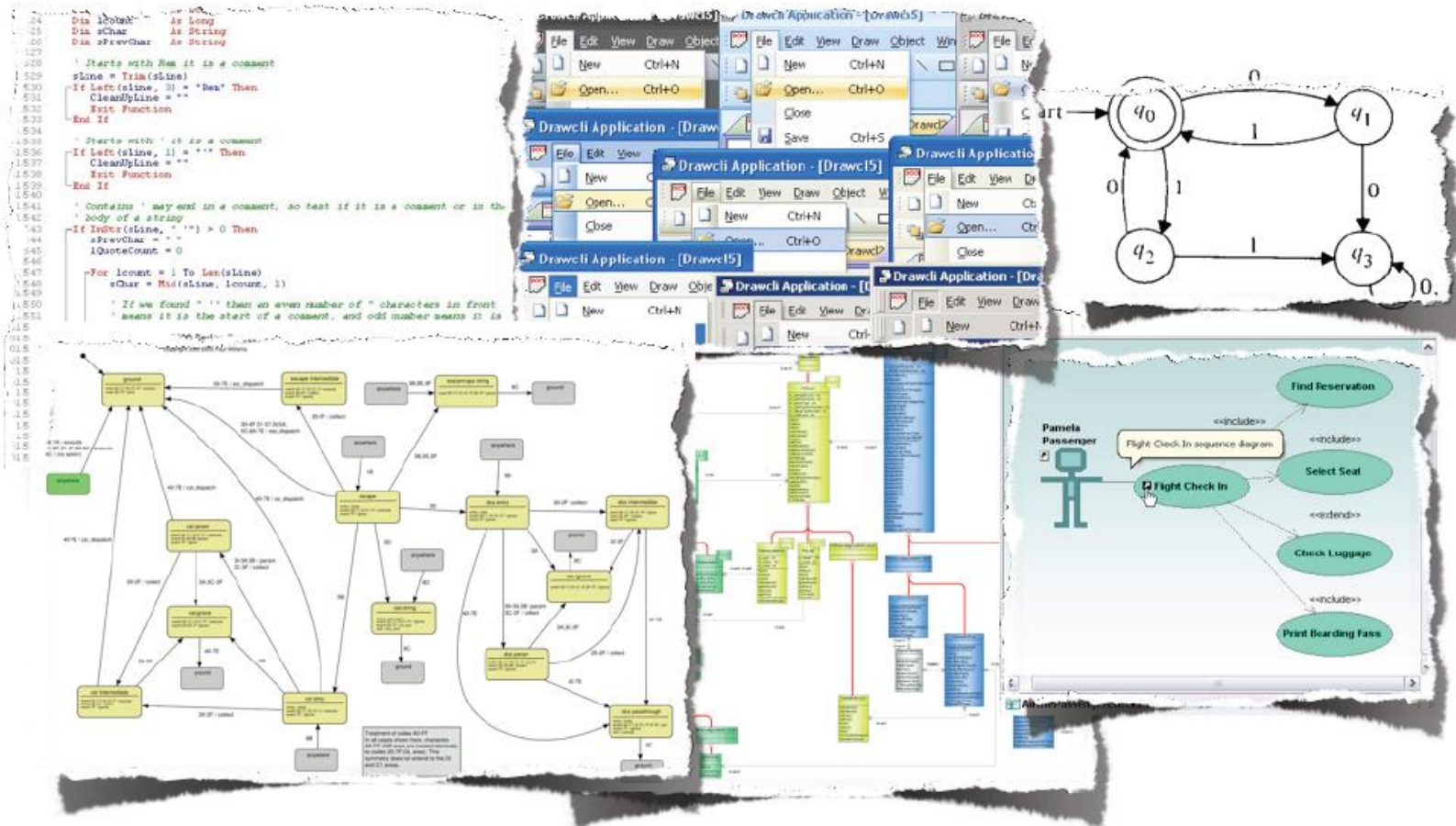


# Random testing

- Select inputs independently at random from the program's input domain:
  - Identify the input domain of the program.
  - Map random numbers to that input domain.
  - Select inputs from the input domain according to some probability distribution.
  - Determine if the program achieves the appropriate outputs on those inputs.
- Random testing can provide probabilistic guarantees about the likely faultiness of the program.
  - E.g., Random testing using  $\sim 23,000$  inputs without failure ( $N = 23,000$ ) establishes that the program will not fail more than one time in  $10,000$  ( $F = 10^4$ ), with a confidence of 90% ( $C = 0.9$ ).



# We can measure coverage on almost anything



A. Zeller, Testing and Debugging Advanced course, 2010

# Limits of Testing

- Cannot find bugs in code not executed, cannot assure absence of bugs
- Oracle problem
- Nondeterminism, flaky tests
  - Certain kinds of bugs occur only under very unlikely conditions
- Hard to observe/assert specifications
  - Memory leaks, information flow, ...
- Potentially expensive, long run times
- Potentially high manual effort
- Verification, not validation
- ...

# HW4: Testing Quality Attributes

# Foundations of Software Engineering

Part 15: Inspections and Reviews

Michael Hilton

# Learning Goals

- Understand different forms of peer reviews with different formality levels
- Select appropriate review forms for a project
- Conduct an inspection session, aware of common pitfalls and social issues
- Perform code reviews with automated software tools
- Understand the expectations and outcomes of modern peer reviews

# Find the Bug(s)!

```
BlockingQueue queue = ...
```

```
while (!queue.isEmpty() && ...) {  
    CheaterFutureTask Task =  
        queue.remove();  
    incompleteTasks.add(Task);  
    taskValues.add(  
        Task.getRawCallable().  
        call());  
}
```

BatchCommitLogExecutorService.java using BlockingQueue in Cassandra,  
one bug injected



### Fix daemon issues caused by Ubuntu's surprising intermediary shell Closed

**Author** epriestley

Press ? to show keyboard shortcuts. ?

**Reviewers** rm, aran, tuomaspelkonen, jungejason, terabyte, puneet

**CCs** aran, epriestley, rm, jcleveley, hugobarauna, feynman, biti, ramk, w31rd0, dleyanlin, taligahack, jiangzhongbo, tomlinsonryan, forrestchu12, davideuler, abekkine, puneet, zakary, lasseespeholt, suwandi.cahyadi, lancelot\_yao, ncu, rafatuita, jacob-zhoupeng, xiaoping, andrei.belyaev, ganesanramkumar, thangtp, jamesjyu, googleyufei, demo, xiaobozi, alpha, jacobcyl, michaelqv, szwedyx, yoel.amram, paprotnik123

**Lint** ★ Lint OK

**Unit** ★ No Unit Test Coverage

**Commits** rPHU3721204cc896: Fix daemon issues caused by Ubuntu's surprising intermediary shell

**Branch** master

**Arcanist Project** libphutil

**Apply Patch** arc patch D212

**Tokens** 🍪

- Subscribe
- Edit Dependencies
- Edit Manifest Tasks
- Herald Transcripts
- Download Raw Diff
- Award Token
- Flag For Later



epriestley summarized this revision.

May 2 2011, 4:56 PM · [D212#summary](#)

On OSX and other Linuxii, `proc_open('./exec_daemon ...')` opens a PHP process; on Ubuntu it opens a "sh -c" process which opens a PHP process. The existence of this surprising shell made everything stop working.

Use 'exec' to replace the shell with the PHP process.



epriestley explained the test plan for this revision.

May 2 2011, 4:56 PM · [D212#test-plan](#)

Ran daemons on OSX and Ubuntu, behavior seems okay in all cases.

Keep in mind I have absolutely no idea how Lunix works so this probably breaks the world. (cc: simpkins)



epriestley commented on this revision.

May 2 2011, 4:57 PM · [D212#1](#)

See [T128](#) for context.



rm accepted this revision.

May 2 2011, 5:13 PM · [D212#2](#)

Nice sleuthing

# Checklists!



## OFFICIAL A.A.F. PILOT'S CHECK LIST

B-17F AND B-17G

For detailed instructions see Pilot's Handbook AN 01-20EF-1 or AN 01-20EG-1 in data case

### PILOT

#### BEFORE STARTING

1. Pilot's Pre-flight — Complete.
2. Form IA, Form F, Weight and Balance — Checked.
3. Controls and Seats — Checked — Checked.
4. Fuel Transfer Valves and Switch — Off.
5. Intercoolers — Cold.
6. Gyros — Uncaged.
7. Fuel Shut-off Switches — Open.
8. Gear Switch — Neutral.
9. Cowl Flaps — Open Right — Open Left — Locked.
10. Turbos — Off.
11. Idle cut-off — Checked.
12. Throttles — Closed.
13. High RPM — Checked.
14. Auto Pilot — Off.
15. De-icers and Anti-icers Wing and Prop. — Off.
16. Cabin heat — Off.
17. Generators — Off.

#### STARTING ENGINES

1. Fire Guard and Call Clear — Left-Right.
2. Master Switches — On.
3. Battery Switches and Inverters — On and Checked.
4. Parking Brakes — Hydraulic Check — On — Checked.
5. Booster Pumps — Pressure — On and Checked.
6. Carburetor Filters — Open.
7. Fuel Quantity — Gallons per tank.
8. Start Engines
  - a. Fire Extinguisher Engine Selector — Checked.
  - b. Prime — As Necessary.

### CO-PILOT

#### BEFORE TAKE OFF

1. Tail Wheel — Locked.
2. Gyro — Set.
3. Generators — On.

#### AFTER TAKE OFF

1. Wheels — Pilot's Signal.
2. Power Reduction.
3. Cowl Flaps.
4. Wheel Check — OK Right. OK Left.

#### BEFORE LANDING

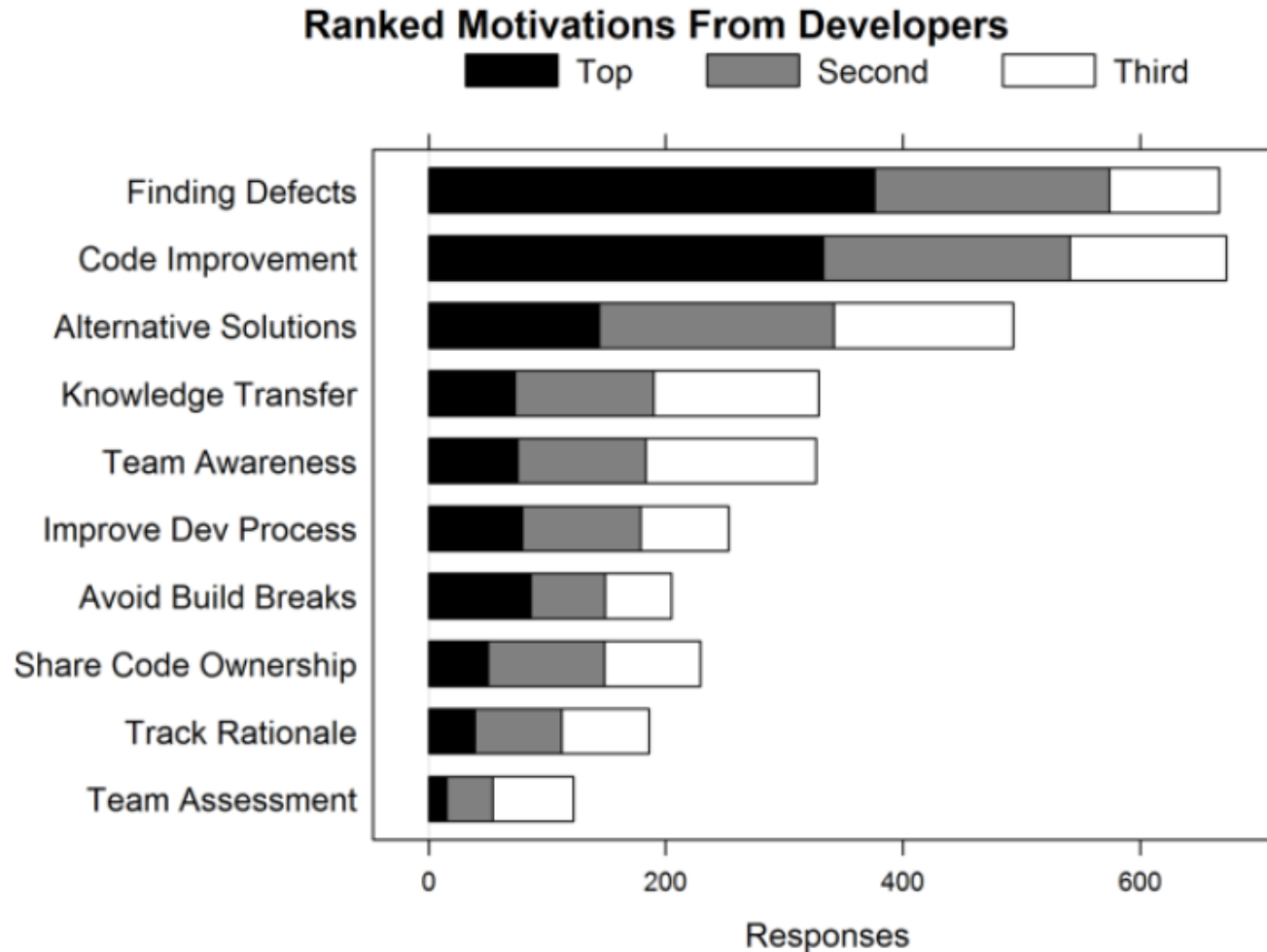
1. Radio Call Altimeter — Set.
2. Crew Positions — OK.
3. Auto Pilot — Off.
4. Booster Pumps — On.
5. Mixture Controls — Auto Rich.
6. Intercooler — Set.
7. Carburetor Filters — Open.
8. Wing De-icers — Off.
9. Landing Gear
  - a. Visual — Down right Down left Tail wheel Down, Antenna In
  - b. Light — OK.
  - c. Switch Off — Neutral.
10. Hydraulic Pressure — OK. Valve closed.
11. RPM 2100 — Set.
12. Turbos — Set.
13. Flaps  $\frac{1}{3}$  —  $\frac{1}{3}$  Down

#### FINAL APPROACH

14. Flaps — Pilot's Signal.
15. High RPM — Pilot's Signal.

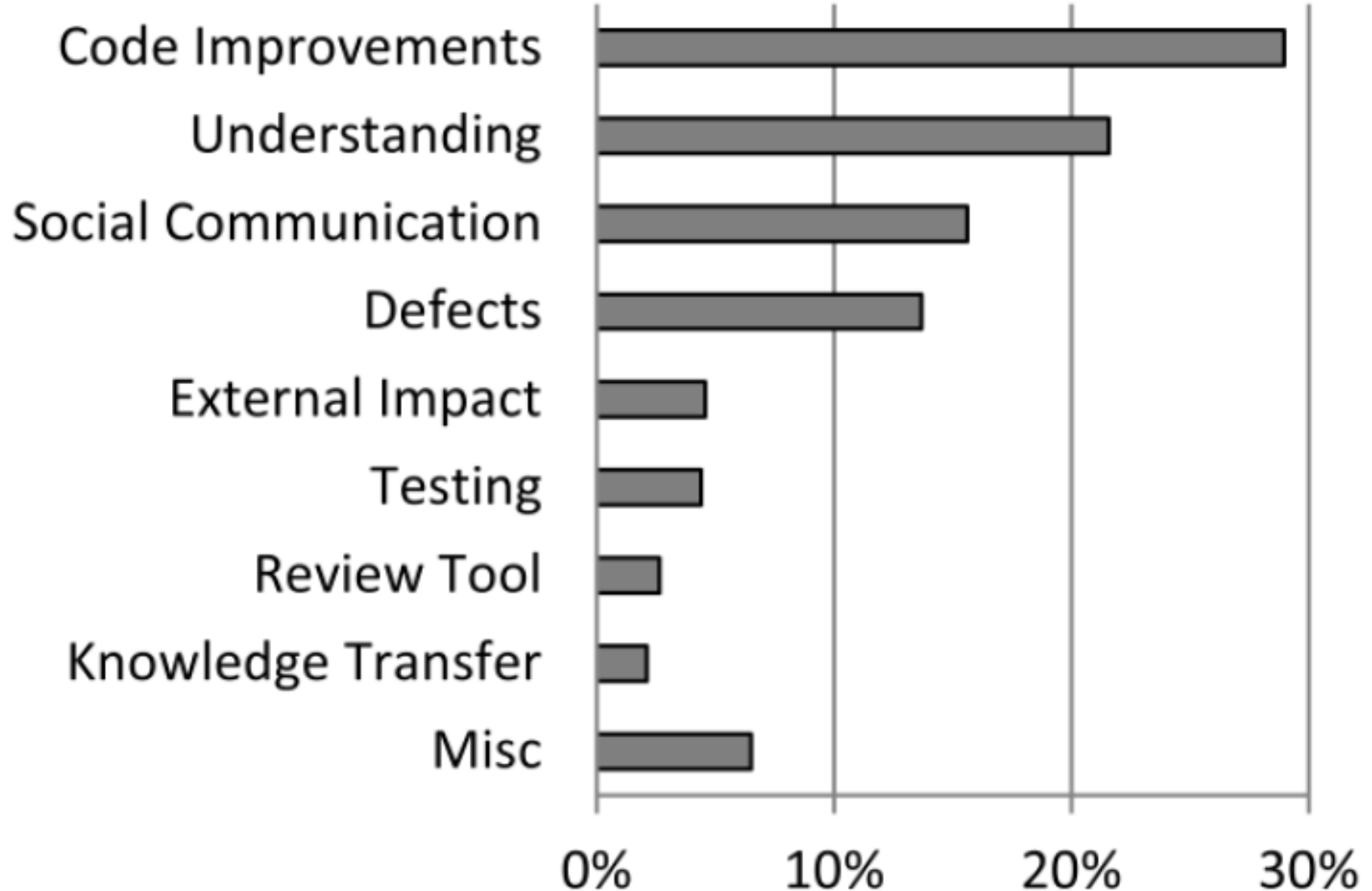
The Checklist: <https://www.newyorker.com/magazine/2007/12/10/the-checklist>

# Code Review at Microsoft



Bacchelli, Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013.

# Outcomes (Analyzing Reviews)

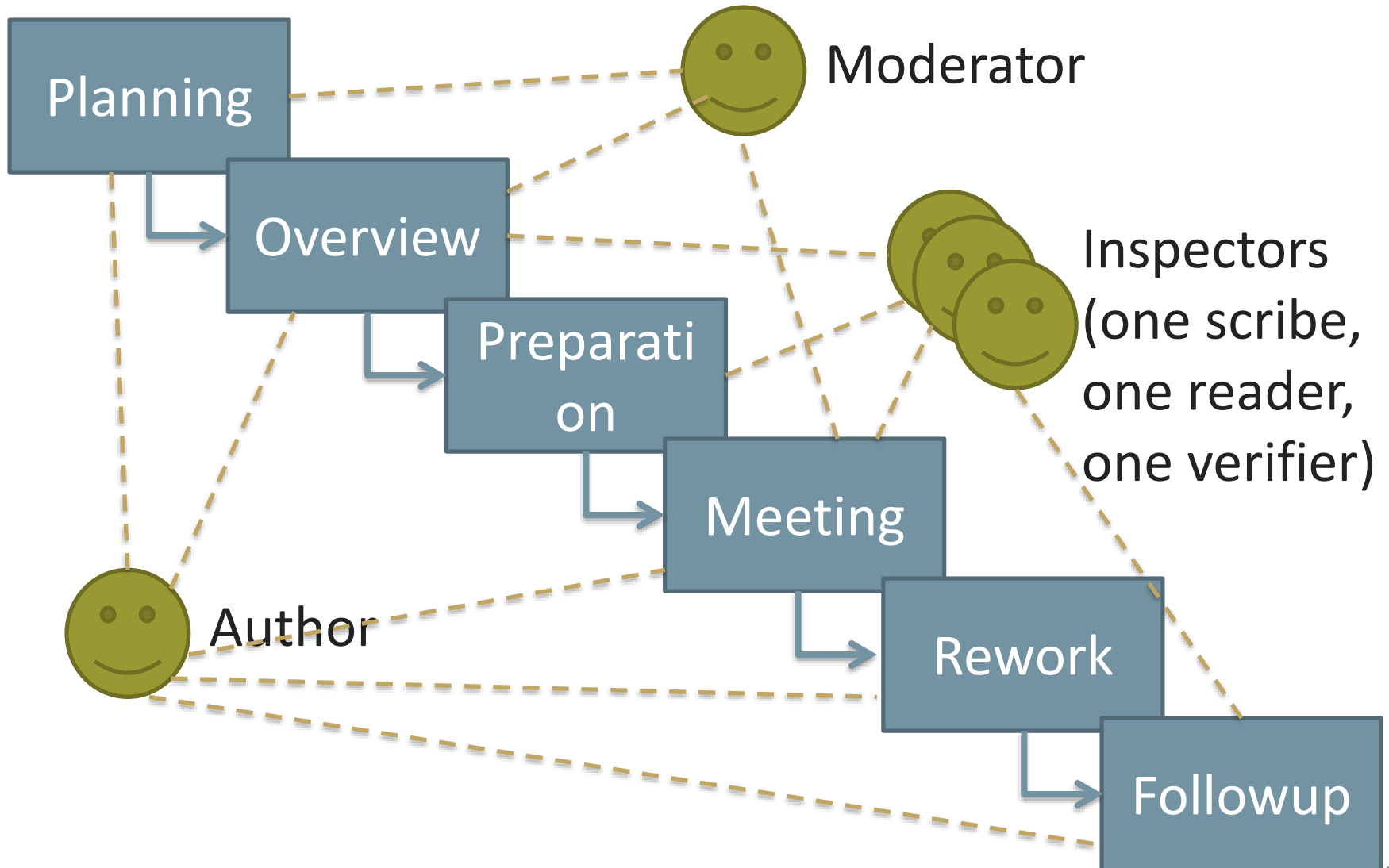


Bacchelli, Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013.

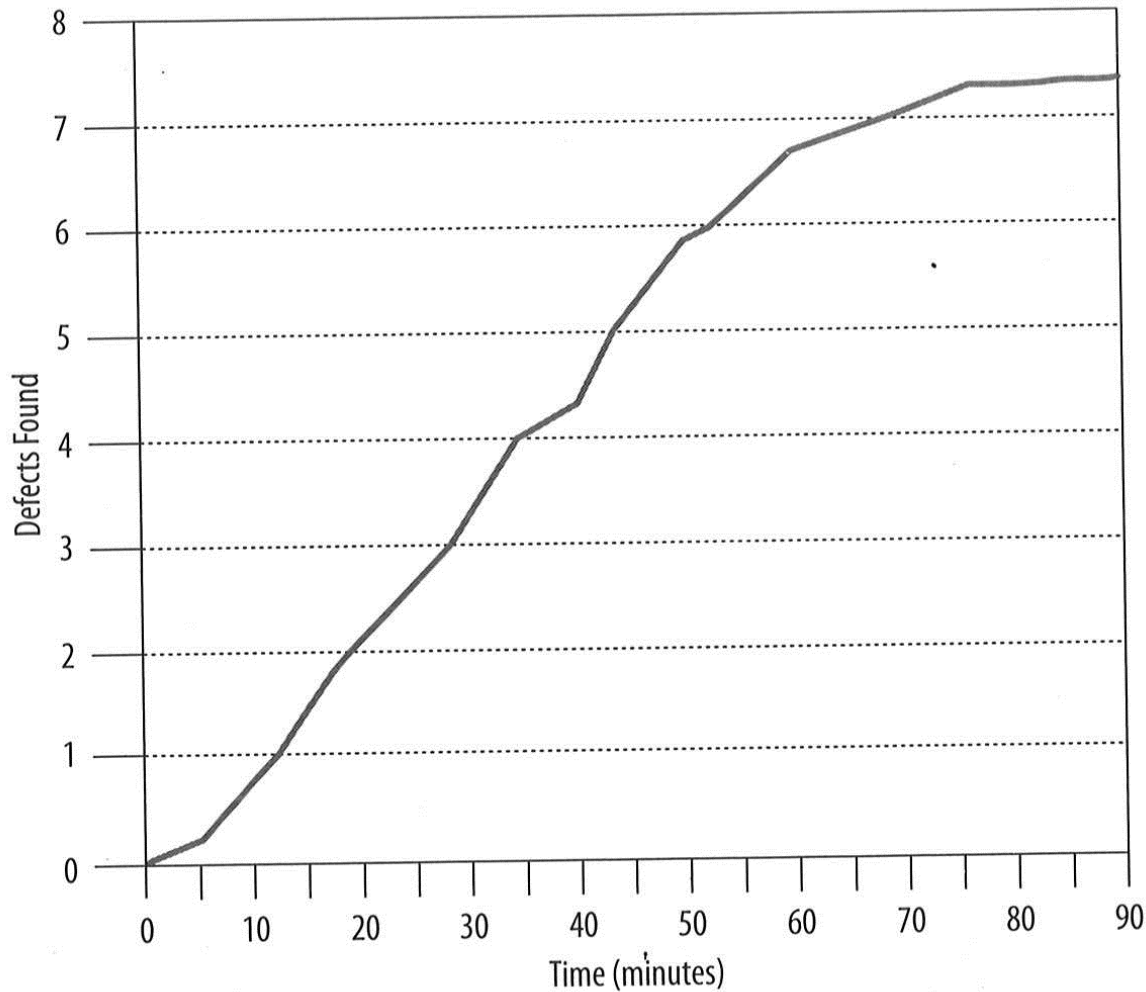
# Code Review at Google

- Introduced to “*force developers to write code that other developers could understand*”
- 3 Found benefits:
  - checking the consistency of style and design
  - ensuring adequate tests
  - improving security by making sure no single developer can commit arbitrary code without oversight

# Inspection Process




# Focus Fatigue



Recommendation:  
Do not exceed  
60 minute session

# Types of Code Reviews by Formality

- 
- Ad hoc review
  - Passaround (“modern code reviews”)
  - Pair programming
  - Walkthrough
  - Inspection

More formal

Source: Wieggers. Peer Reviews in Software. Addison-Wesley 2002



# IsTrueCryptAuditedYet? Yes!

**Update Apr 2, 2015:** [Phase II complete](#). TrueCrypt has been audited.

**Update Feb 18, 2015:** Matthew posted an update on the [Phase II cryptanalysis](#) today. The Phase I audit report [is available](#) on the Open Crypto Audit Project site, and a verified source and download archive for TrueCrypt v. 7.1a can be found on our [GitHub mirror](#). We'll be posting further news [@opencryptaudit](#) on Twitter in the months ahead.

TrueCrypt (TC) is an open source file and disk encryption software package used by people all over the world, but a complete cryptanalysis has not been performed on the software, and questions remain about differences between Windows, Linux and Mac OS X versions. In addition, there has been no legal review on the current TrueCrypt v. 3.0 open source license - preventing inclusion in most of the free operating systems, including Ubuntu, Debian, RedHat, CentOS and Fedora. We want to be able to trust it, but a fully audited, independently verified repository and software distribution would make us feel better about trusting our security to this software. We're pledging this money to sponsor a comprehensive public audit of TrueCrypt.

## Support the Project

You can help support the Project on [our FundFill site](#), or our new [IndieGoGo site](#) (*note: both funds accept credit cards; FundFill also accepts Bitcoin, while IndieGoGo also takes PayPal & eChecks*).

## Goals

- Resolve license status on the [current \(v. 7.1a\) TrueCrypt source code](#) (license [v. 3.0](#)) copyright & distribution, in order to create a verified, independent version control history repository (signed source and binary)
- Perform and document repeatable, deterministic builds of TC 7.1a from source code for current major operating systems:
  - Windows 7
  - Mac OS X (Lion 10.7 and Mountain Lion 10.8)
  - Ubuntu 12.04 LTS and 13.04, RedHat 6.4, CentOS 6.4, Debian 7.1, Fedora 19
- Conduct a public cryptanalysis and security audit of the TC 7.1a

## Rules

# Foundations of Software Engineering

Dynamic Analysis

Christian Kästner

# Learning goals

- Identify opportunities for dynamic analyses
- Define dynamic analysis, including the high-level components of such analyses, and understand how abstraction applies
- Collect targeted information for dynamic analysis; select a suitable instrumentation mechanism
- Understand limitations of dynamic analysis
- Chose whether, when, and how to use dynamic analysis as part of quality assurance efforts

```
129 | }
130 | }
```

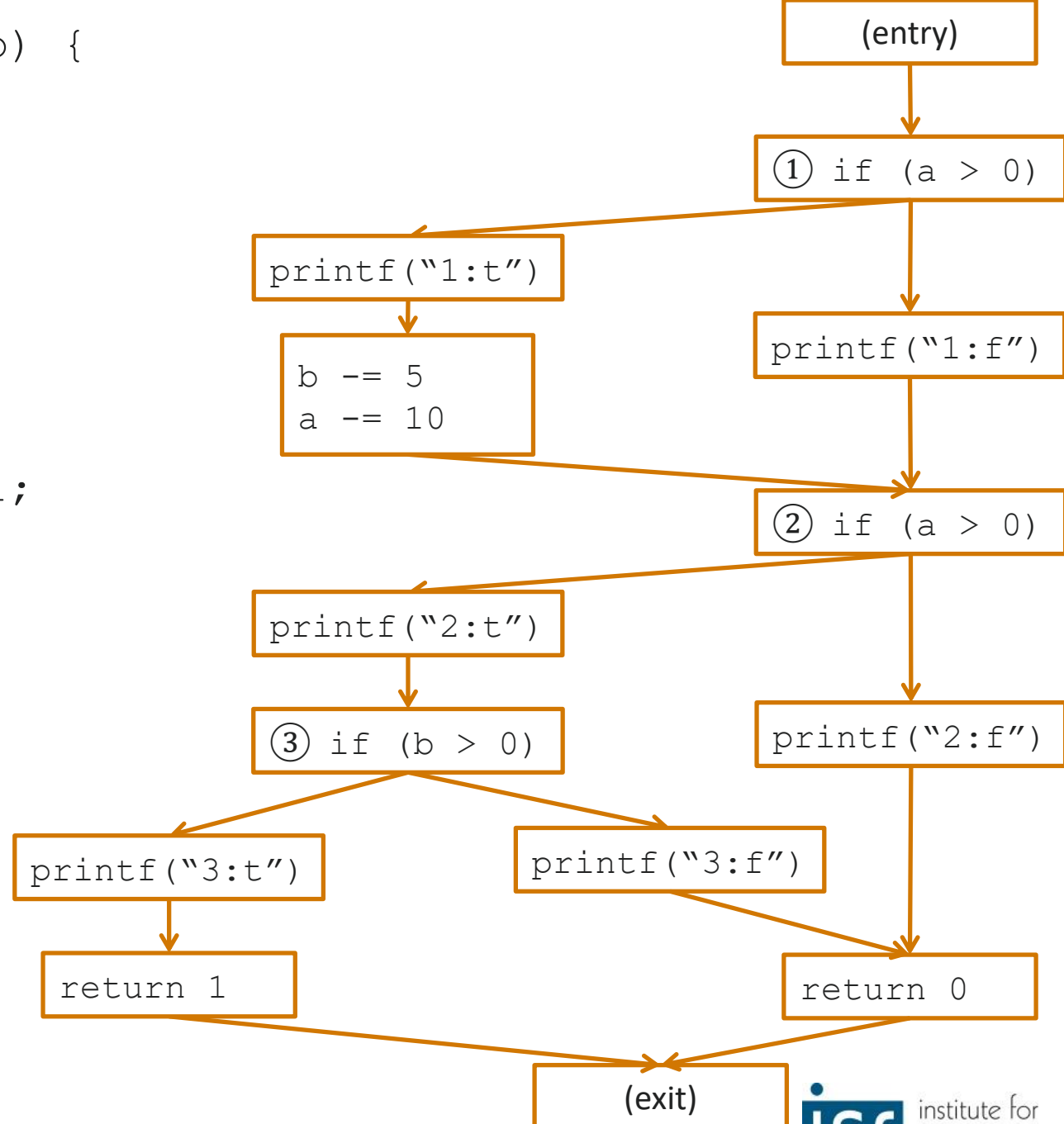
Problems @ Javadoc Declaration Console

<terminated> ClassLoaderLeakExample [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_40.jdk/Contents/Home/bin/java (Oct 20, 2014, 4:15:32 PM)

```
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: Java heap space
```

# What's a memory leak?

```
1. int foobar(a,b) {
2.     if (a > 0) {
3.         b -= 5;
4.         a -= 10;
5.     }
6.     if(a > 0) {
7.         if (b > 0)
8.             return 1;
9.     }
10.    return 0;
11. }
```



# Dynamic Type Checking

```
Object a = getList();  
List<Integer> b = (List<Integer>) a;  
Long c = b.get(1);  
if (random()>0.000000001)  
    System.out.println("foo");  
else  
    Math.max(a, 100);
```

DEBUG

Launch Program



test.js

launch



VARIABLES

Local

```

arguments: Object {__proto__: Object,...
bestCorrectness: -1
bestWeights: undefined
correctness: 0.49
i: 0
testSet: Array[]
testSetCorrectness: undefined

```

WATCH

CALL STACK

PAUSED ON STEP

main	test.js	22
onTimeout	timers.js	365
tryOnTimeout	timers.js	237
listOnTimeout	timers.js	207

BREAKPOINTS

- All Exceptions
- Uncaught Exceptions
- test.js 31

```

9   var trainingSet = generateTestData(0, 100)
10  var testSet = generateTestData(10000, 11000)
11
12  var weights, bestWeights
13  var bestCorrectness = -1
14
15  for (var i=0; i<ITERATIONS; i++){
16    weights = getRandomWeights()
17
18
19    var correctness = getCorrectness(weights, trainingSet)
20
21    if (correctness > bestCorrectness) {
22      debugger
23      bestCorrectness = correctness
24      bestWeights = cloneObject(weights)
25
26      var testSetCorrectness = getCorrectness(weights, testSet)
27
28      console.log("New best correctness in training (test) set:", correctness)
29    }
30  }
31 }
32
33
34
35 function getRandomWeights()

```

DEBUG CONSOLE

```

/Users/mattzeunert/Downloads/node-v7.0.0-nightly201612169538b4aa4f-darwin-x64/bin/node --debug-brk=17011 -TTDebug:log -TTBreakFirst test.js
Debugger listening on 127.0.0.1:17011

```

# Parts of a dynamic analysis

- Property of interest.
- Information related to property of interest.
- Mechanism for collecting that information from a program execution.
- Test input data.
- Mechanism for learning about the property of interest from the information you collected.

*What are you trying to learn about? Why?*

*How are you learning about that property?*

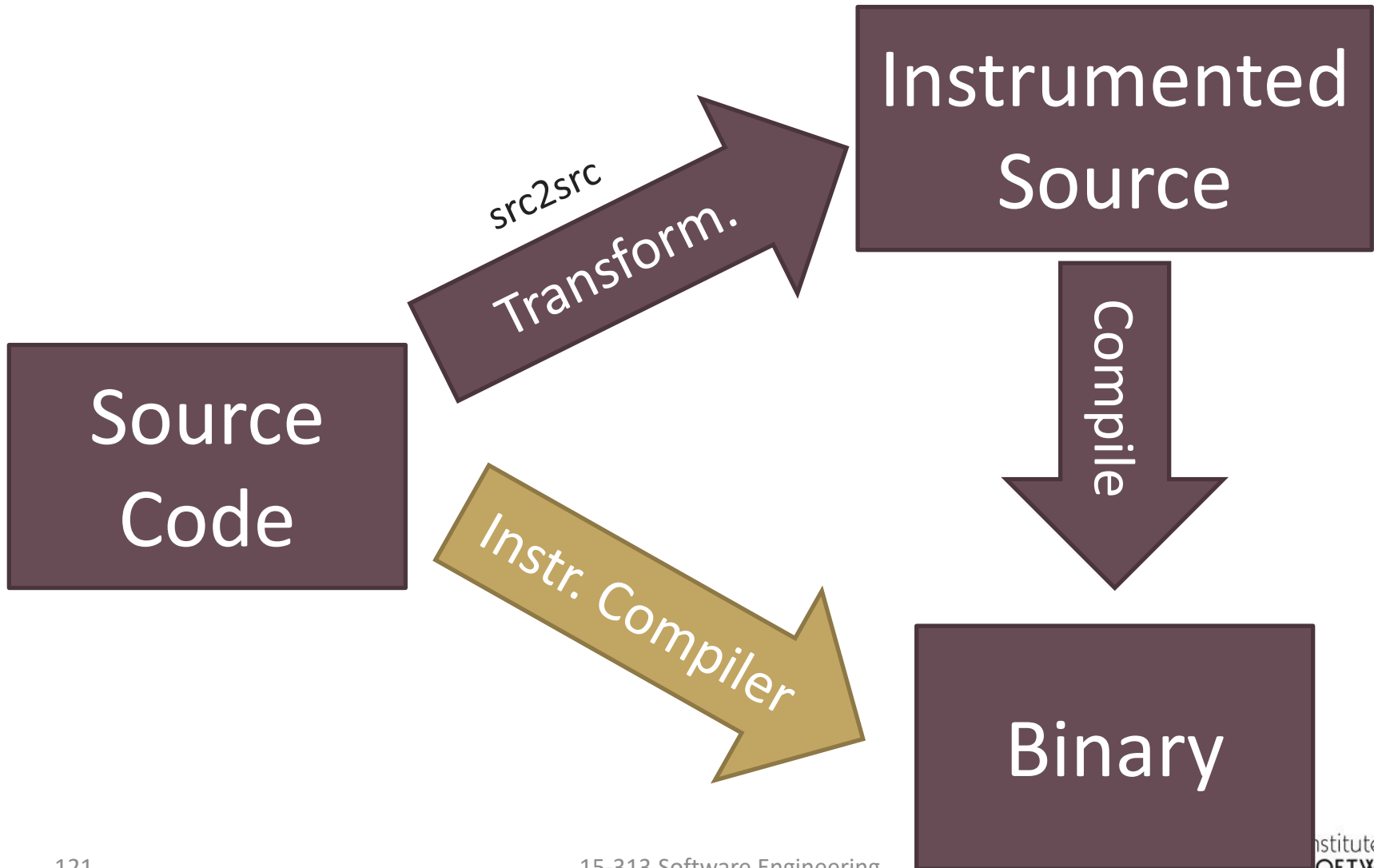
*Instrumentation, etc.*

*What are you running the program on to collect the information?*

*For example: how do you get from the logs to branch coverage?*

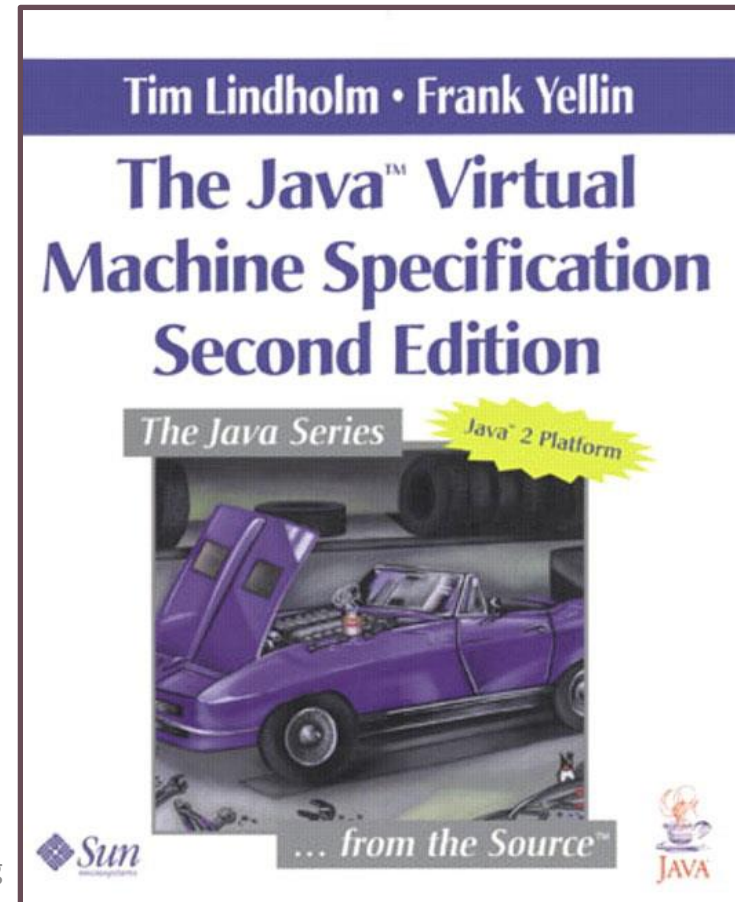


# Code Transformation



# JVM Specification

- <https://docs.oracle.com/javase/specs/>
- See byte code of Java classes with *javap* or ASM Eclipse plugin
- Several analysis/rewrite frameworks as ASM or BECL (internally also used by AspectJ, ...)



	Error exists	No error exists
Error Reported	True positive (correct analysis result)	False positive
No Error Reported	False negative	True negative (correct analysis result)

### Sound Analysis:

- reports all defects
- > no false negatives
- typically overapproximated

### Complete Analysis:

- every reported defect is an actual defect
- > no false positives
- typically underapproximated

# Very input dependent

- Good if you have lots of tests!
  - (system tests are often best)
- Are those tests indicative of normal use
  - And is that what you want?
- Can also use logs from live software runs that include actual user interactions (sometimes, see next slides).
- Or: specific inputs that replicate specific defect scenarios (like memory leaks).

# Foundations of Software Engineering

Static analysis

Christian Kaestner

# Learning goals

- Give a one sentence definition of static analysis. Explain what types of bugs static analysis targets.
- Give an example of syntactic or structural static analysis.
- Construct basic control flow graphs for small examples by hand.
- Distinguish between control- and data-flow analyses; define and then step through on code examples simple control and data-flow analyses.
- Implement a dataflow analysis.
- Explain at a high level why static analyses cannot be sound, complete, and terminating; assess tradeoffs in analysis design.
- Characterize and choose between tools that perform static analyses.

```

1. static OSStatus
2. SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
3.     SSLBuffer signedParams,
4.     uint8_t *signature,
5.     UInt16 signatureLen) {
6.     OSStatus err;
7.     ....
8.     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9.         goto fail;
10.    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11.        goto fail;
12.        goto fail;
13.    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14.        goto fail;
15.    ...
16.fail:
17.    SSLFreeBuffer(&signedHashes);
18.    SSLFreeBuffer(&hashCtx);
19.    return err;
20.}

```

```

1. /* from Linux 2.3.99 drivers/block/raid5.c */
2. static struct buffer_head *
3. get_free_buffer(struct stripe_head * sh,
4.                int b_size) {
5.     struct buffer_head *bh;
6.     unsigned long flags;
7.     save_flags(flags);
8.     cli(); // disables interrupts
9.     if ((bh = sh->buffer_pool, == NULL)
10.        return NULL;
11.     sh->buffer_pool = bh -> b_next;
12.     bh->b_size = b_size;
13.     restore_flags(flags); // re-enables interrupts
14.     return bh;
15. }

```

ERROR: function returns with  
interrupts disabled!

With thanks to Jonathan Aldrich; example from Engler et al., *Checking system rules Using System-Specific, Programmer-Written Compiler Extensions*, OSDI '000



# The Bad News: Rice's Theorem

"Any nontrivial property about the language recognized by a Turing machine is undecidable."

Henry Gordon Rice, 1953

Every static analysis is necessarily incomplete or unsound or undecidable (or multiple of these)

# Syntactic Analysis

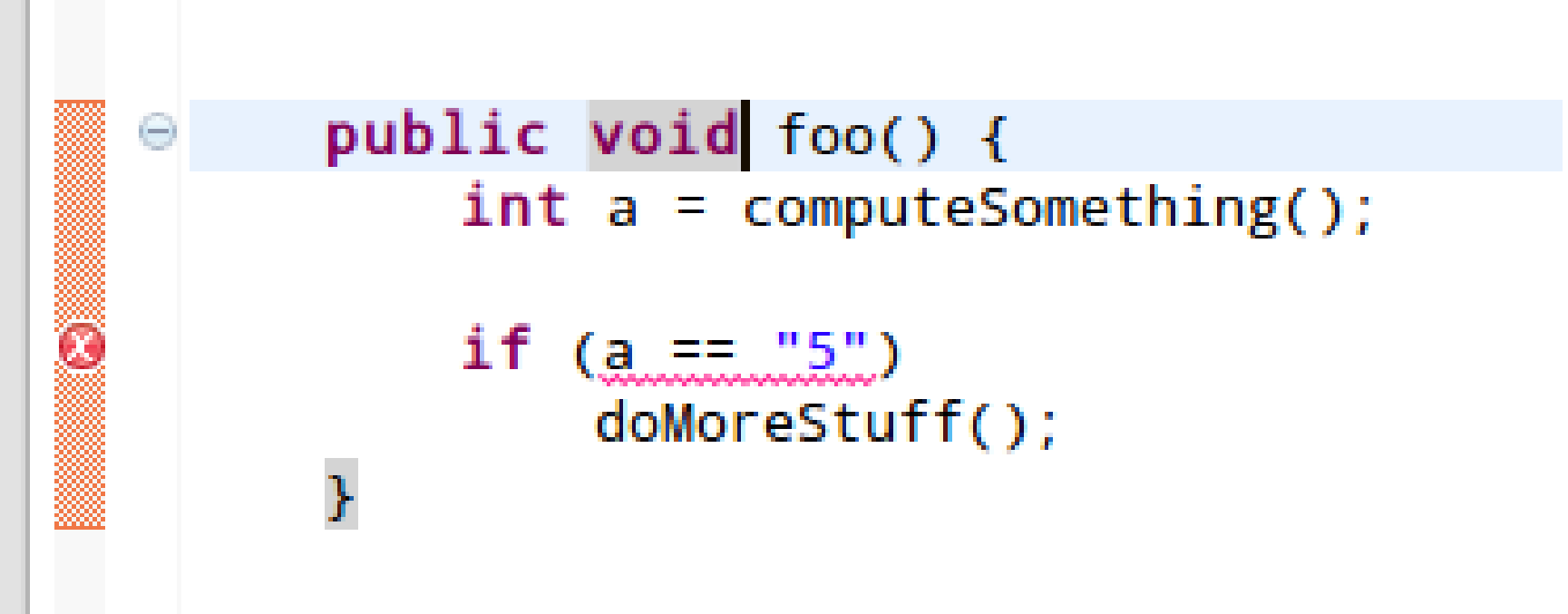
Find every occurrence of this pattern:

```
public foo() {  
    ...  
    logger.debug("We have " + conn + "connections.");  
}
```

```
public foo() {  
    ...  
    if (logger.isDebugEnabled()) {  
        logger.debug("We have " + conn + "connections.");  
    }  
}
```

`grep "if \(\logger\.inDebug" . -r`

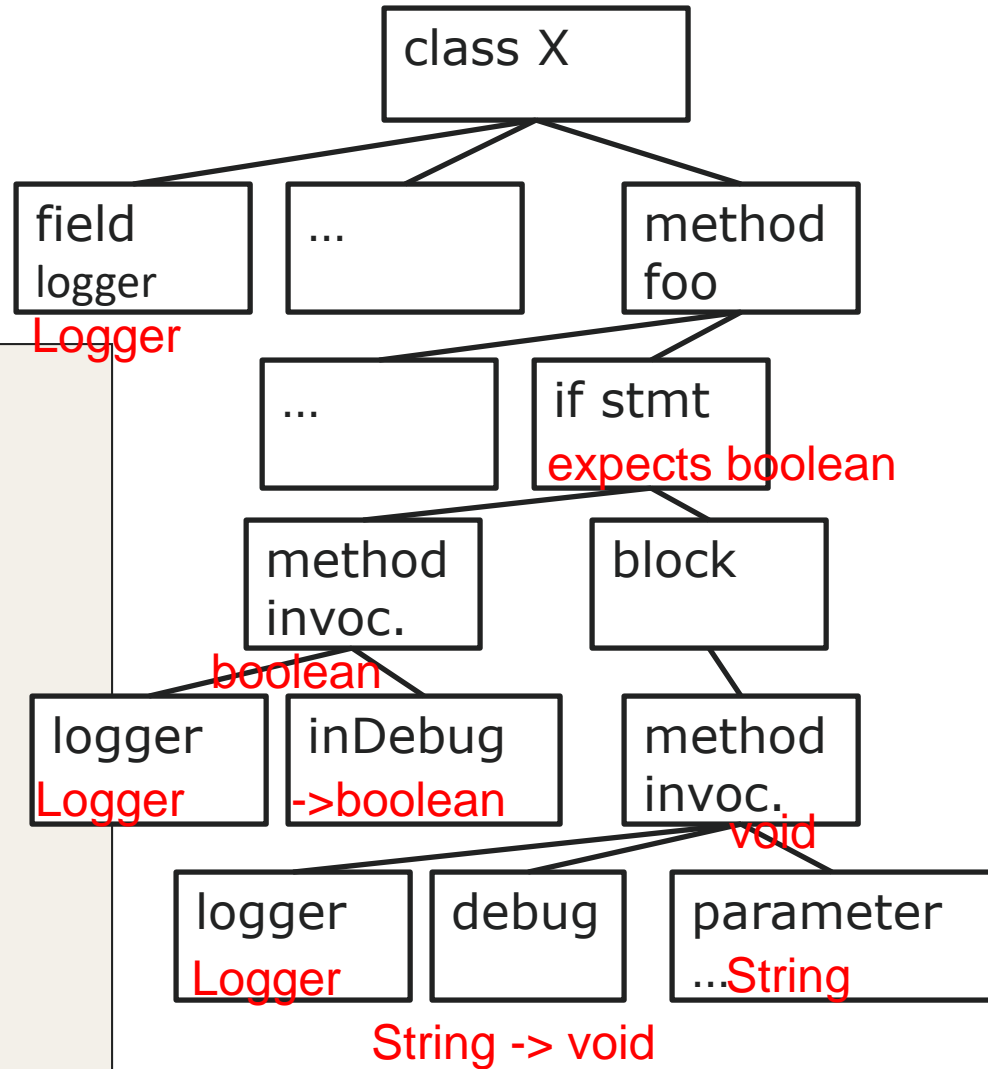
# Type Analysis



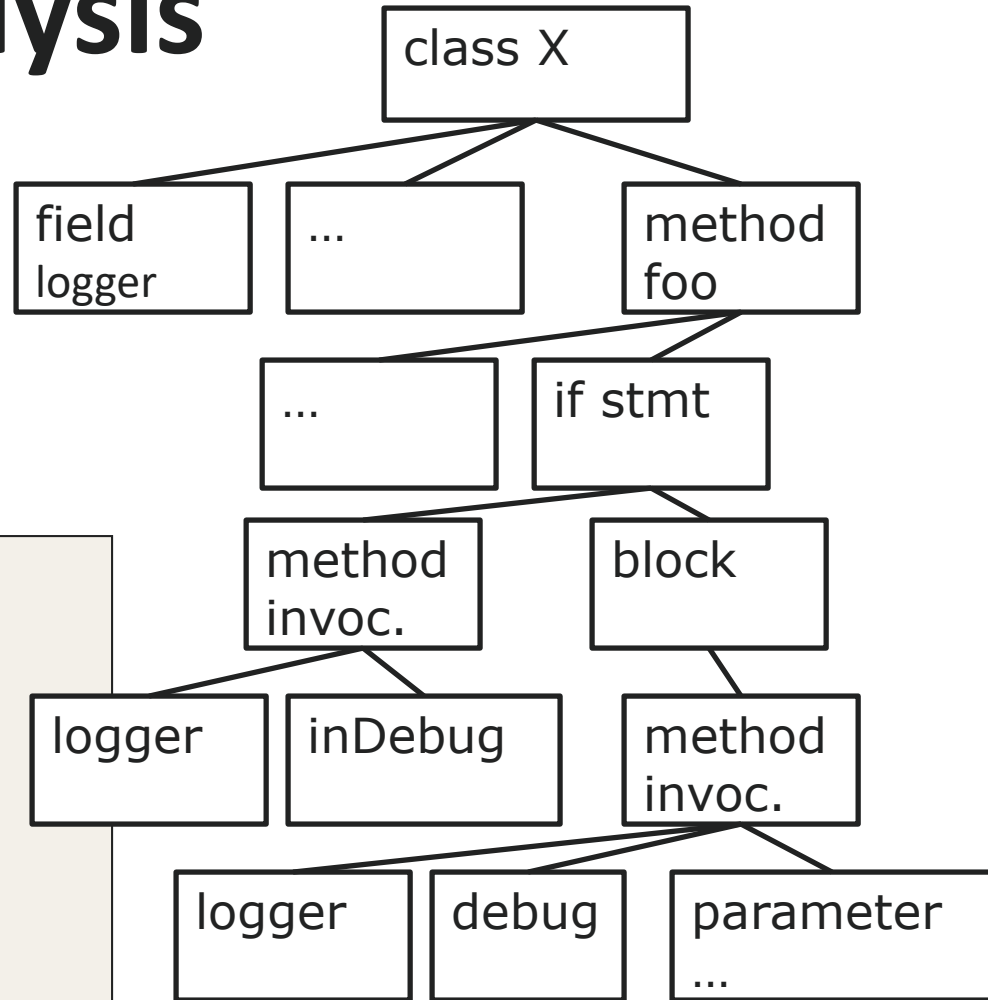
```
public void foo() {  
    int a = computeSomething();  
  
    if (a == "5")  
        doMoreStuff();  
}
```

# Type checking

```
class X {  
  Logger logger;  
  public void foo() {  
    ...  
    if (logger.inDebug()) {  
      logger.debug("We have " +  
conn + "connections.");  
    }  
  }  
}  
class Logger {  
  boolean inDebug() {...}  
  void debug(String msg) {...}  
}
```



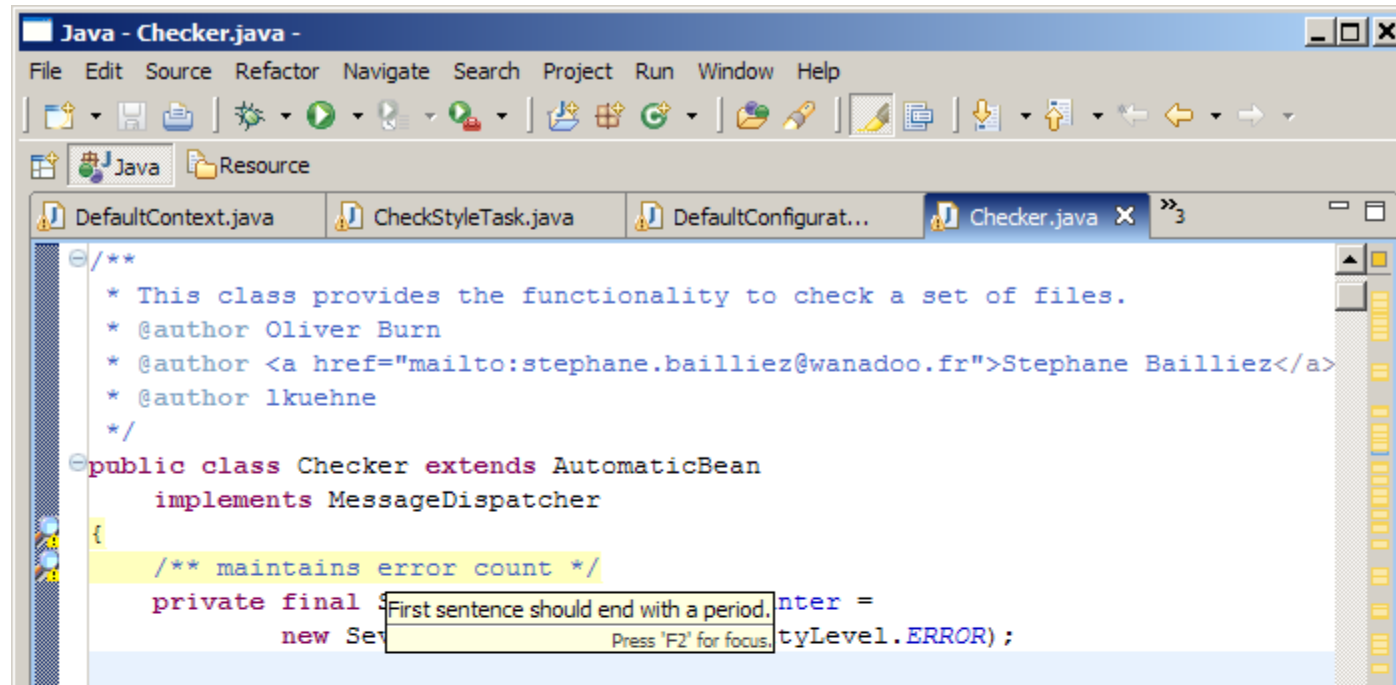
# Structural Analysis



```
class X {  
  Logger logger;  
  public void foo() {  
    ...  
    if (logger.inDebug()) {  
      logger.debug("We have " +  
conn + "connections.");  
    }  
  }  
}
```

# Tools

- Checkstyle
- Many linters (C, JS, Python, ...)
- Findbugs (some analyses)



The screenshot shows an IDE window titled "Java - Checker.java". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development. The editor shows the following code:

```
/**
 * This class provides the functionality to check a set of files.
 * @author Oliver Burn
 * @author <a href="mailto:stephane.bailliez@wanadoo.fr">Stephane Bailliez</a>
 * @author lkuehne
 */
public class Checker extends AutomaticBean
    implements MessageDispatcher
{
    /** maintains error count */
    private final int errorCount;
    private final int errorLevel;

    public Checker() {
        errorCount = 0;
        errorLevel = MessageDispatcher.ERROR;
    }

    public void checkFiles(String[] files) {
        for (String file : files) {
            checkFile(file);
        }
    }

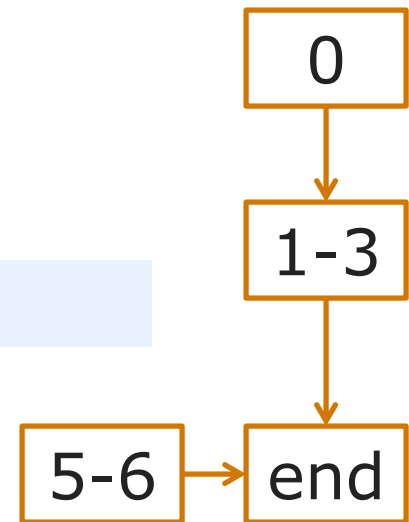
    public void checkFile(String file) {
        // ...
    }
}
```

A tooltip is visible over the line `private final int errorLevel;`, displaying the message: "First sentence should end with a period. Press 'F2' for focus." The tooltip also shows the text `tyLevel.ERROR);` from the line below.

```
public int foo() {
    doStuff();

    return 3;

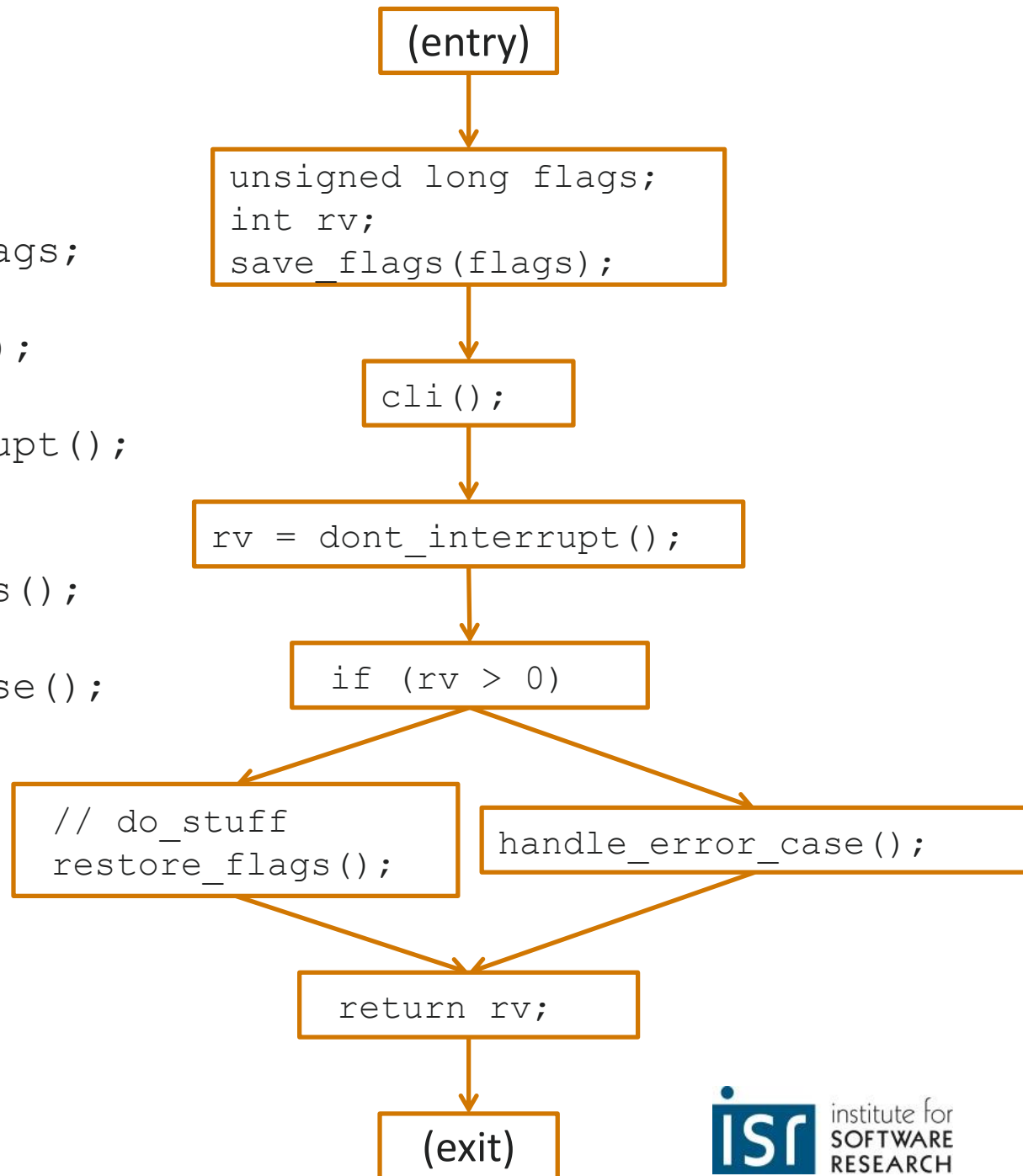
    doMoreStuff();
    return 4;
}
```



```

1.  int foo() {
2.      unsigned long flags;
3.      int rv;
4.      save_flags(flags);
5.      cli();
6.      rv = dont_interrupt();
7.      if (rv > 0) {
8.          // do_stuff
9.          restore_flags();
10.     } else {
11.         handle_error_case();
12.     }
13.     return rv;
14. }

```

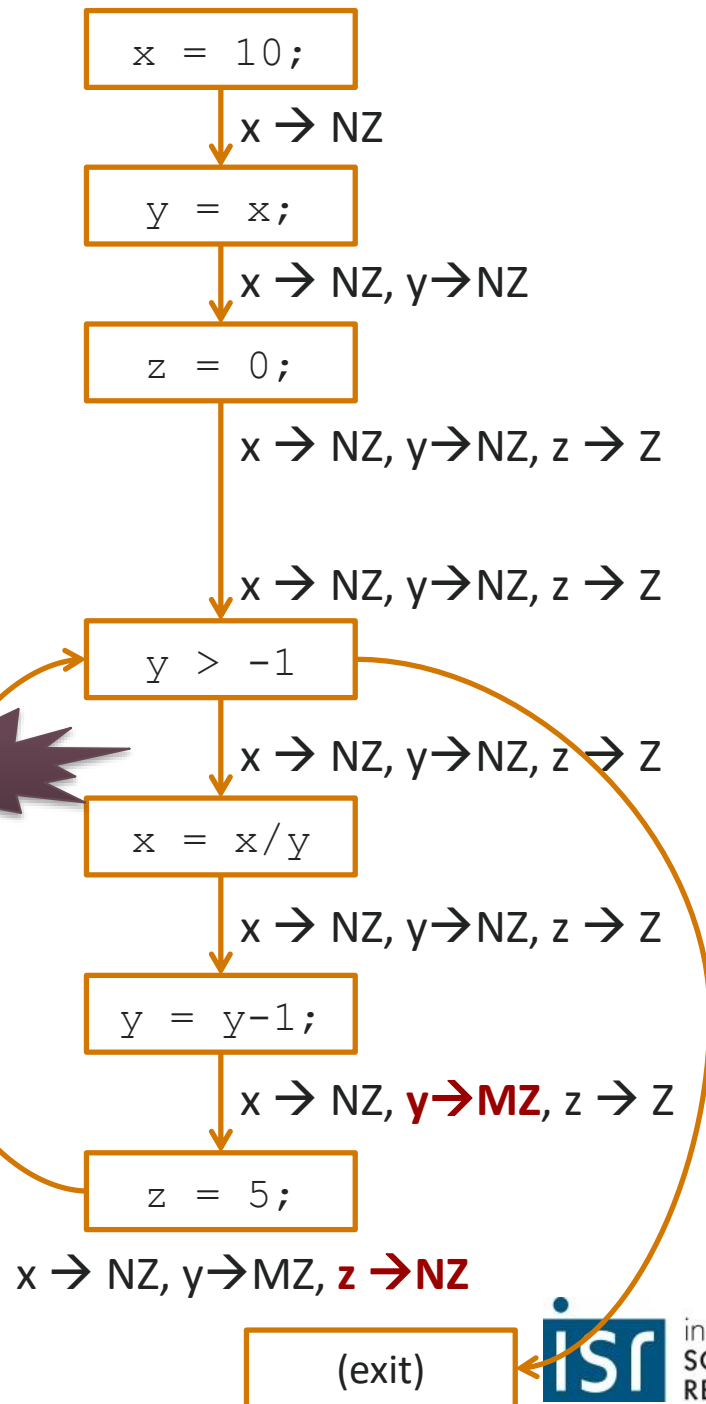




Reminder:

x: Join(NZ,NZ) → NZ  
y: Join(MZ,NZ) → MZ  
z: Join(NZ, Z) → MZ

```
x = 10;  
y = x;  
z = 0;  
while (y > -1) {  
    x = x/y;  
    y = y-1;  
    z = 5;  
}
```



# Abstraction at work

- Number of possible states gigantic
  - n 32 bit variables results in  $2^{32*n}$  states
    - $2^{(32*3)} = 2^{96}$
  - With loops, states can change indefinitely
- Zero Analysis narrows the state space
  - Zero or not zero
  - $2^{(2*3)} = 2^6$
  - When this limited space is explored, then we are done
    - Extrapolate over all loop iterations

# Kildall's Worklist Algorithm

```
1. worklist = new Set();
2. for all node indexes i do
3.   input[i] = ?A;
4.   input[entry] = initialA;
5.   worklist.add(all nodes);
6. while (!worklist.isEmpty()) do
7.   i = worklist.pop();
8.   output = flow(input[i], i);
9.   for j ∈ succ(i) do
10.    if ! (output v input[j])
11.      input = input[j] join output
12.      worklist.add(j)
```

Note on line 5: it's OK to just add entry to worklist if the flow functions cannot return bottom, which is true for our example but not generally.

	Error exists	No error exists
Error Reported	True positive (correct analysis result)	False positive
No Error Reported	False negative	True negative (correct analysis result)

### Sound Analysis:

- reports all defects
- > no false negatives
- typically overapproximated

### Complete Analysis:

- every reported defect is an actual defect
- > no false positives
- typically underapproximated

# Foundations of Software Engineering

Taint Analysis

Miguel Velez

# Learning goals

- Define taint analysis.
- Compare the dynamic and static approaches, as well as their benefits and limitations.
- Apply the analysis to several examples
- Understand how dynamic and static analyses can be combined to overcome the limitations of each other.

# Example

```
1. input = Source();  
2. tmp = “select ...”  
   + input;  
3. tmp = encode(tmp)  
4. Sink(tmp);  
5. log(tmp);
```

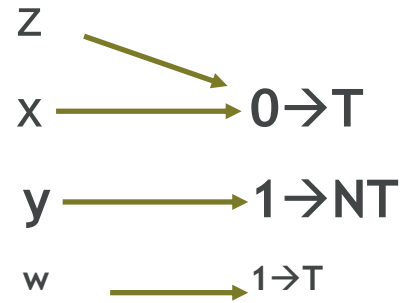
input → priv

tmp → ...

OK

# Example

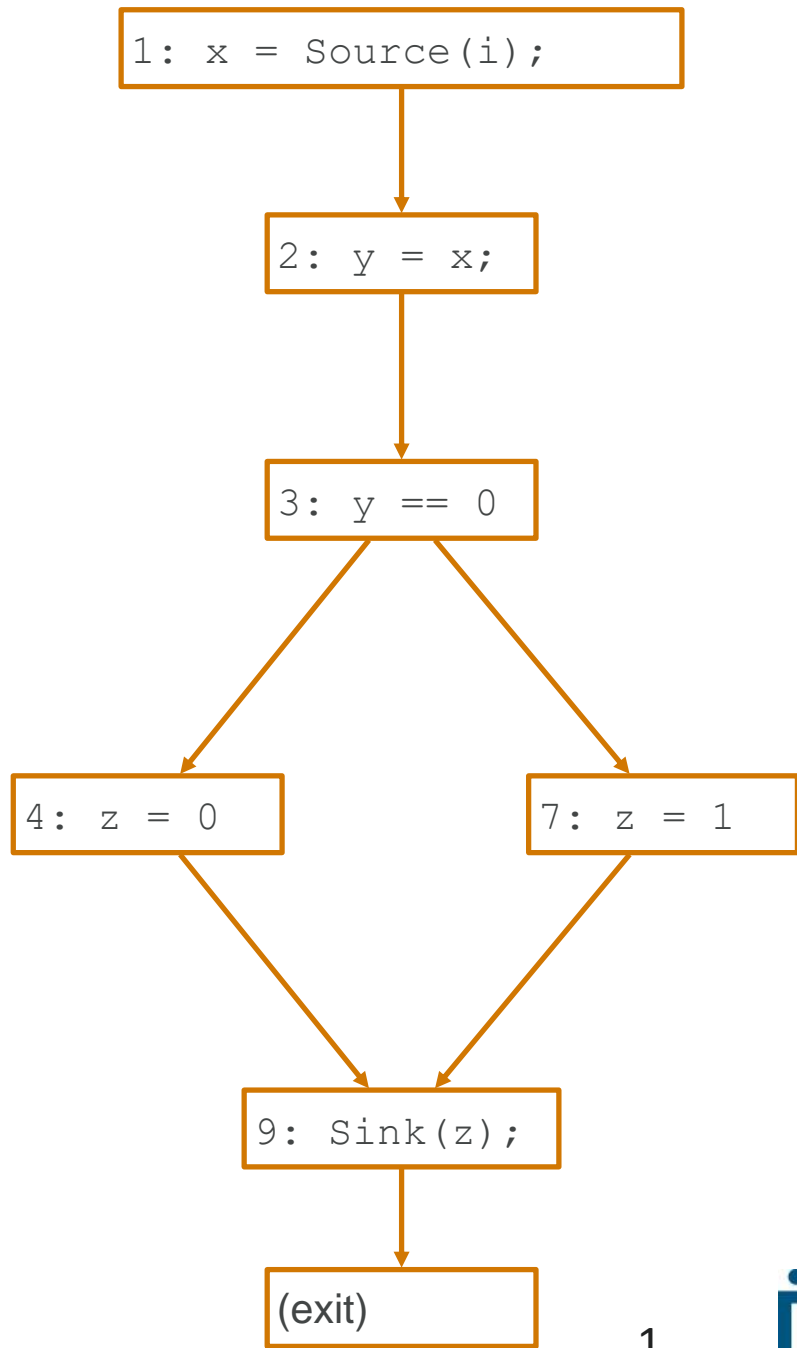
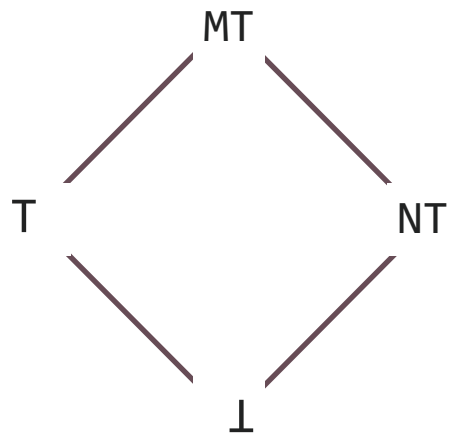
```
1. x =  
   Source(0);  
2. y = 1;  
3. z = x;  
4. w = y + z;  
5. Sink(w);
```



Leak in the program!



```
1. x = Source(i);
2. y = x;
3. if(y == 0) {
4.     z = 0
5. }
6. else {
7.     z = 1
8. }
9. Sink(z);
```



# Kildall's Worklist Algorithm

```
for Instruction i in program
    input[i] =  $\perp$ 
input[firstInstruction] = initialDataflowInformation
worklist = { firstInstruction }

while worklist is not empty
    take an instruction i off the worklist
    output = flow(i, input[i])
    for Instruction j in succs(i)
        if output  $\not\sqsubseteq$  input[j]
            input[j] = input[j]  $\sqcup$  output
            add j to worklist
```

# HW5: Static and Dynamic Analysis

# Foundations of Software Engineering

Process: Linear to Iterative

Michael Hilton

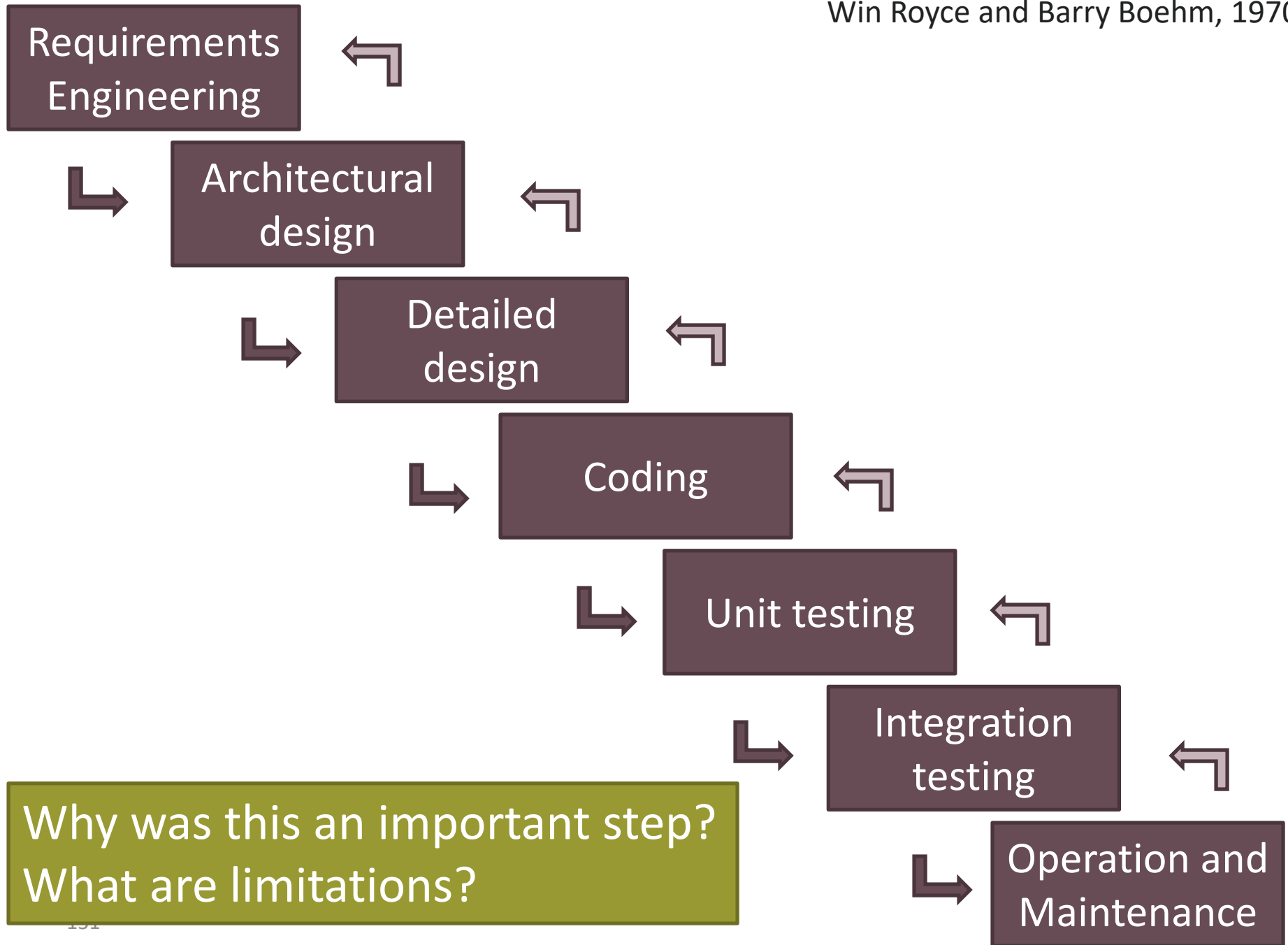
# Learning goals

- Understand the need for process considerations
- Select a process suitable for a given project
- Address project and engineering risks through iteration
- Ensure process quality.

# Interview

- Sean McDonald



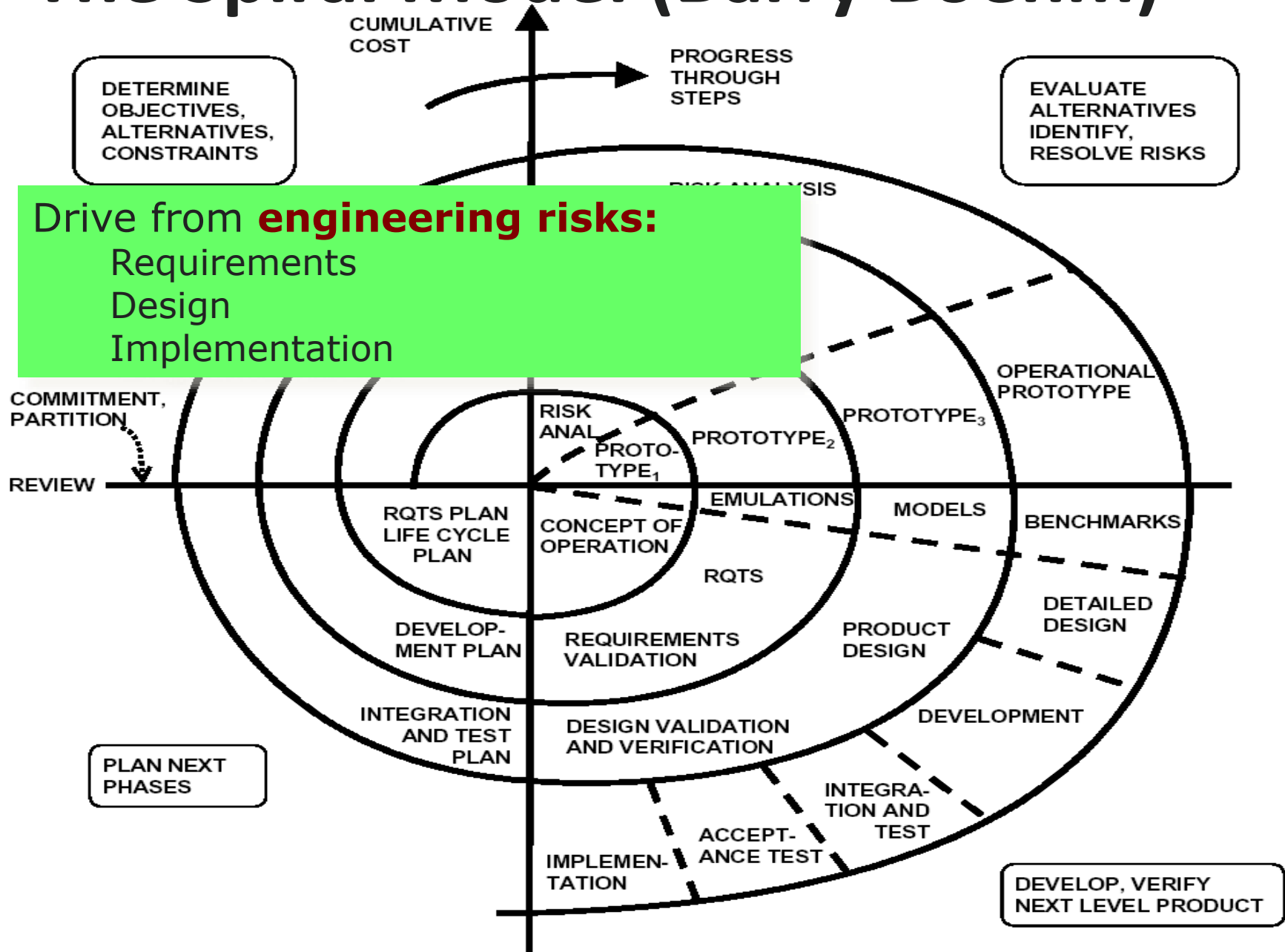




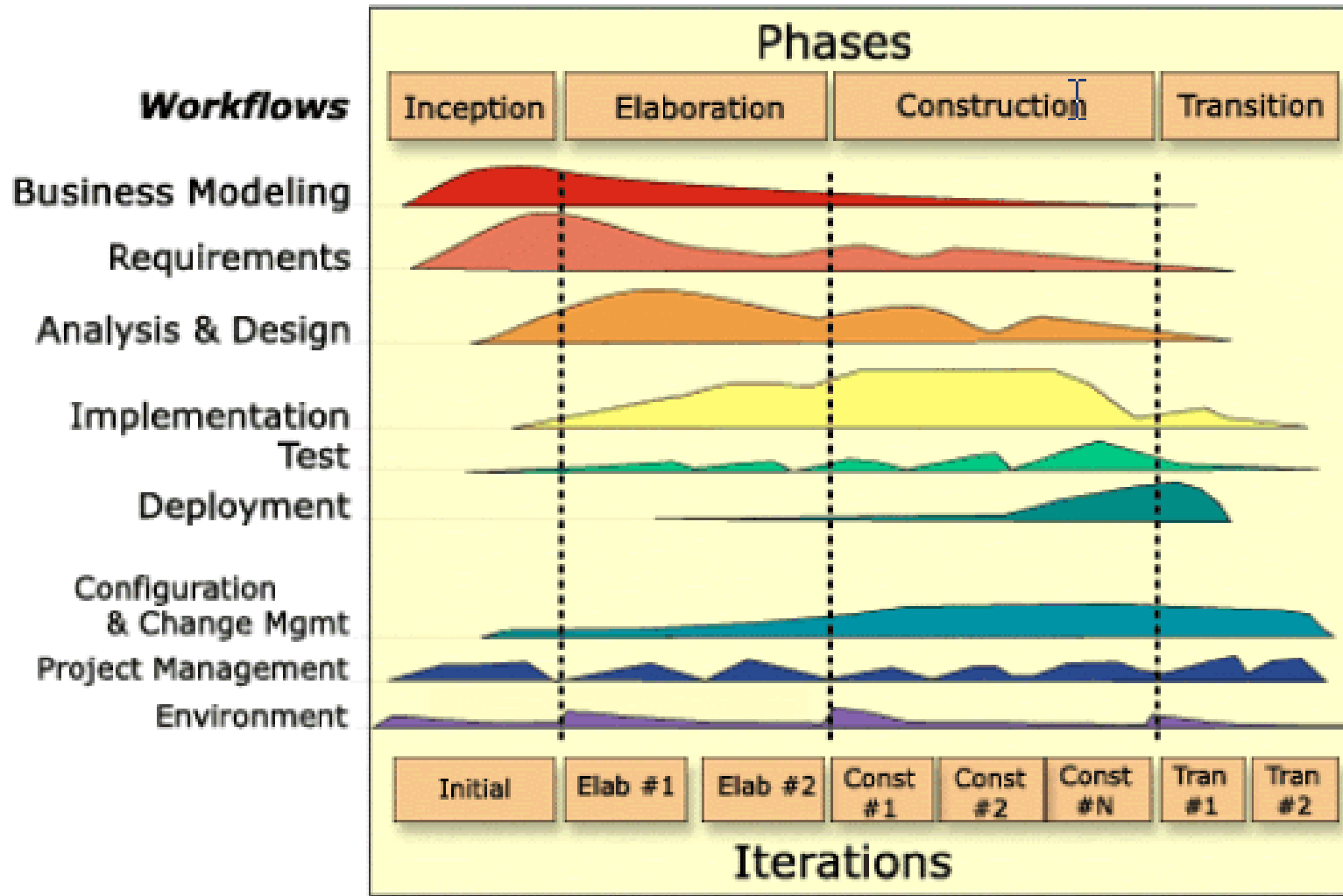
1:32pm  
July 16th 1969



# The Spiral Model (Barry Boehm)

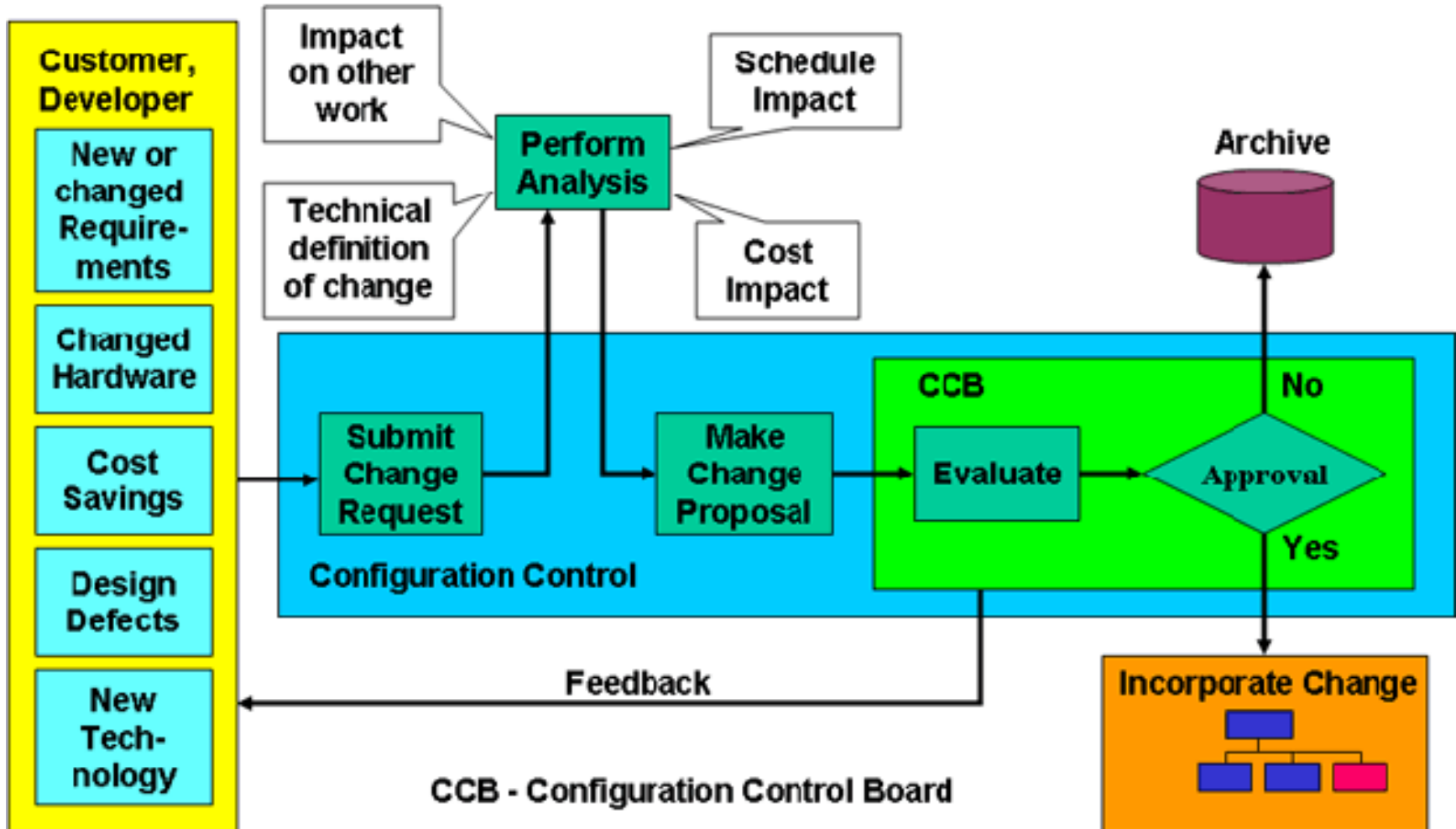


# Rational Unified Process (UP)



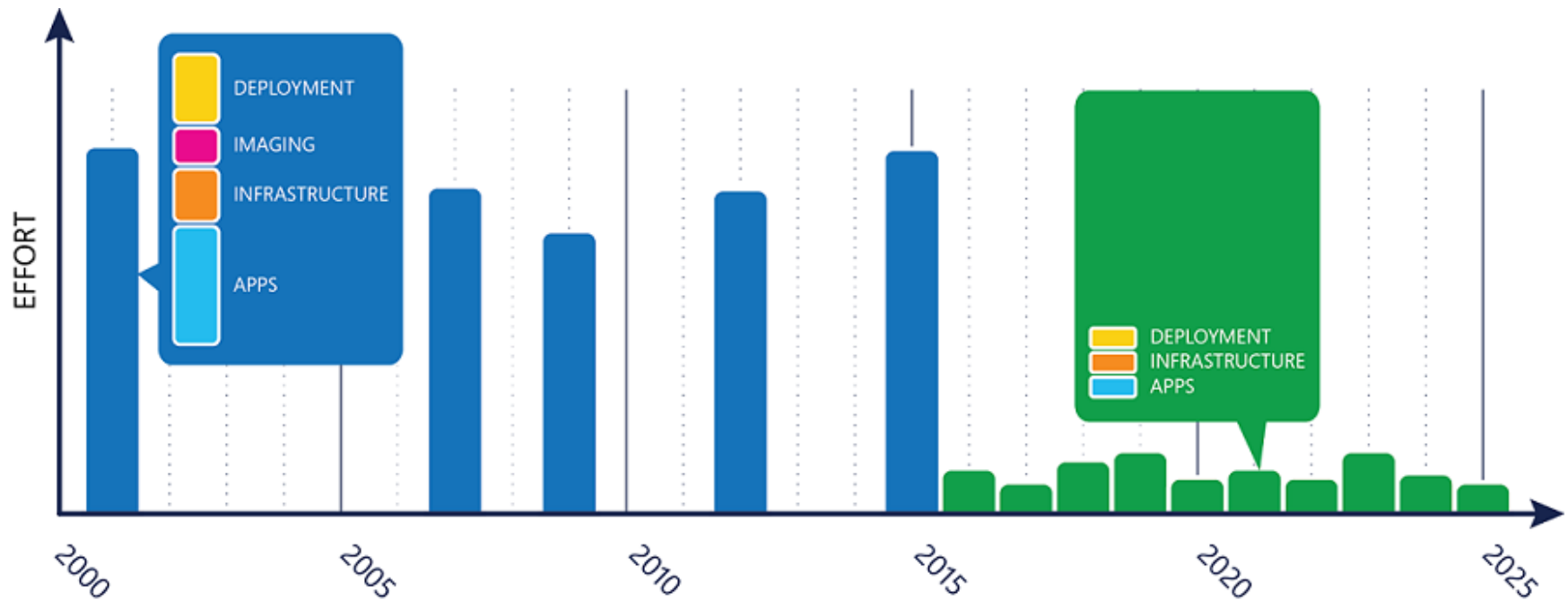
from Rational Software

# Change Control Board



# Prepare servicing strategy for Windows 10 updates

📅 07/26/2017 • ⌚ 6 minutes to read • Contributors 



# Microsoft slows down Windows 10 update pace for businesses following complaints

*Relief for IT admins*

By Tom Warren | @tomwarren | Sep 6, 2018, 11:00am EDT



# Foundations of Software Engineering

Process: Agile Practices

Michael Hilton

# Learning goals

- Define agile as both a set of iterative process practices and a business approach for aligning customer needs with development.
- Explain the motivation behind and reason about the tradeoffs presented by several common agile practices.
- Summarize both scrum and extreme programming, and provide motivation and tradeoffs behind their practices.
- Identify and justify the process practices from the agile tradition that are most appropriate in a given modern development process.

# Brief History of Agile

## ***Inception of Iterative and Incremental Development (IID):***

Walter Shewhart (Bell Labs, signal transmission) proposed a series of “plan-do-study-act” (PDSA) cycles

## ***Introduction of Scrum:***

Jeff Sutherland and Ken Schwaber presented a paper describing the Scrum methodology at a conference workshop

## ***XP reified:*** Kent Beck

released *Extreme*

*Programming Explained:*

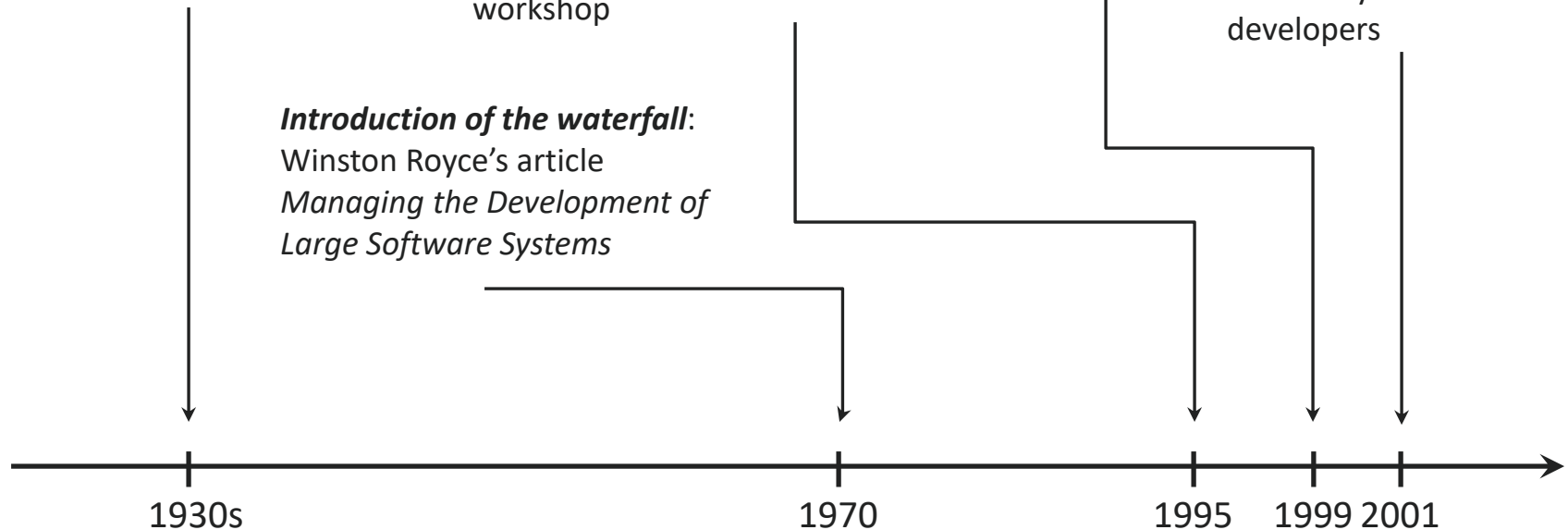
*Embrace Change*

## ***Introduction of “Agile”:***

The Agile Manifesto written by 17 software developers

## ***Introduction of the waterfall:***

Winston Royce’s article *Managing the Development of Large Software Systems*





# 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**

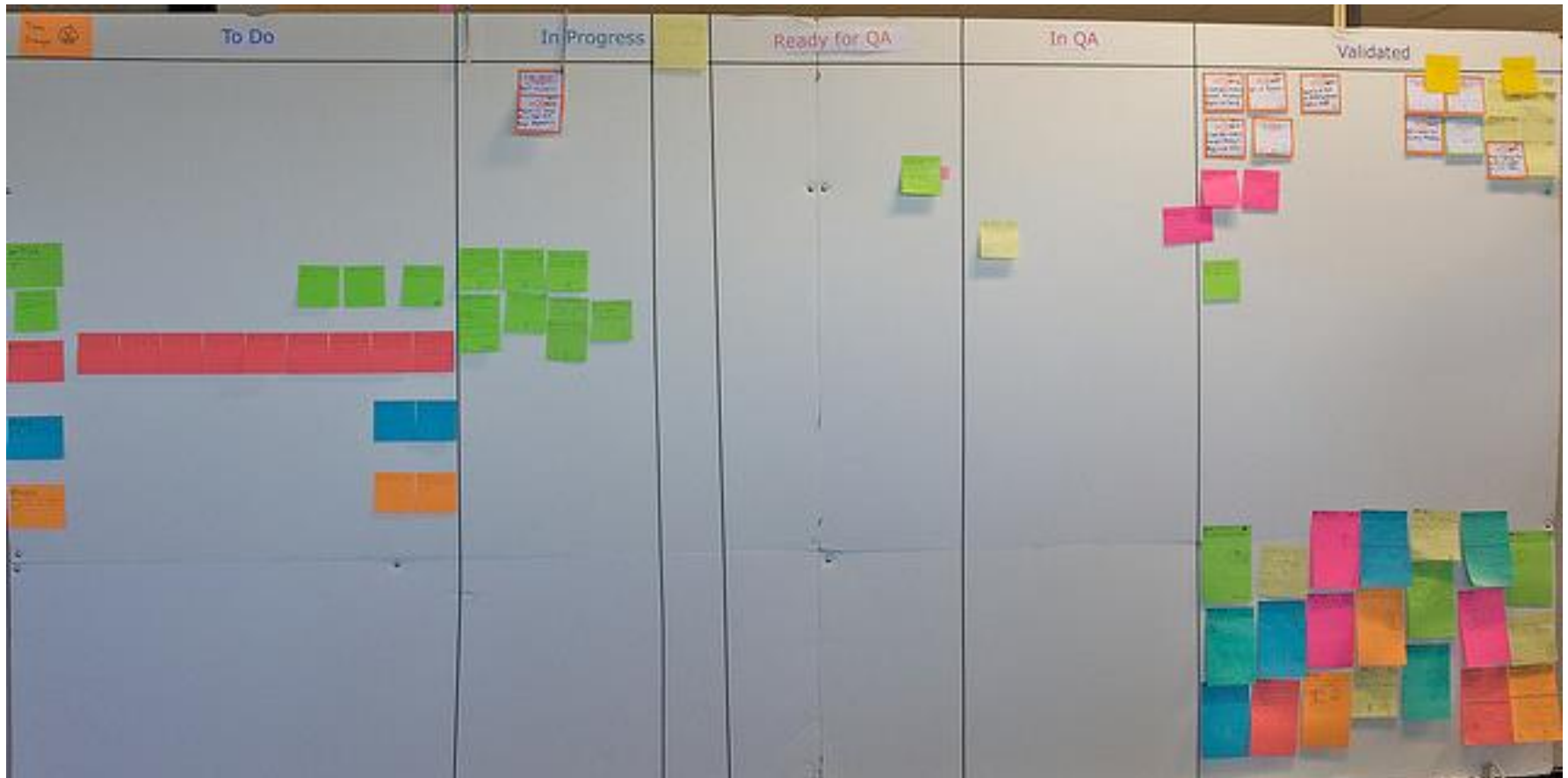
*over*

Following a plan

# Planning Poker



# Kanban Board



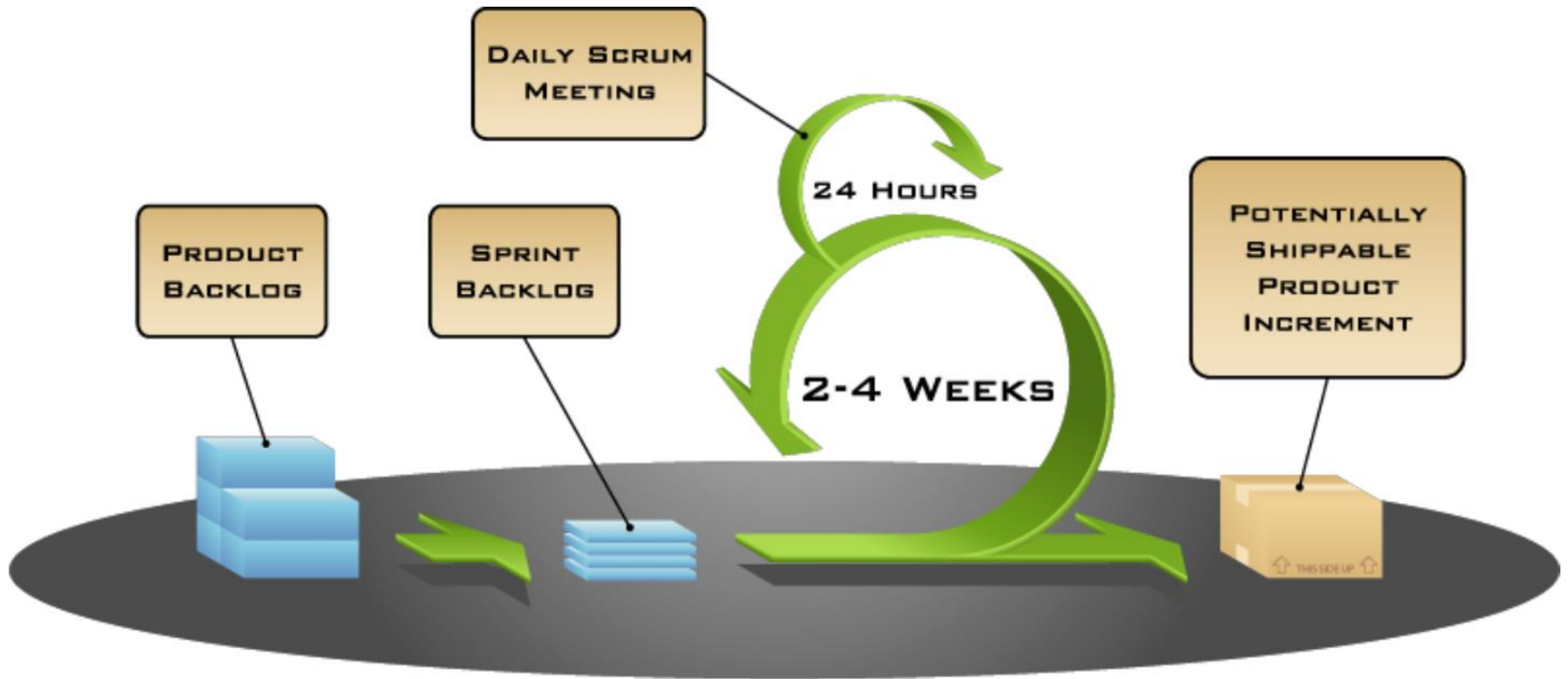
# Pair Programming



# Open workspace



# Scrum Process



Universal Credit

# Case Study

# Foundations of Software Engineering

Quality-Assurance Process

Christian Kästner



# Foundations of Software Engineering

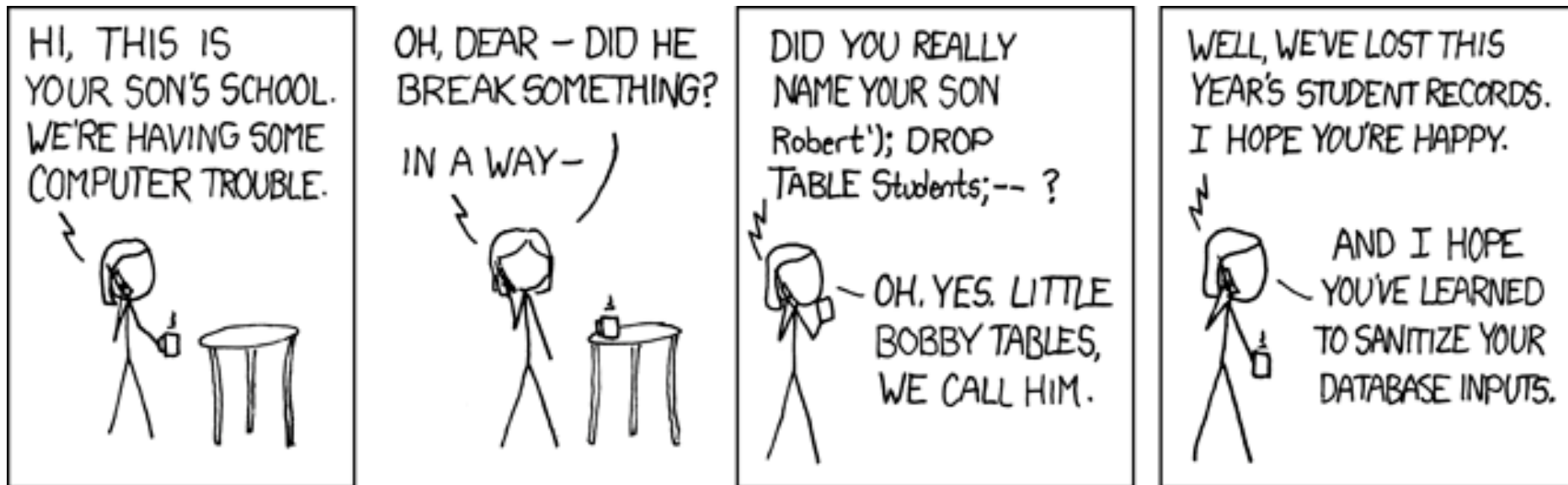
How to get developers to  
[write tests | use static analysis | appreciate testers]

Christian Kästner

# Learning Goals

- Understand process aspects of QA
- Describe the tradeoffs of QA techniques
- Select an appropriate QA technique for a given project and quality attribute
- Decide the when and how much of QA
- Overview of concepts how to enforce QA techniques in a process
- Select when and how to integrate tools and policies into the process: daily builds, continuous integration, test automation, static analysis, issue tracking, ...
- Understand human and social challenges of adopting QA techniques
- Understand how process and tool improvement can solve the dilemma between features and quality

# Example: SQL Injection Attacks



Which QA strategy is suitable?

<http://xkcd.com/327/>

# Example: Scalability



twitter

Home Public Timeline Help

**Twitter is over capacity.**  
Too many tweets! Please wait a moment and try again.

Which QA strategy is suitable?

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

for  
ARE  
:CH

NATIONAL BESTSELLER

"A meticulous accounting of how Microsoft operates...A blueprint for any company...  
facing fast-paced markets and harrowing competition." —Business Week

# Microsoft SECRETS

How the World's Most Powerful Software Company

Creates Technology,

Shapes Markets,

and Manages People

Michael A. Cusumano

Richard W. Selby

WITH A NEW PREFACE BY THE AUTHORS

# 1989 Retreat and “Zero defects”

- see memo

A problem has been detected and windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE\_FAULT\_IN\_NONPAGED\_AREA

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

\*\*\* STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

\*\*\* SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3

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

```
public class Test {
```

▼ **Lint** Missing a Javadoc comment.

Java  
1:02 AM, Aug 21

[Please fix](#)

[Not useful](#)

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

▼ **ErrorProne** String comparison using reference equality instead of value equality  
(see <http://code.google.com/p/error-prone/wiki/StringEquality>)

StringEquality  
1:03 AM, Aug 21

[Please fix](#)

[Not useful](#)

**Suggested fix attached:** [show](#)

```
    }  
  
    public String getString() {  
        return new String("foo");  
    }  
}
```



# Foundations of Software Engineering

Security Development Lifecycles  
Christian Kästner

(Based on slides by Michael Maass)

# Learning goals

- Understand basic concepts of vulnerabilities and secure software
- Implement security mechanisms across the entire software development lifecycle
- Design and inspect architecture for security with threat modeling
- Decide how do adopt security practices and educate participants. Who, when, and how much?

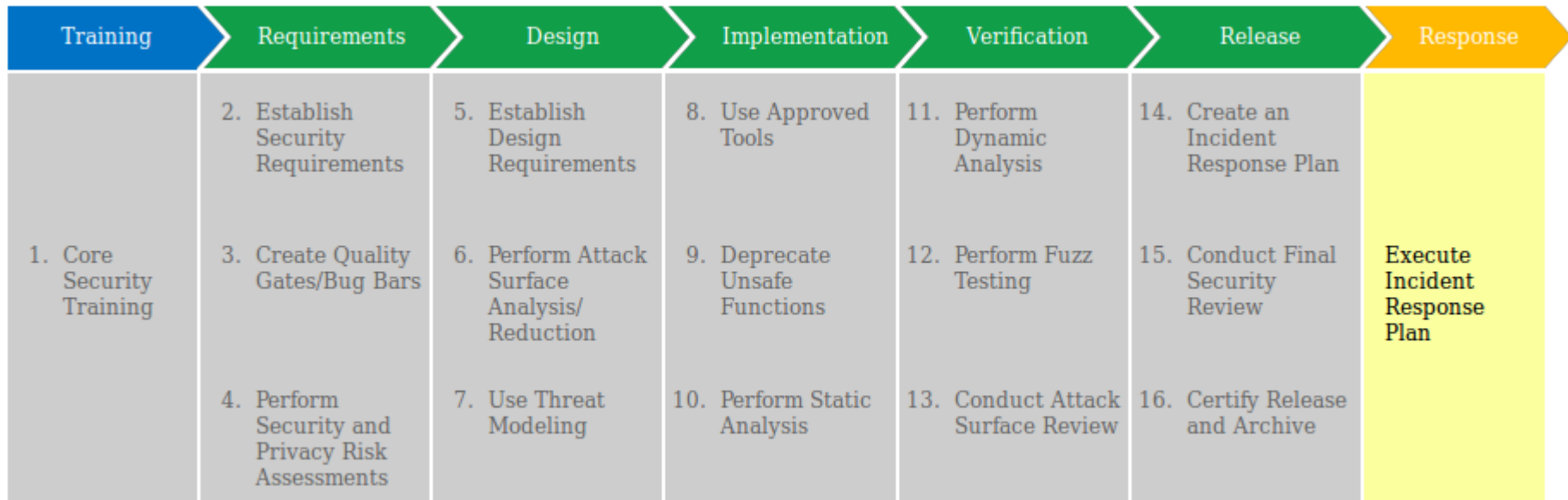
# Sources of Software Insecurity

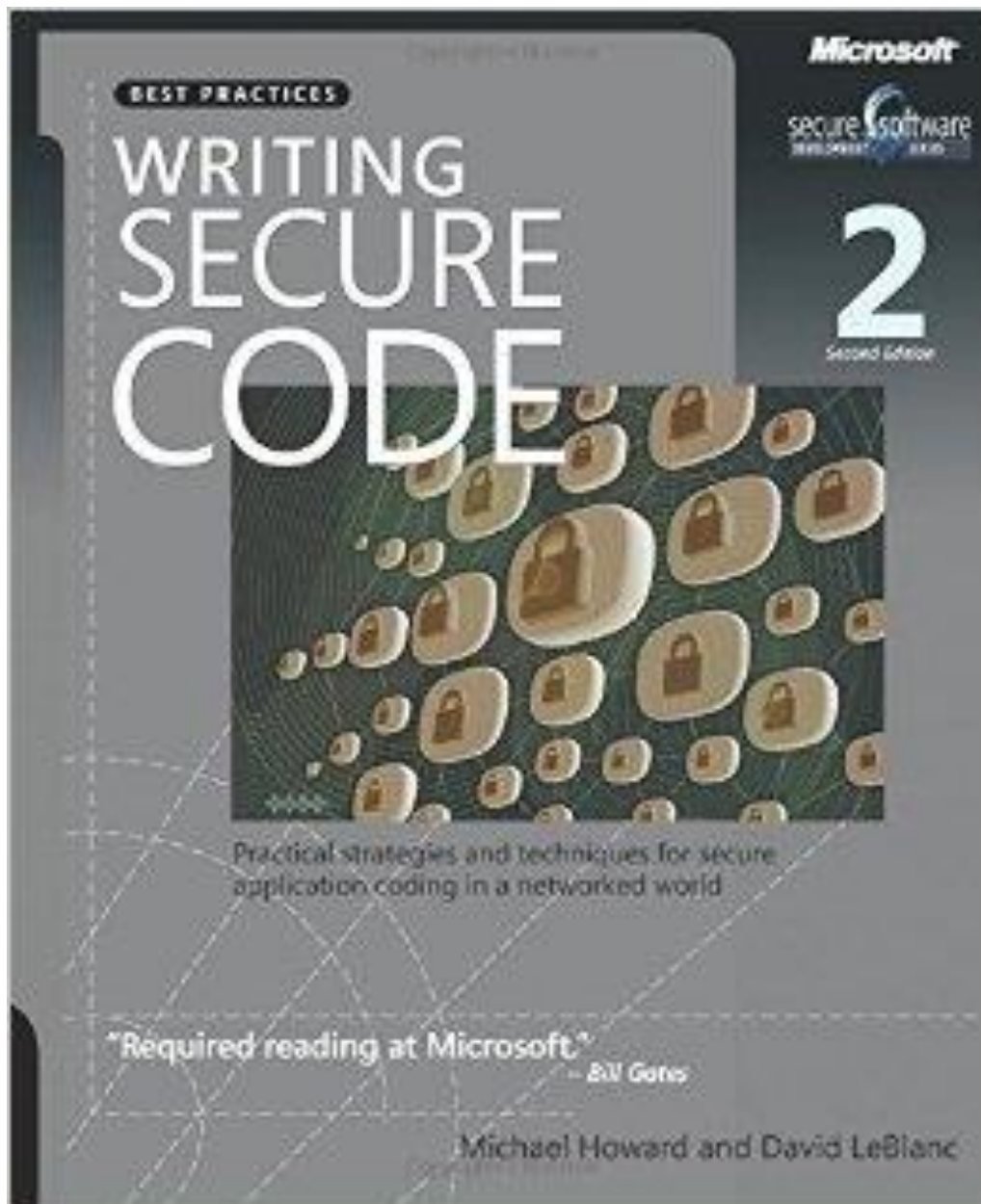
- Complexity, inadequacy, and change
- Incorrect or changing assumptions (capabilities, inputs, outputs)
- Flawed specifications and designs
- Poor implementation of software interfaces (input validation, error and exception handling)
- Inadequate knowledge of secure coding practices



***EQUIFAX***®

# Microsoft SDLs

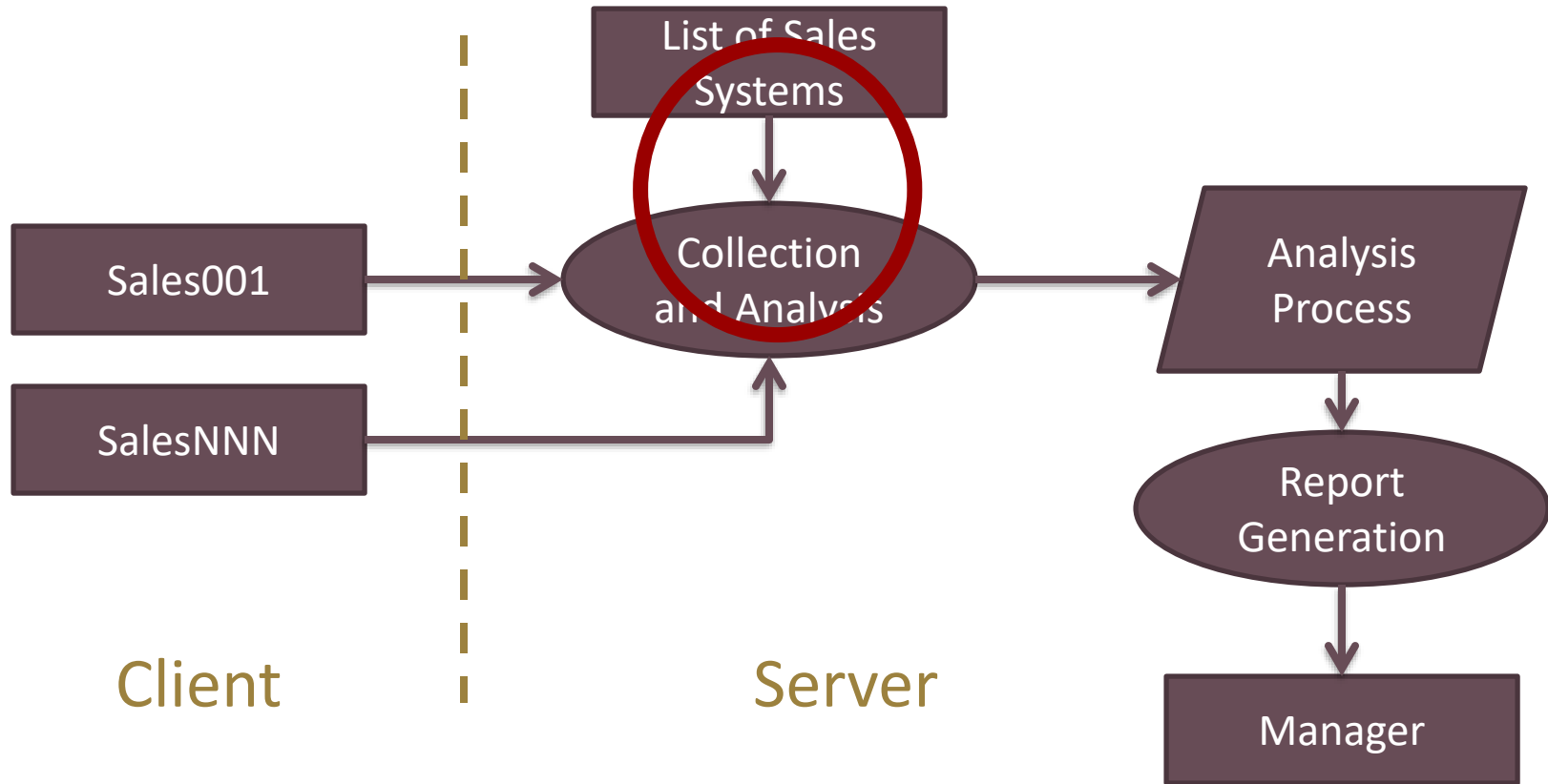




# STRIDE vs Security Properties

Threat	Security Property
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-repudiation
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

# Use case: Sales entry





# Static Analysis, Deprecation

- Microsoft runs static checkers at checking (quality gates)
- Banned over 100 C functions for new code



<https://cwe.mitre.org/>

# Foundations of Software Engineering

Motivation

Michael Hilton

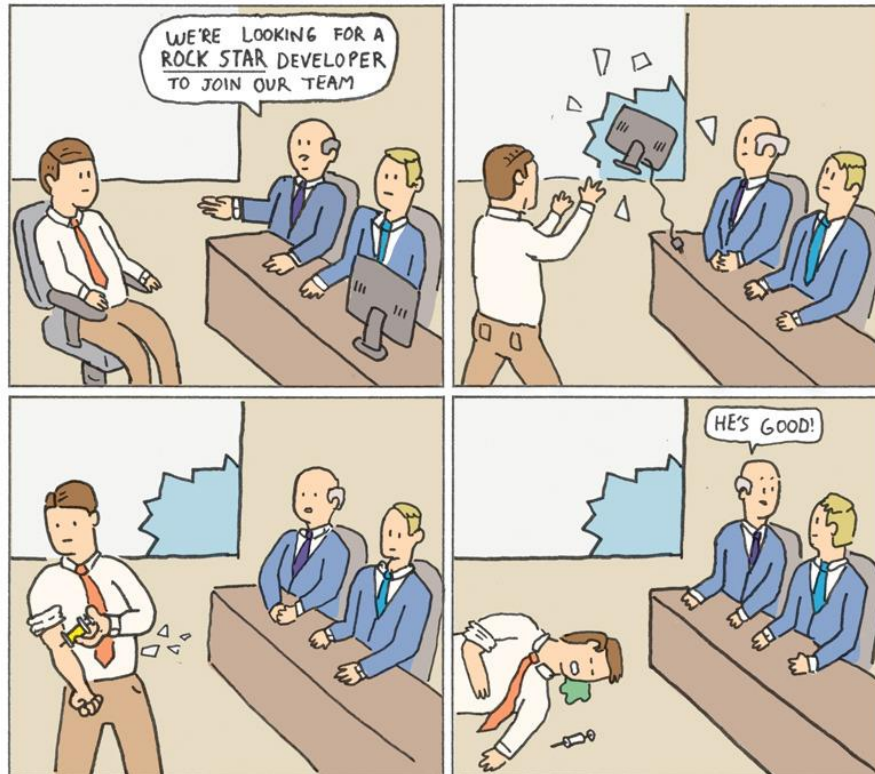
# Learning Goals

- Understand the differences among developers and implications for hiring and teamwork.
- Describe various models of motivation and their relationship to productive work environments.
- Design conditions that motivate developers.
- Understand team development and progression.

# 10X Engineers

- Aka “rock-star”, “ninja”

ROCK STAR DEVELOPER



@SKELETON\_CLAW

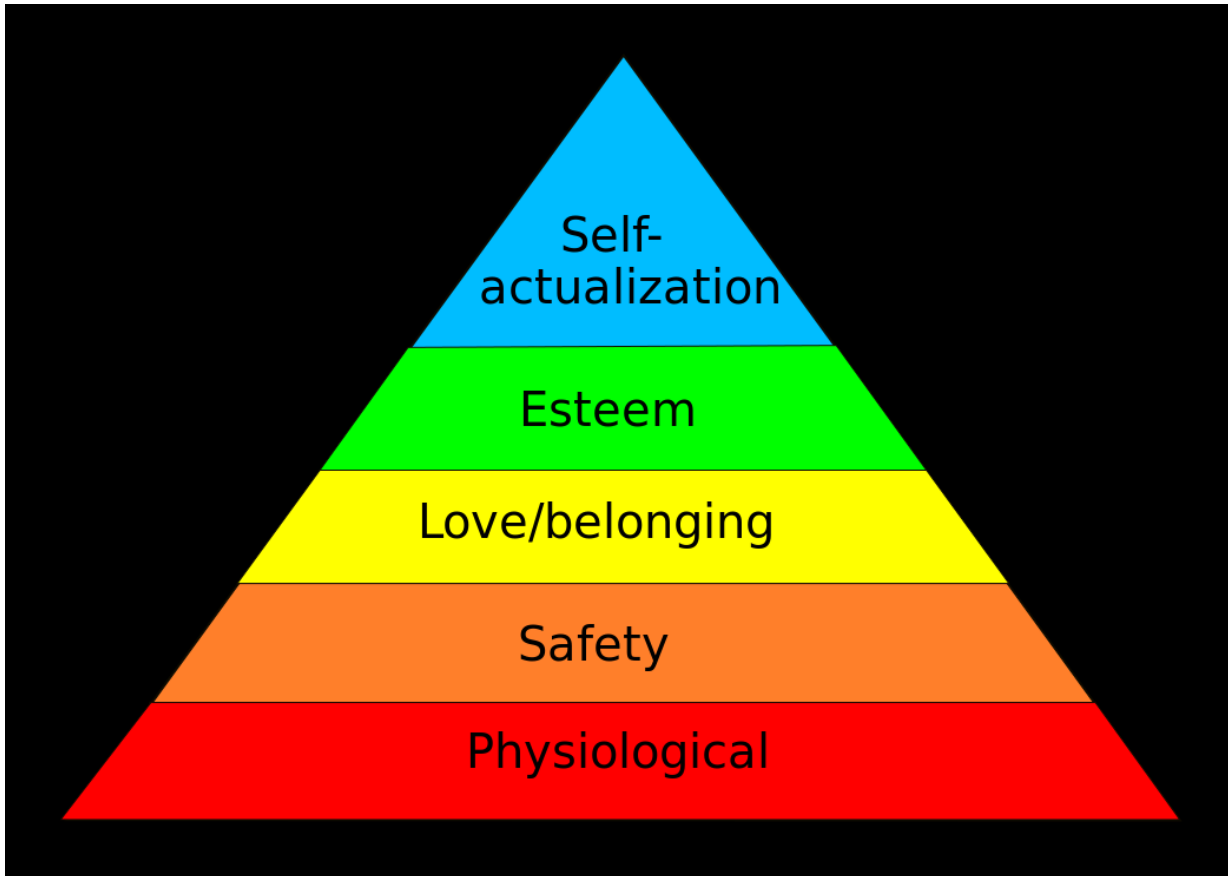
SKELETONCLAW.COM

Rank	Employer Name	Median Age of Employees	Median Employee Tenure	Median Pay
1	Massachusetts Mutual Life Insurance Company	38	0.8	\$60,000
2 - tie	Amazon.com Inc	32	1.0	\$93,200
2 - tie	American Family Life Assurance Company of Columbus (AFLAC)	38	1.0	\$38,000
4 - tie	Google, Inc.	29	1.1	\$107,000
4 - tie	Mosaic	37	1.1	\$69,900
6 - tie	Chesapeake Energy Corporation	31	1.2	\$60,500
6 - tie	Group 1 Automotive, Inc.	32	1.2	\$33,200
6 - tie	Ross Stores, Inc	29	1.2	\$23,800
6 - tie	Wellcare Health Plans, Inc.	38	1.2	\$49,900
*				
11 - tie	Amerigroup Corporation	39	1.3	\$54,800
11 - tie	Brightpoint North America, Inc.	45	1.3	\$42,100
11 - tie	Devon Energy Corporation	31	1.3	\$63,200
11 - tie	Family Dollar Stores Inc	38	1.3	\$23,400
11 - tie	Freeport-McMoRan Copper & Gold Inc	36	1.3	\$62,900
11 - tie	Paccar Corporation	33	1.3	\$62,200

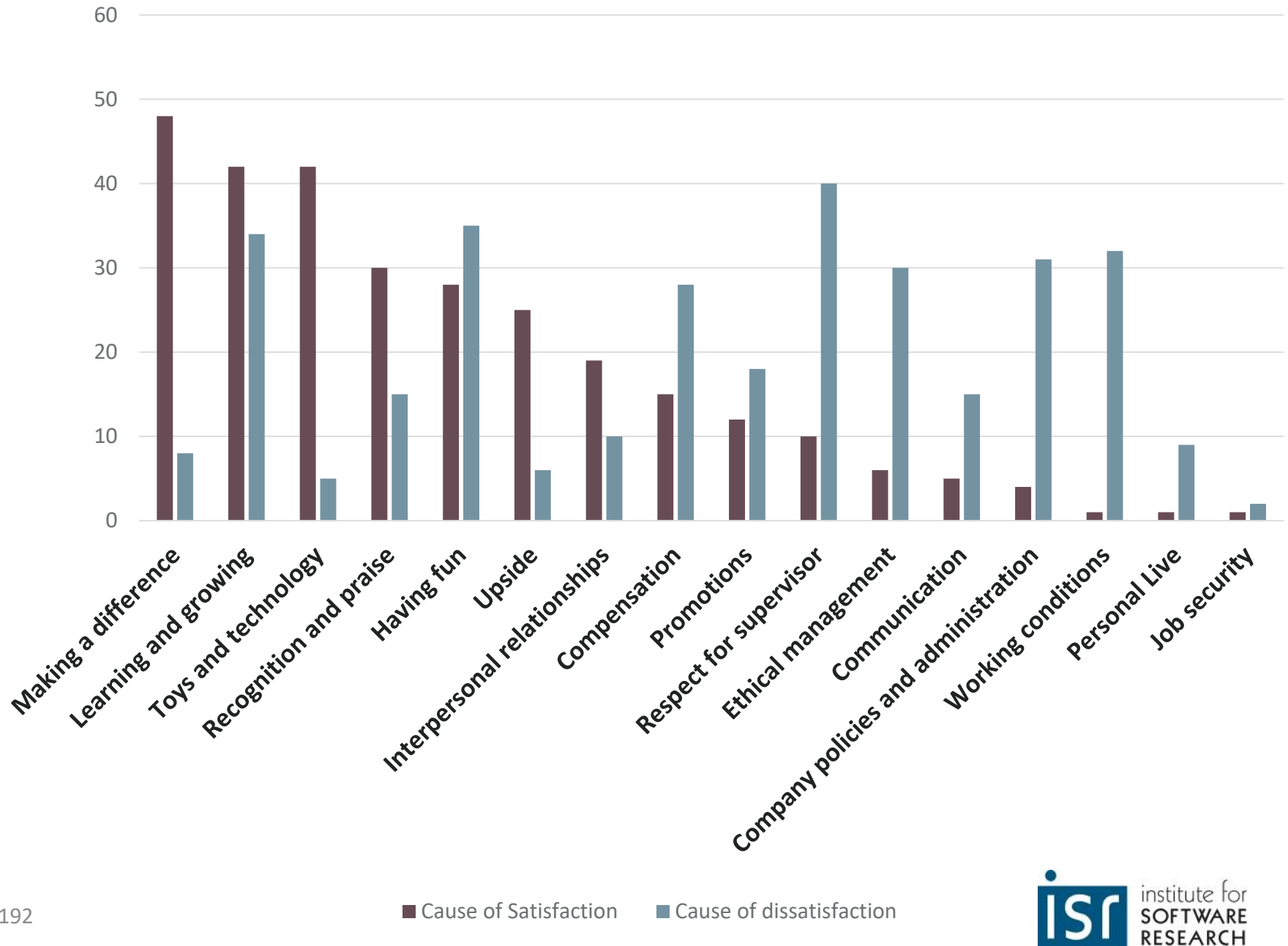
Source: <http://www.techrepublic.com/blog/career-management/tech-companies-have-highest-turnover-rate/>; payscale.com data

18 - tie	Sandisk Corp	34	1.5	\$116,000
18 - tie	Tenneco Inc	40	1.5	\$69,900

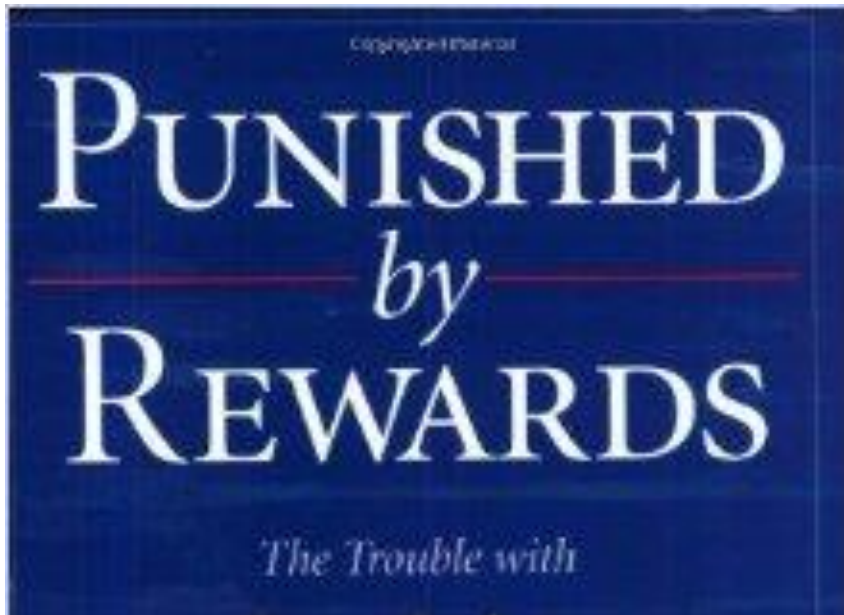
# Maslow's hierarchy of needs (1943)



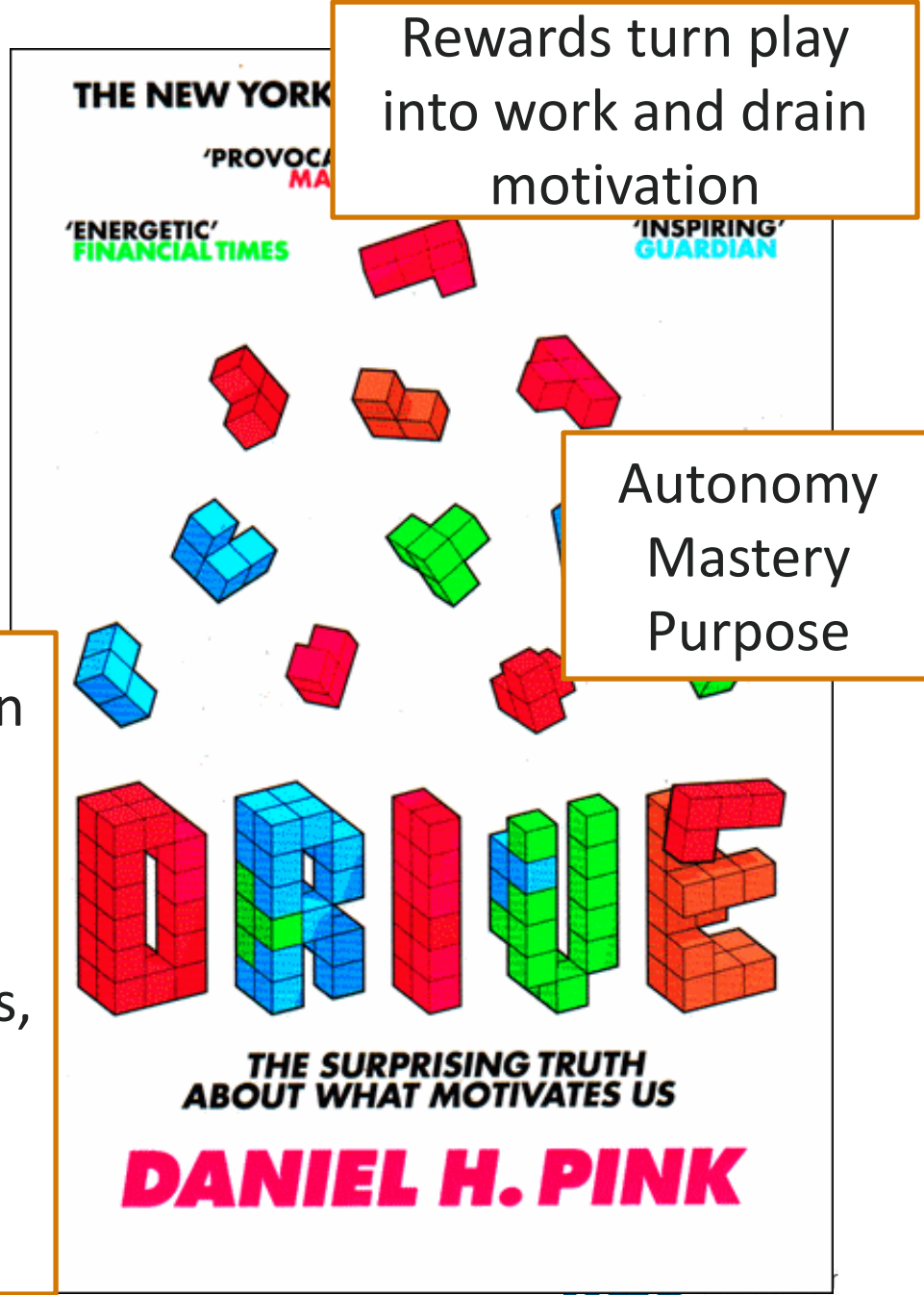
(Observation by Mantle and Lichy, not empirical data)







- Can extinguish intrinsic motivation
- Can diminish performance
- Can crush creativity
- Can crowd out good behavior
- Can encourage cheating, shortcuts, and unethical behavior
- Can become addictive
- Can foster short-term thinking



Rewards turn play into work and drain motivation

Autonomy  
Mastery  
Purpose

# Foundations of Software Engineering

Part 24: Teams

Michael Hilton

# Learning Goals

- ?

# How to structure teams?

- Microblogging platform; 3 friends



A screenshot of the Twitter homepage. At the top is the green 'twitter' logo. To the right of the logo are navigation links: 'Home | Timeline: You, Public | Invite! | Settings | Help'. Below the logo is the heading 'What your friends are doing. (over the last 24 hours)'. The feed contains several tweets, each with a profile picture, a name, and text. The tweets are: 1. RayReadyRay: 'Happy to have my coffee, but reminded of a less present scene from the movie Brain Candy, involving man enjoying his coffee. (2 minutes ago) x'. 2. Jack: 'enjoying the music of my friends. (39 minutes ago) x'. 3. Florian: 'just bought my plane ticket. will get to san francisco on the 16th of september. just in time to still see the drawing restraint exhibition. (about 1 hour ago) x'. 4. Crystal: 'listening to Erlend Øye and making up for lost time. (about 1 hour ago) x'. 5. ev: 'Waiting for slow bagel. Board mtng in 20 (about 1 hour ago) x'. 6. Jack: 'twtr is sweating in anticipation of its imminent launch... (about 1 hour ago) x'. Each tweet has a small icon to its right, likely representing a retweet or reply function.

# How to structure teams?

- Mobile game;
- 50ish developers;
- distributed teams?



# How to structure teams?

- Ride sharing app and self-driving cars; 1200 developers; 4 sites











### Star Wars: Episode I - The Phantom Menace (1999)

🌟 55% 🍿 59%

Critics Consensus: Burdened by exposition and populated with stock characters, *The Phantom Menace* gets the *Star Wars* prequels off to a bumpy – albeit visually dazzling – start.

Starring: Liam Neeson, Ewan McGregor, Natalie Portman

Director: George Lucas



### Star Wars: Episode VI - Return of the Jedi (1983)

🍿 80% 🍿 94%

Critics Consensus: Though failing to reach the cinematic heights of its predecessors, *Return of the Jedi* remains an entertaining sci-fi adventure and a fitting end to the classic trilogy.

Starring: Mark Hamill, Carrie Fisher, Harrison Ford

Director: Richard Marquand



### Star Wars: Episode V - The Empire Strikes Back (1980)

🍿 95% 🍿 97%

Critics Consensus: Dark, sinister, but ultimately even more involving than *A New Hope*, *The Empire Strikes Back* defies viewer expectations and takes the series to heightened emotional levels.

Starring: Mark Hamill, Harrison Ford, Carrie Fisher

Director: Irvin Kershner



### Star Wars: Episode IV - A New Hope (1977)

🍿 93% 🍿 96%

Critics Consensus: A legendarily expansive and ambitious start to the sci-fi saga, George Lucas opened our eyes to the possibilities of blockbuster filmmaking and things have never been the same.

Starring: Mark Hamill, Harrison Ford, Carrie Fisher

Director: George Lucas





**FastLane** is an interactive real-time system used to conduct NSF business over the Internet. FastLane is for official NSF use only. [More About FastLane...](#)

**FastLane  
User  
Support**

(7 AM to 9 PM Eastern Time • M-F)  
**1-800-673-6188**  
FastLane Availability (recording):  
**1-800-437-7408**

[Proposals, Awards and Status](#) | [Proposal Review](#) | [Panelist Functions](#) | [Research Administration](#) | [Financial Functions](#)

[Honorary Awards](#) | [Graduate Research Fellowship Program](#) | [Postdoctoral Fellowships and Other Programs](#)

### Quick Links

- ▶ [Help for Proposal Preparation](#)
- ▶ [Frequently Asked Questions About FastLane Proposal Preparation](#)
- ▶ [Grant Proposal Guide](#)
- ▶ [Deadlines and Target Dates](#)
- ▶ [Change Password](#)
- ▶ [Lookup NSF ID](#)

## Proposals, Awards and Status

Log in for the following permission-based functions:

- ▶ Proposal Functions
  - Letters of Intent
  - Proposal Preparation
  - Proposal Status
  - Display Reference Status
  - Revise Submitted Proposal Budget
  - Proposal File Update
- ▶ Award and Reporting Functions
  - Notifications and Requests - Disabled in FastLane. Log in to [Research.gov](#)
  - Continuation Funding Status
  - View/Print Award Documents
  - Project Reports System - Disabled in FastLane. Log in to [Research.gov](#)
  - Supplemental Funding Request
- ▶ Change PI Information

### PI/Co-PI Log In

Last Name:	<input type="text" value="Kästner"/>
NSF ID: <a href="#">Privacy Act</a>	<input type="text" value="....."/>
Password:	<input type="password" value="....."/>

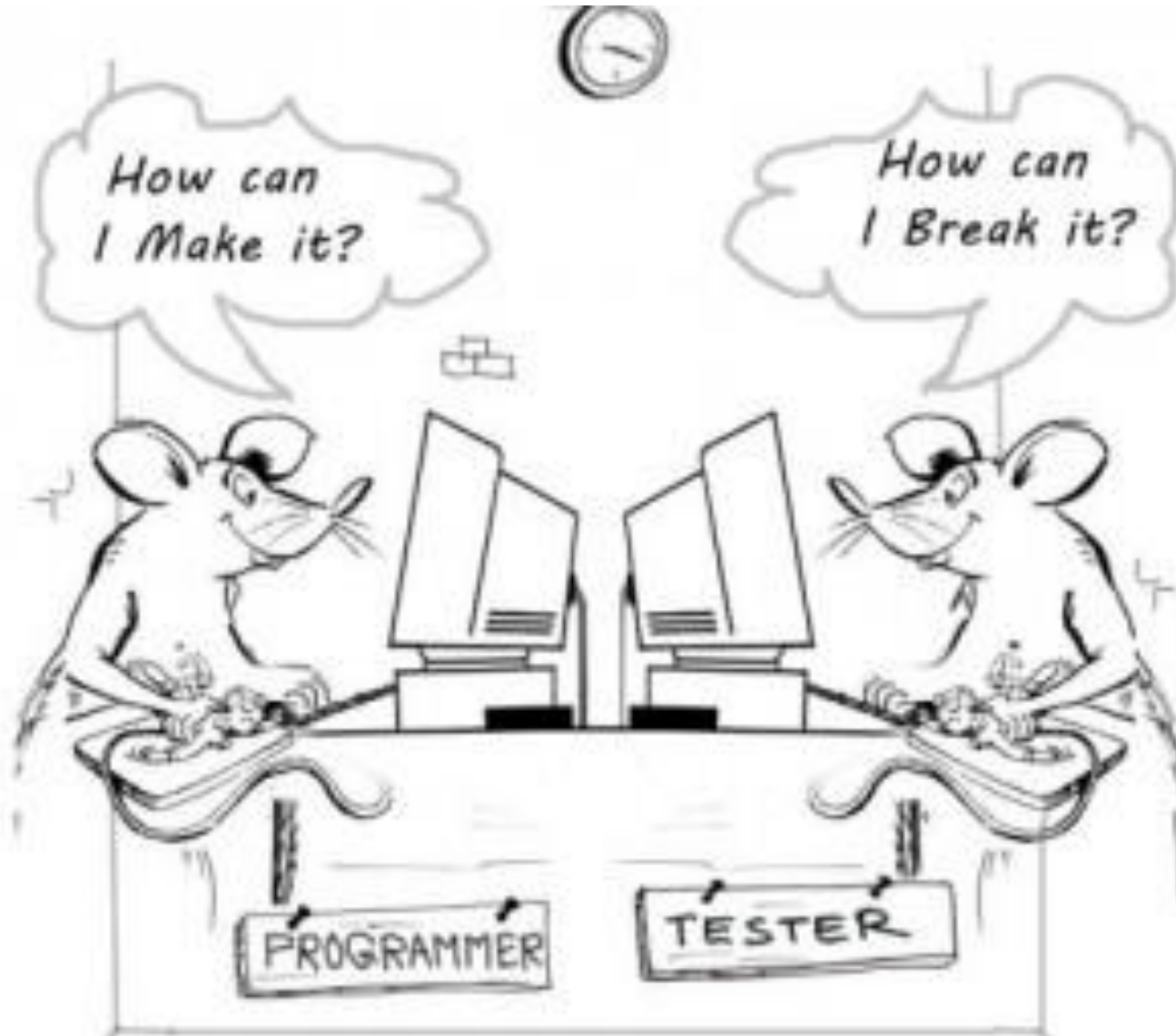
[Log In](#)

[Forgot Password?](#)  
[Lookup NSF ID](#)

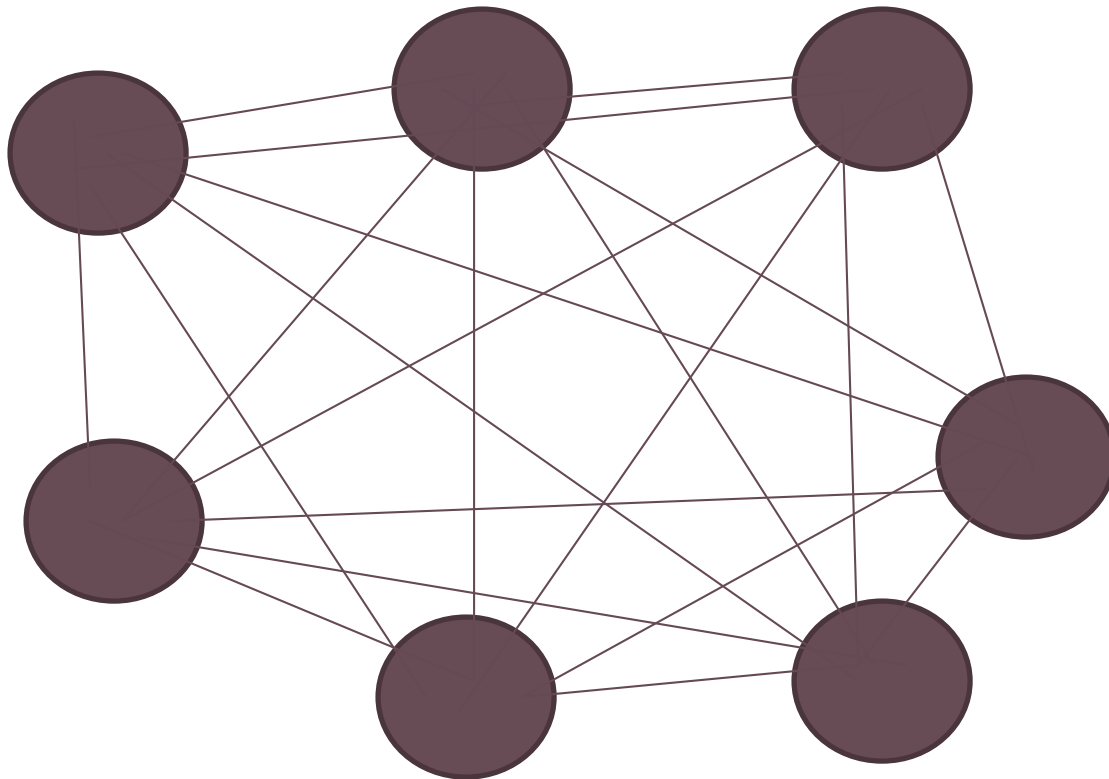
### Other Authorized Users (OAU) Log In

#### Log In by Proposal ID

OAU Last Name:	<input type="text"/>
OAU NSF ID:	<input type="text"/>



# Process Costs



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



# Spotify Squads

# Elitism Case Study: The Black Team

- Legendary team at IBM in the 1960s
- Group of talented ("slightly better") testers
  - Goal: Final testing of critical software before delivery
- Improvement over first year
- Formed team personality and energy
  - "adversary philosophy of testing"
  - Cultivated image of destroyers
  - Started to dress in black, crackled laughs, grew mustaches
- Team survived loss of original members

DeMarco and Lister. Peopleware. Chapter 22

SHIP IT  
SHIP IT GOOD

17-356/17-766  
SOFTWARE ENGINEERING  
FOR STARTUPS



# Foundations of Software Engineering

Lecture 24: Open Source

Michael Hilton

# Learning goals

- Understand the terminology “free software” and explain open source culture and principles.
- Express an educated opinion on the philosophical/political debate between open source and proprietary principles.
- Reason about the tradeoffs of the open source model on issues like quality and risk, both in general and in a proprietary context.

WHEN YOU PROGRAM OPEN SOURCE,  
YOU'RE PROGRAMMING

**COMMUNISM**



A REMINDER  
*from*  
YOUR FRIENDS AT MICROSOFT

Perception:

- Anarchy
- Demagoguery
- Ideology
- Altruism
- Many eyes

# “Free as in free speech.”



February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds \$40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than \$2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

*Bill Gates*  
Bill Gates  
General Partner, Micro-Soft

# Open Source Business Models

- Open source as hobby; resume building
- Selling support/expertise instead of software
  - RedHat
- Selling complementary services
  - Wordpress
- Developers hired as consultants, for extensions

# Coverity Report of Open Source

Only tested programs which use Coverity

Defect density: defects per 1,000 lines

Average defect density of 0.69 for open source and 0.68 for proprietary software, surpassing the industry standard of 1 or less

## Defect Density Based on Size

	Proprietary	Open Source
500,000-1,000,000 (LOC)	0.98	0.44
1,000,000+ (LOC)	0.66	0.75

[Coverity, 2012, <http://www.coverity.com/press-releases/annual-coverity-scan-report-finds-open-source-and-proprietary-software-quality-better-than-industry-average-for-second-consecutive-year/>]

# Microsoft Embraces Open Source

## Redmond top man Satya Nadella: 'Microsoft LOVES Linux'

Open-source 'love' fairly runneth over at cloud event



20 Oct 2014 at 23:45, Neil McAllister





▲ FindBugs project in its current form is dead (umd.edu)

264 points by fanalin 24 days ago | hide | past | web | 116 comments | favorite

▲ billpugh 23 days ago [-]

FindBugs isn't dead (although my participation had been in hibernation for a while).

I've been juggling far too many projects, but I'm now working to move FindBugs back into the active rotation.

I also want announce I'll be working with GrammaTech as part of the Swamp Project, and they will be helping with rebooting the FindBugs project. year), and although I've known that GrammaTech was likely to win an award, this hasn't been official and something I could talk about until recent concrete to talk about as far as that goes; but I don't yet have the information I wanted to share.

Thanks to all the FindBugs fans and supporters who lobbied for me to return to active maintenance of FindBugs. Give me a week to get up to speed

Bill Pugh

▲ unreal37 23 days ago [-]

It's still not a good sign that it took this level of public attention to get you to reply to the active community on their urgent needs.

▲ dmuth 23 days ago [-]

^ this. There is absolutely no reason to not be answering emails, even if to say, "I'm really swamped, and need help."

I don't mean to denigrate you, but I must be candid here: hoarding admin rights so that only you have them and no one else can get

Going forward, I would recommend taking a look through other projects you may be involved with, and make sure that you are not to increase the "bus factor" to greater than 1: [https://en.wikipedia.org/wiki/Bus\\_factor](https://en.wikipedia.org/wiki/Bus_factor)

Good luck.

▲ junkster78 23 days ago [-]

Totally agree. Even worst, the community has already forked and is getting aligned behind <https://github.com/spotbugs/spotbugs> stays forked, with split efforts and APIs diverging forward, or it's shutdown to trust Bill again and hope for the best.

I for one have lost all trust on his ability to lead the project after such long absence, and the fact that he only reappeared after more about what people may be thinking / saying of him, than the well being of FindBugs.

I fear, the only way for FindBugs to stay alive, is for Bill to do what he probably should have done a long time ago: step down.

▲ issaria 22 days ago [-]

This is also the problem for other projects, the leaders' unwilling to giveup their position even they are inactive for a ver

# Open SSL/Heartbleed.

- In 2013, OpenSSL made \$2,000 in donations (and some from other sources)
- One full time programmer
- Heartbleed (2014): Vulnerability was found that effected about 17.5% of web servers (half a million)
- Used by Yahoo, Twitter, Google
- Who is responsible?



# Open Source Licenses

Software	Percentage
MIT License	24%
GNU General Public License (GPL) 2.0	23%
Apache License 2.0	16%
GNU General Public License (GPL) 3.0	9%
BSD License 2.0 (3-clause, New or Revised) License	6%
GNU Lesser General Public License (LGPL) 2.1	5%
Artistic License (Perl)	4%
GNU Lesser General Public License (LGPL) 3.0	2%
Microsoft Public License	2%
Eclipse Public License	2%

List from: <https://www.blackducksoftware.com/resources/data/top-20-open-source-licenses>

# HW6: Open Source Contribution

# Data Analytics in Software Engineering

Christian Kaestner

# Learning Goals

- Understand importance of data-driven decision making also during software engineering
- Collect and analyze measurements
- Design evaluation strategies to evaluate the effectiveness of interventions
- Understand the potential of data analytics at scale for QA data

# Once Upon a Time...



Seven Years' War (1754-63)

Britain loses 1,512 sailors to enemy action...

...and almost 100,000 to scurvy

What metrics are the **best predictors of failures**?

If I increase **test coverage**, will that actually increase software quality?

What is the **data quality** level used in empirical studies and how much does it actually matter?

Are there any **metrics that are indicators of failures** in both Open Source and Commercial domains?

I just submitted a **bug report**.  
Will it be fixed?

Should I be writing **unit tests** in my software project?

How can I tell if a piece of software will have **vulnerabilities**?

Do **cross-cutting concerns** cause defects?

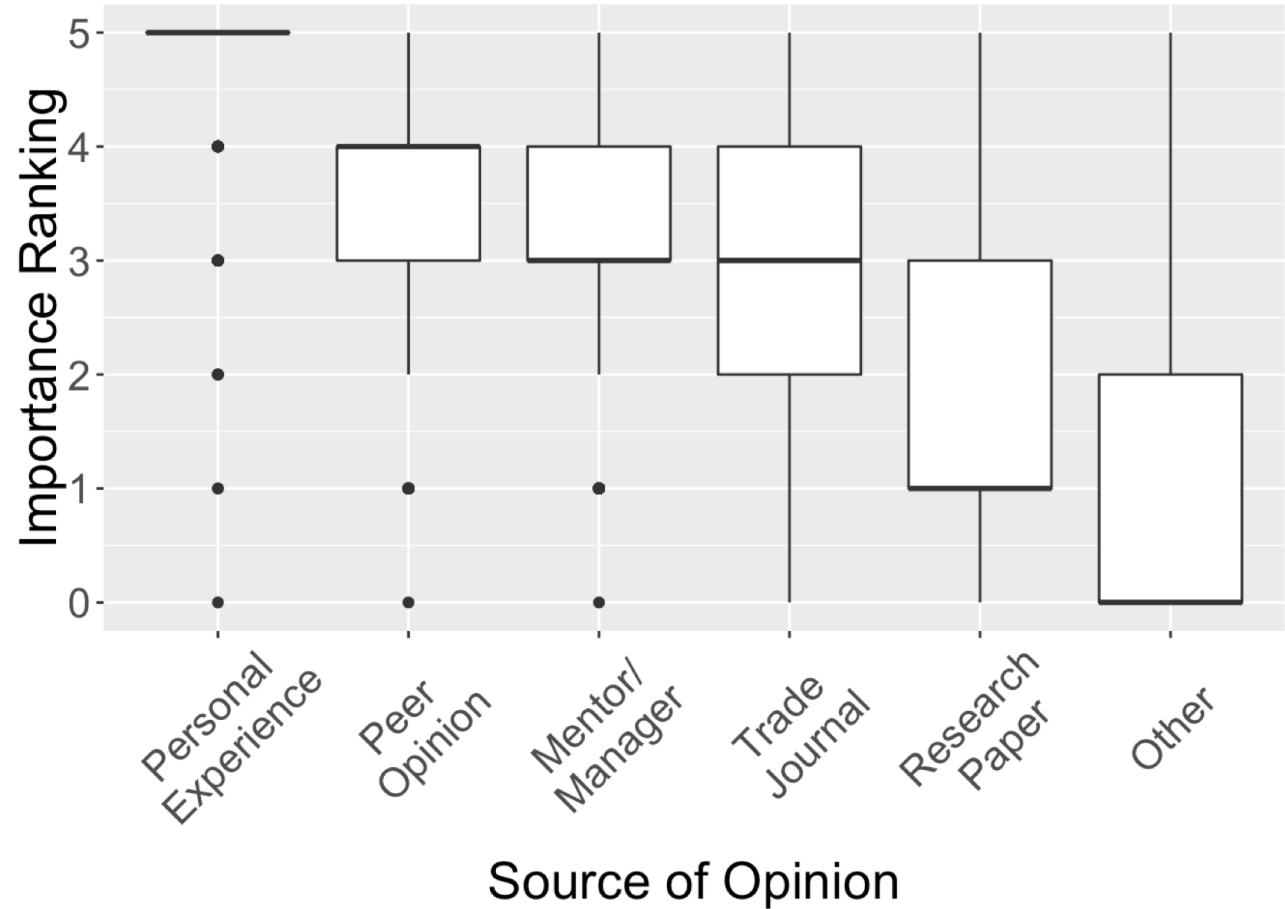
Is strong **code ownership** good or bad for software quality?

Does **Test Driven Development** (TDD) produce better code in shorter time?

Does **Distributed/Global software development** affect quality?



# Source of Believes



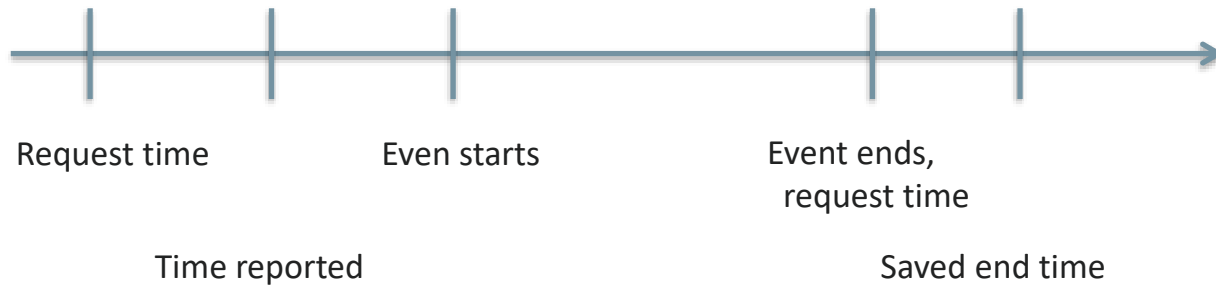
225



# Timer Overhead



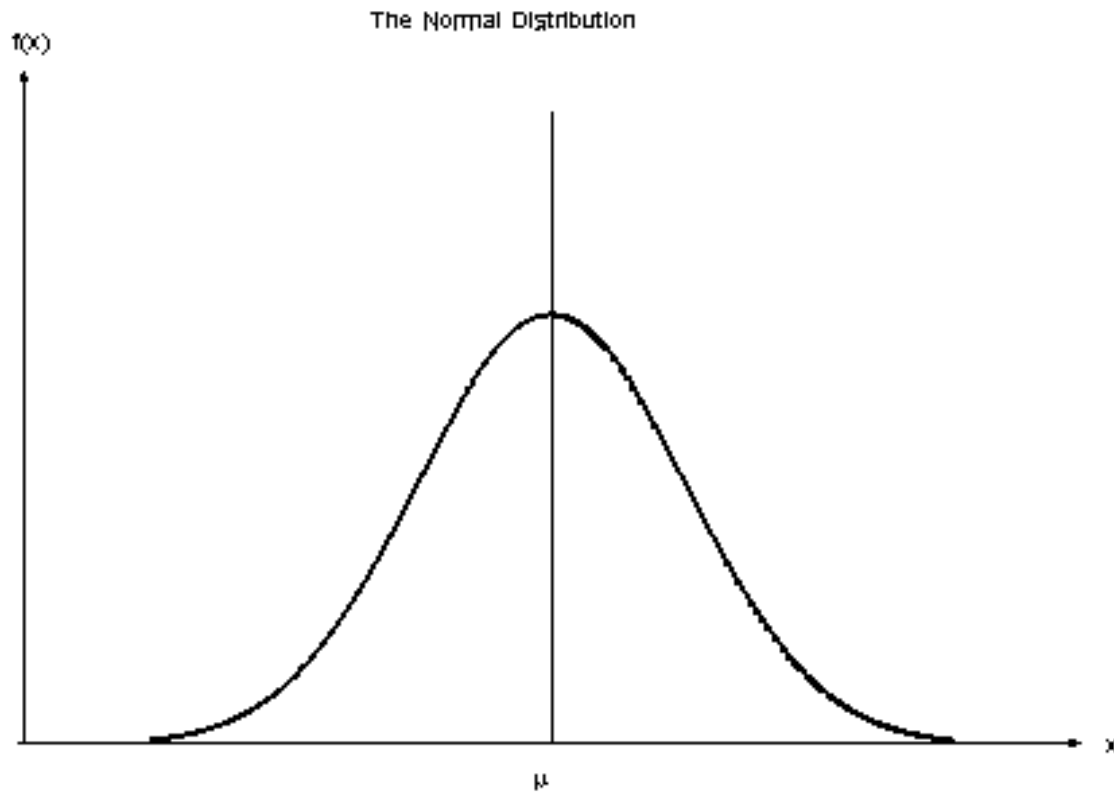
- Measurement itself consumes time



Memory access and interaction  
with operating system

Measured event should be 100-1000x  
longer than measurement overhead

# Normal distributions



# Abundance of Data

- Code history
- Developer activities
- Bug trackers
- Sprint backlog, milestones
- Continuous integration logs
- Static analysis and technical debt dashboards
- Test traces; dynamic analyses
- Runtime traces
- Crash reports from customers
- Server load, stats
- Customer data, interactions
- Support requests, customer reviews
- Working hours
- Team interactions in Slack/issue tracker/email/...
- ...

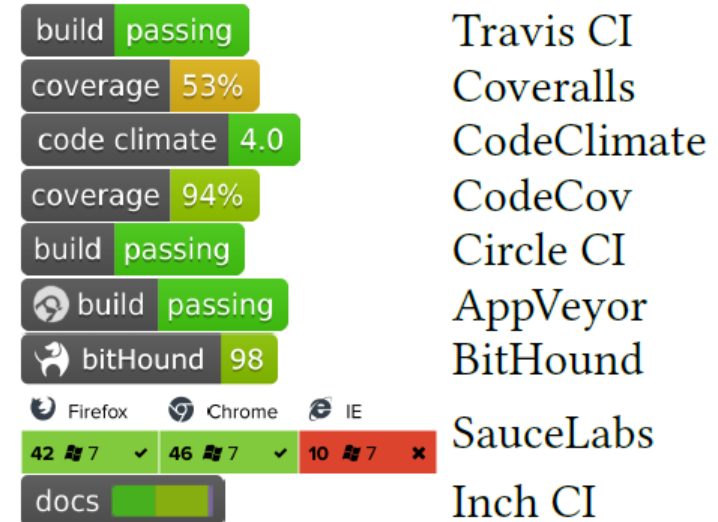
# Example: Badges

	Basic Model response: <i>freshness</i> = 0 17.3% deviance explained		Full Model response: <i>freshness</i> = 0 17.4% deviance explained		RDD response: $\log(\textit{freshness})$ $R_m^2 = 0.04, R_c^2 = 0.35$	
	Coeffs (Err.)	LR Chisq	Coeffs (Err.)	LR Chisq	Coeffs (Err.)	Sum sq.
(Interc.)	3.54 (0.03)***		3.50 (0.03)***		1.45 (0.09)***	
Dep.	-1.78 (0.01)***	32077.8***	-1.79 (0.01)***	32292.8***	-0.04 (0.02)	3.01
RDep.	0.22 (0.01)***	610.3***	0.21 (0.01)***	560.6***	-0.01 (0.02)	0.11
Stars	-0.08 (0.00)***	301.4***	-0.09 (0.00)***	311.2***	0.00 (0.01)	0.00
Contr.	-0.24 (0.01)***	500.5***	-0.25 (0.01)***	548.7***	-0.04 (0.02)*	4.39*
lastU	-0.65 (0.01)***	12080.9***	-0.64 (0.01)***	11537.9***	0.01 (0.02)	0.37
hasDM			0.24 (0.03)***	116.1***	0.45 (0.08)***	2.43
hasInf			0.11 (0.02)***	48.3***	0.04 (0.05)	0.45
hasDM:hasInf			-0.05 (0.04)	1.9	-0.32 (0.10)**	
hasOther			0.01 (0.01)			
time					0.03 (0.00)***	82.99***
intervention					-0.93 (0.03)***	1373.22***
time_after_intervention					0.11 (0.00)***	455.56***
time_after_intervention:hasDM					-0.10 (0.01)***	230.36***
time_after_intervention:hasInf					-0.00 (0.01)	1.14
time_after_intervention:hasDM:hasInf					0.03 (0.01)**	10.62**

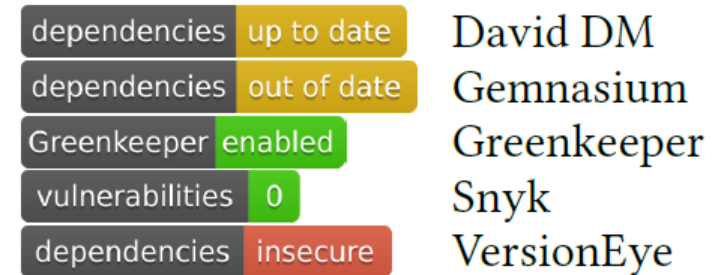
\*\*\*  $p < 0.001$ , \*\*  $p < 0.01$ , \*  $p < 0.05$ ;

Dep: dependencies; RDep: dependents; Contr.: contributors; lastU: time since last update; hasDM: has dependency-manager badge; hasInf: has information badge; hasOther: adopts

## QUALITY ASSURANCE

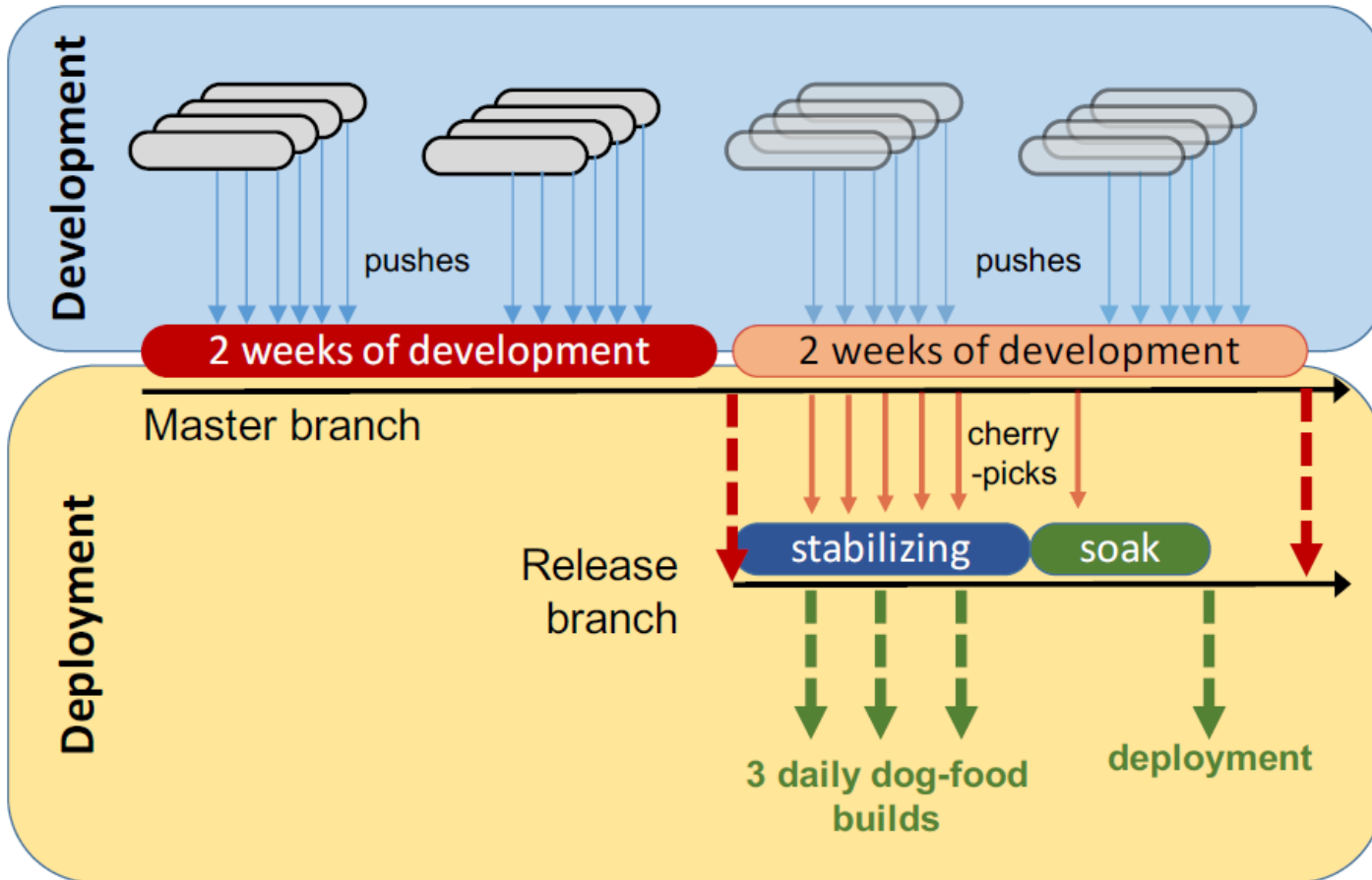


## DEPENDENCY MANAGEMENT



# Testing in Production

- Beta tests
- AB tests
- Tests across hardware/software diversity (e.g., Android)
- “Most updates are unproblematic”
- “Testing under real conditions, with real workloads”
- Avoid expensive redundant test infrastructure



Release cycle of Facebook's apps



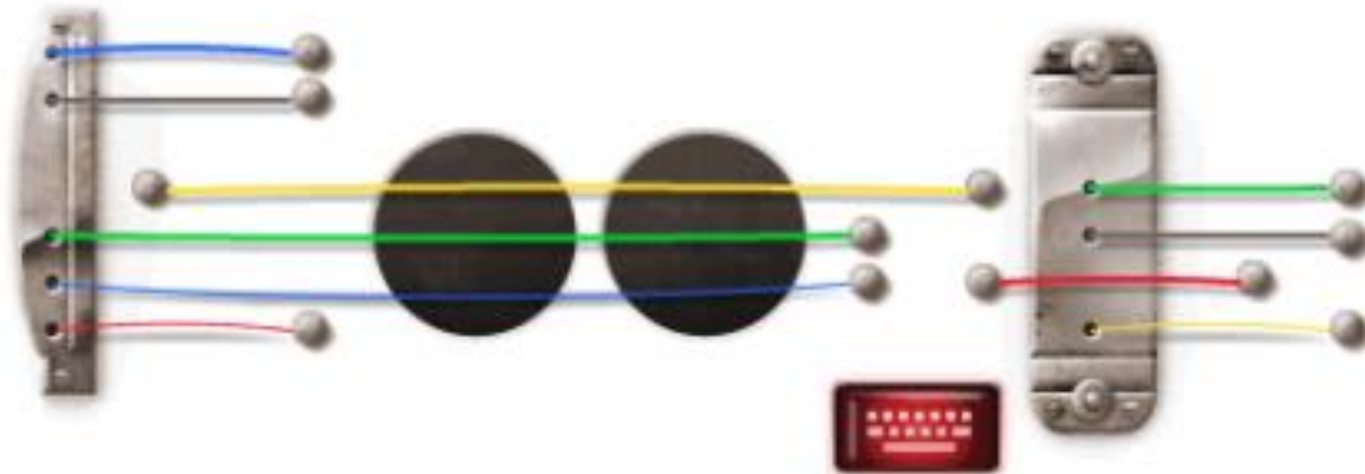
# Foundations of Software Engineering

Part 26: Software Engineering Ethics

Michael Hilton

# Learning goals

- Awareness of ethical issues in software engineering
- Reflection on decision making
- Knowledge of professional codes
- Starting points to dig deeper



Google Search

I'm Feeling Lucky

“Update Jun 17: Wow—in just 48 hours in the U.S., you recorded 5.1 years worth of music—40 million songs—using our doodle guitar. And those songs were played back 870,000 times!”

# Open Source Maintainers



dominictarr commented 7 days ago

Owner ...



dominictarr commented 7 days ago

Owner ...



limonte commented 7 days ago • edited ▾

...



dominictarr commented 6 days ago

Owner ...



XhmikosR commented 6 days ago

...



jaydenseric commented 6 days ago

...

There is a huge difference between not maintaining a repo/package, vs giving it away to a hacker (which actually takes more effort than doing nothing), then denying all responsibility to fix it when it affects millions of innocent people.



884



162



7



16



18



# A/B Testing

## The Morality Of A/B Testing



Josh Constine @joshconstine / 4 years ago

 Comment



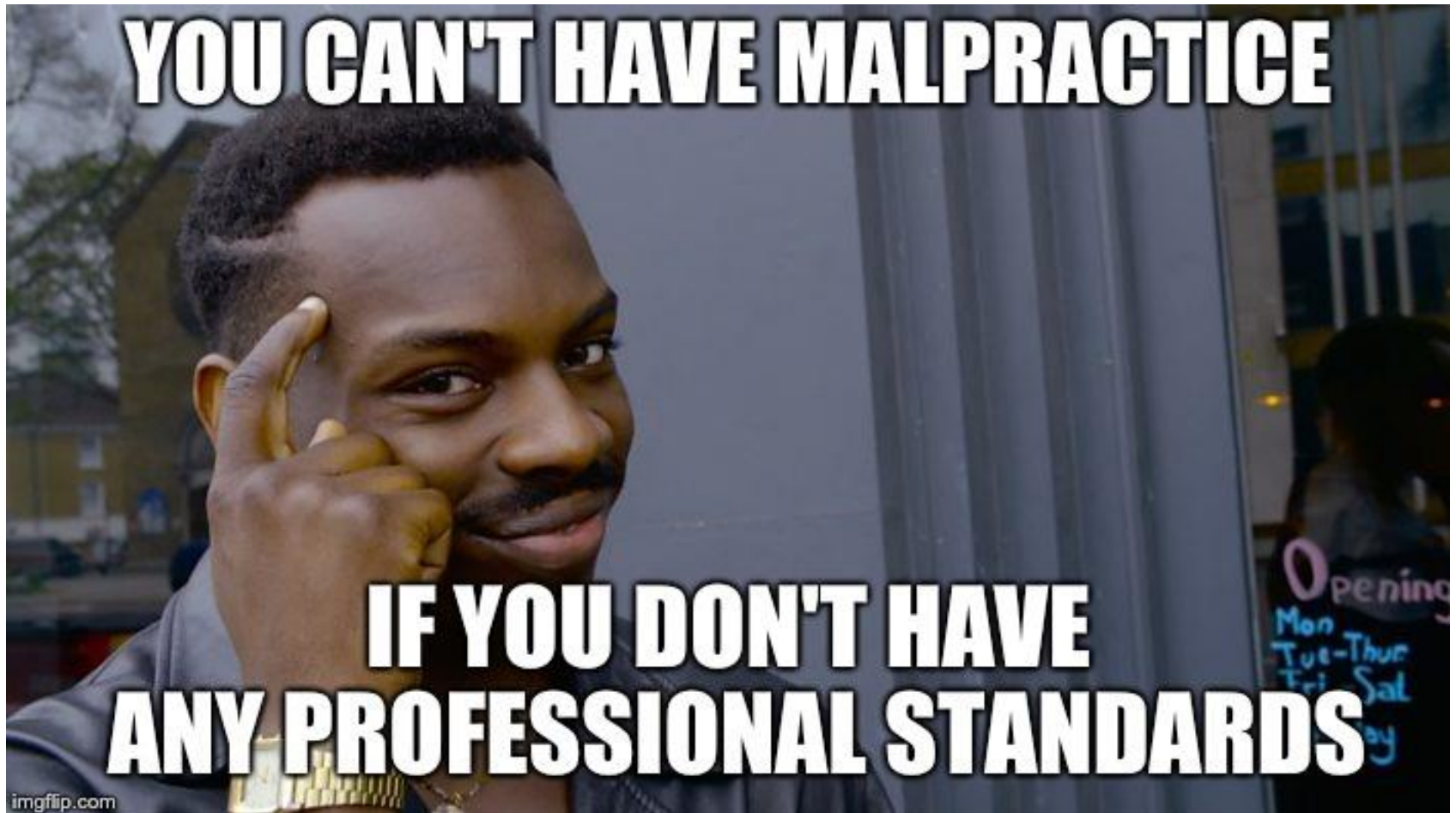
# Dual use



# Professional Engineer



# Malpractice vs. Negligence





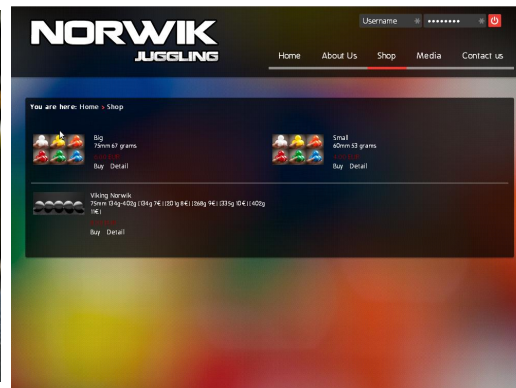
# IEEE CS/ACM Software Engineering Code of Ethics (short version)

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

- **Public:** Software engineers shall act consistently with the public interest.
- **Client and Employer:** Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.
- **Product:** Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
- **Judgement:** Software engineers shall maintain integrity and independence in their professional judgment.
- **Management:** Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
- **Profession:** Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
- **Colleagues:** Software engineers shall be fair to and supportive of their colleagues.
- **Self:** Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

# Obligations to whom?

- Public welfare
- Some cases more obvious: QA for pacemaker
- Analyze stakeholders, including fringes



# SE 17200 - Ethics and Policy Issues in Computing

## Description

Note: Previously offered as 08-200. In this course, students will study the social impacts of computing technology and systems. The course will provide a brief introduction to ethics and to the new and difficult ethical questions modern computing technology presents us with. It will focus on a number of areas in which computers and information technology are having an impact on society including data privacy, social media, and autonomous technologies.

## Credits

9

## Recent Professors

[James Herbsleb](#)

## Schedule Planner

[Add SE 17200 to your schedule](#)

## Recent Semesters

Spring 2019

## Offered

TuTh

## Avg. Sections

1

# HW6 Presentations

# Thanks!