Course Overview & Software Process

Advanced Software Engineering
Spring 2023

Introduction

- Jon, Prof Bell, Prof Jon, Dr Bell, Dr Jon, etc…
  - Research: Software Engineering, Program Analysis
  - Open source contributor & project founder

Introductions: Class

- Your name
- Your degree program
- Your past experiences with software engineering
- Your motivation for taking this class/what you want to learn

Course Mechanics

- Course website: https://neu-se.github.io/CS4910-7580-Spring-2023/
- Notes:
  - Calendar & readings
  - Assessments
  - Attendance policy
  - Discord

Software Engineering as a Discipline c. 1969

[Software Engineering as a Class]

- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

SOFTWARE ENGINEERING

Report on a conference sponsored by the
NATO SCIENCE COMMITTEE
Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer
Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

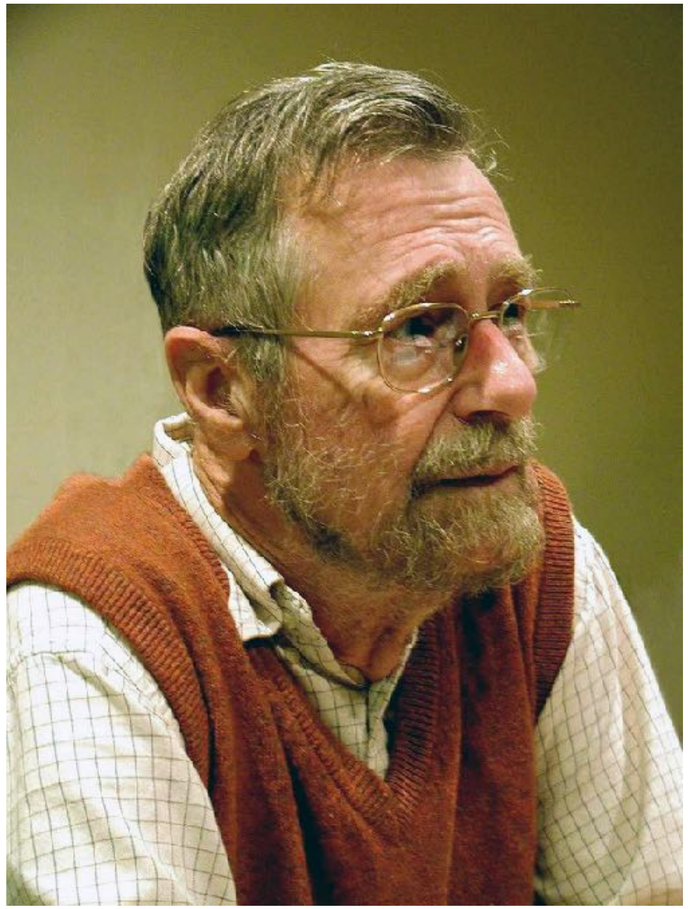
Editors: Peter Naur and Brian Randell

January 1969

A call to action:
We must study
how to build
software

Software Engineering as a Discipline

[Software Engineering as a Class]



The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

- Edsger W. Dijkstra, in his 1972 Turing Award acceptance speech

Increase in computational capacity over time

Increase over software complexity?

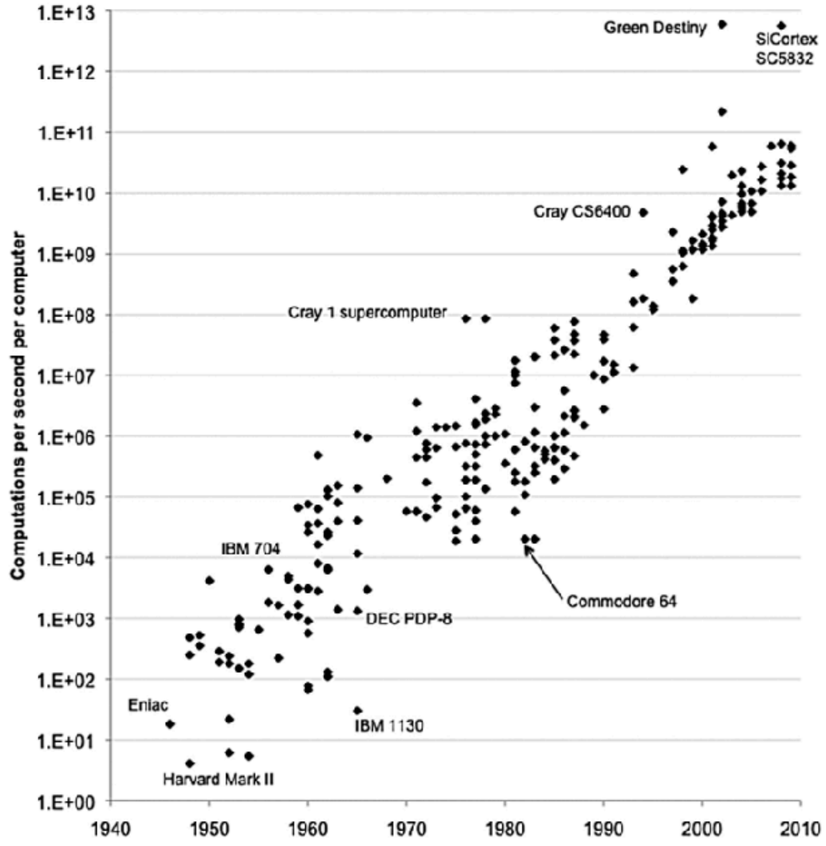
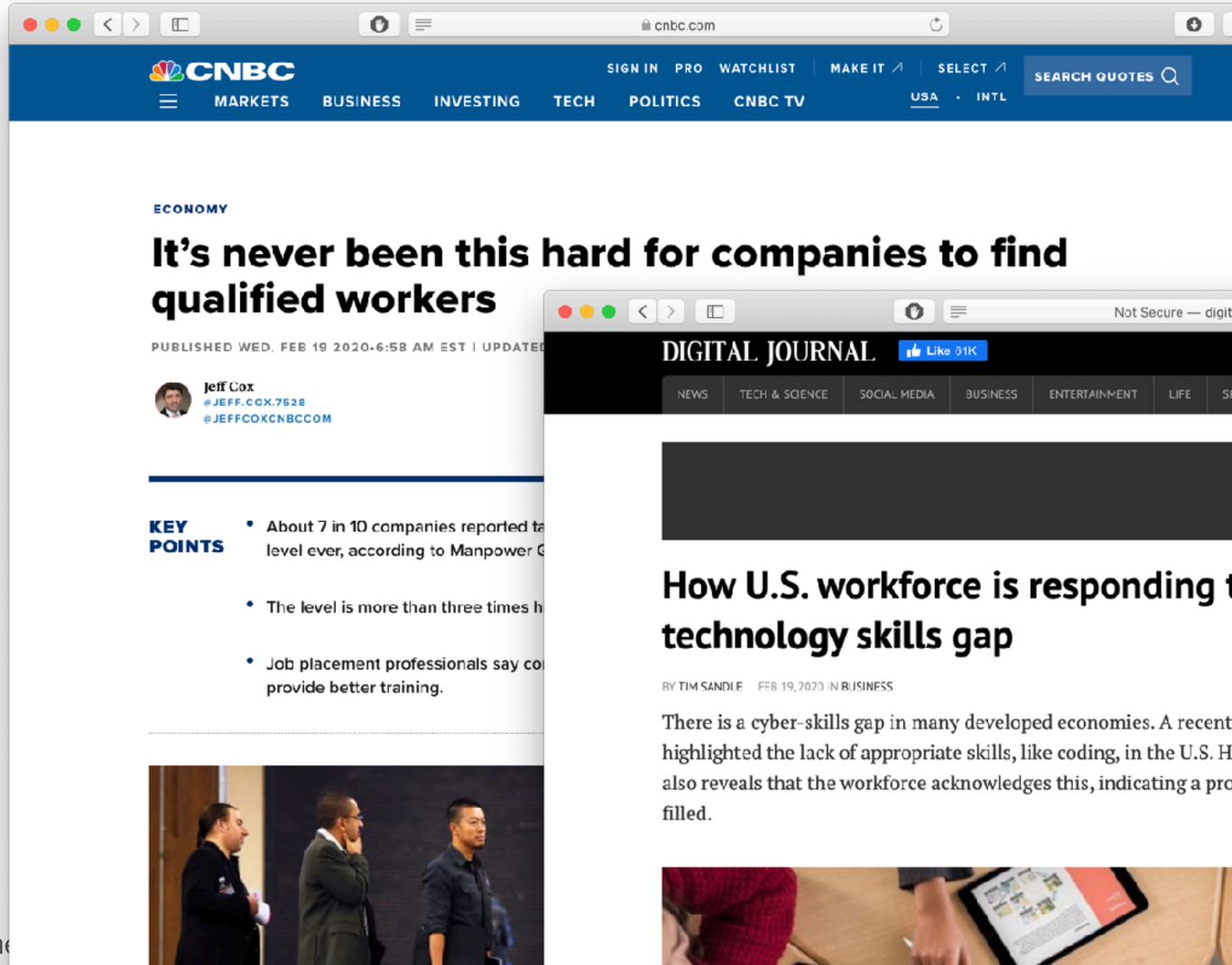


Figure 2. Computational capacity over time (computations/second per computer). These data are based on William D. Nordhaus' 2007 work,[9] with additional data added post-1985 for computers not considered in his study. Doubling time for personal computers only (1975 to 2009) is 1.5 years.

"Implications of Historical Trends in the Electrical Efficiency of Computing" Koome



Java Puzzler

```
HashSet<String> mySet = new HashSet<String>();
mySet.add("a");
mySet.add("b");
Iterator<String> iter = mySet.iterator();
System.out.println(iter.next()); //What is printed?
```

This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element.

-JavaDoc for HashSet

1,000,000 trials: “a” is printed every time

BUT NOT GUARANTEED

Why is this puzzler a problem?

Consider the entire lifespan of this problematic code

```
class BookTest {
    @Test
    public void testGetStringRepresentation() {
        Book b = new Book("book", "name");
        assertEquals("{\"title\":\"book\",\"author\":\"name\"}",
                b.getStringRepresentation());
    }
}
```

What could go wrong here?

What if Book is just a HashMap?

```
{
   "title": "book",
   “author": "name"
}
```

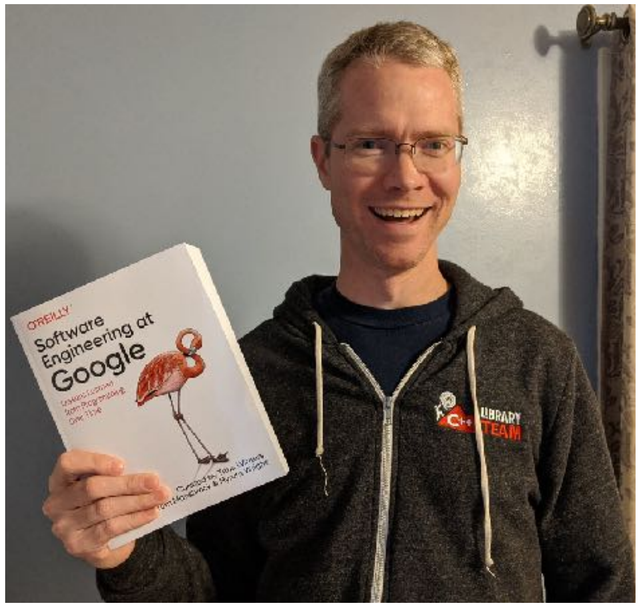
```
{
   “author": “name",
   "title": "book"
}
```

Both are possible :(

Whose fault is this?

Software Engineering is Programming at Scale

Hyrum's Law



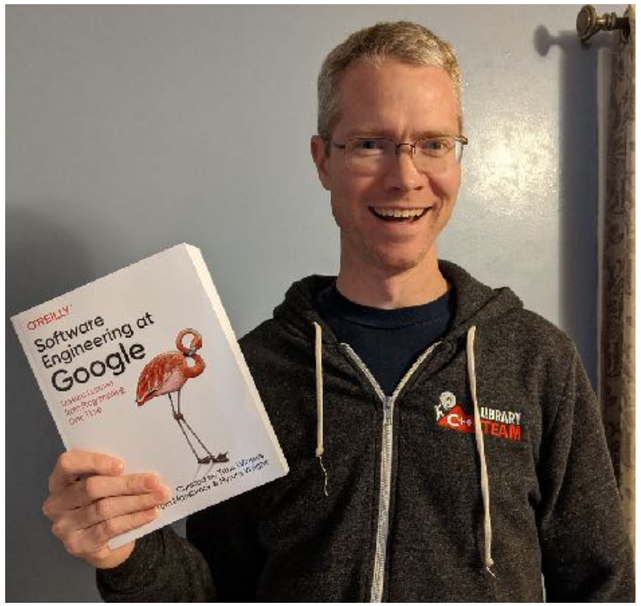
"With a sufficient number of users of an API,
it does not matter what you promise in the contract:
all observable behaviors of your system
will be depended on by somebody."

-Hyrum Wright

For more: see "Software Engineering at Google" Ch 1

Software Engineering is Programming at Scale

Hyrum's Law



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

-Hyrum Wright

XKCD #1172

Can we do better than making a law?

Software engineering research aims to automatically improve development processes

2016 IEEE International Conference on Software Testing, Verification and Validation

Detecting Assumptions on Deterministic Implementations of Non-deterministic Specifications

August Shi, Alex Gyori, Owolabi Legunsen, and Darko Marinov
Department of Computer Science
University of Illinois at Urbana-Champaign, USA
Email: {awshi2,gyori,legunse2,marinov}@illinois.edu

***Abstract*—Some commonly used methods have non-deterministic specifications, e.g., iterating through a set can return the elements in any order. However, non-deterministic specifications typically have deterministic implementations, e.g., iterating through two sets constructed in the same way may return their elements in the same order. We use the term *ADINS code* to refer to code that *A*ssumes a *D*eterministic *I*mplementation of a method with a *N*on-deterministic *S*pecification. Such ADINS code can behave unexpectedly when the implementation changes, even if the specification remains the same. Further, ADINS code can lead to *flaky tests*—tests that pass or fail seemingly non-deterministically.**

We present a simple technique, called NONDEX, for detecting flaky tests due to ADINS code. We implemented NONDEX for Java: we found 31 methods with non-deterministic specifications in the Java Standard Library, manually built non-deterministic models for these methods, and used a modified Java Virtual Machine to explore various non-deterministic choices. We evaluated NONDEX on 195 open-source projects from GitHub and 72 student submissions from a programming homework assignment. NONDEX detected 60 flaky tests in 21 open-source projects and 110 flaky tests in 34 student submissions.

I. INTRODUCTION

Non-deterministic specifications are not uncommon for many methods, including in the standard libraries of many programming languages. For example, the specification for the `Object#hashCode()` method in Java can return any integer. Non-deterministic specifications are not restricted to simple APIs. The order in which elements of a set are returned by an iterator is not-specified—it can be any order. The order in which entries in a SQL table are returned is also sometimes not specified—it depends on the query. Such specifications give implementers more freedom to develop various implementations for different goals, e.g., to optimize performance, while still satisfying the specification.

Even when specifications allow for non-determinism, typical implementations of such specifications are often deterministic, with respect to certain controlled sources. For example, `Object#hashCode()` could return the same integer (if one controls for all other sources, e.g., OpenJDK Java 8 could return a deterministic value on the first call if the under-code—is often bad. Such ADINS code can behave unexpectedly when the implementation changes, even if the specification remains the same. For example, Java code that assumes a specific iteration order of a `HashSet`, e.g., that a `HashSet` with elements `1` and `2` will be always represented as a string `{1, 2}` rather than `{2, 1}`, is ADINS and not robust: the Java implementation of `HashSet` can change such that the iteration order of the elements changes and the string differs.

Unexpected behavior of ADINS code can lead to *flaky tests*, which are tests that seem to non-deterministically pass or fail. Flaky tests are bad as they can mask bugs (pass when there are bugs) or raise false alarms (fail when there are no bugs). A test that executes ADINS code can be flaky if it assumes that some values are deterministic even if they can change: when the assumptions hold, the test passes, but when the assumptions do not hold, the test may fail. Not all flaky tests are due to ADINS code, e.g., a test asserting that a file system contains `/tmp` could pass on one machine but fail on another. Flaky tests are emerging as an active research topic, with recent work on characterizing [25], detecting [2], [4], [10], [12], [14], [38], and avoiding [1], [22] flaky tests. However, no previous research investigated ADINS code as a cause for flaky tests.

While flaky tests are an important problem in software *practice* and *research*, we also encountered them in *teaching*. Typically, the teaching staff grades students' solutions to programming assignments using automated tests. These tests can be flaky, and as a result students with correct solutions may have failing tests, and students with incorrect solutions may have passing tests. We discuss more details from one recent course in Section IV-B. Besides educating people about flaky tests, how can we help practitioners in the real world and the students in our courses to detect more flaky tests faster?

We propose a simple technique, called NONDEX, to detect flaky tests due to ADINS code. We implement NONDEX for Java, but it can be easily generalized to any other language. In a nutshell, we identify 31 methods with non-deterministic specifications as discussed in Section III-A, wrote models for these methods to produce various non-deterministic choices,

https://github.com/TestingResearchIllinois/NonDex

README.md

build passing | build passing | code climate 86 issues | code quality D

NonDex is a tool for detecting and debugging wrong assumptions on under-determined Java APIs. An example of such an assumption is when code assumes the order of iterating through the entries in a java.util.HashMap is in a specific, deterministic order, but the specification for java.util.HashMap is under-determined and states that this iteration order is not guaranteed to be in any particular order. Such assumptions can hurt portability for an application when they are moved to other environments with a different Java runtime. NonDex explores different behaviors of under-determined APIs and reports test failures under different explored behaviors; NonDex only explores behaviors that are allowed by the specification and any test failure indicates an assumption on an under-determined Java API. NonDex helps expose such brittle assumptions to the developers early, so they can fix the assumption before it becomes a problem far in the future and more difficult to fix.

Supported APIs:

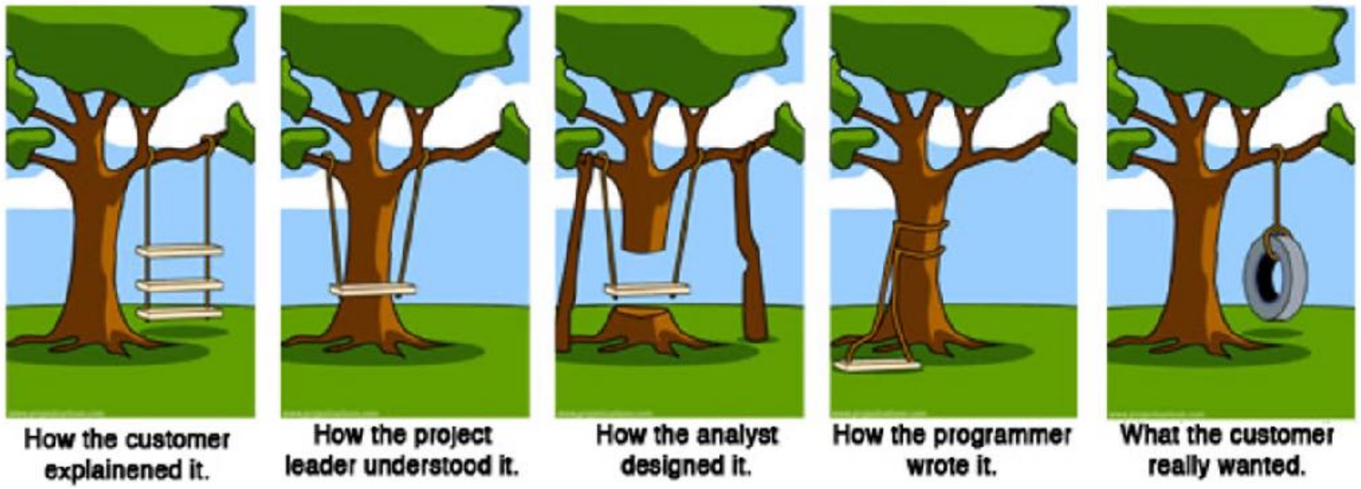
The list of supported APIs can be found here

Prerequisites:

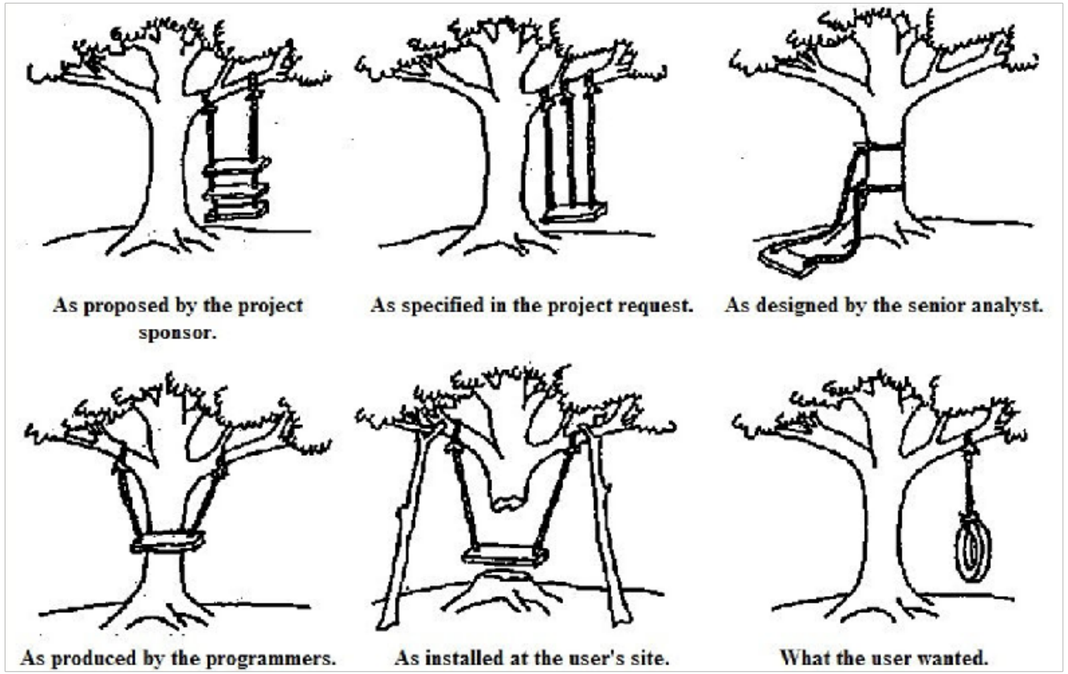
```
- Java 8 (Oracle JDK, OpenJDK).
- Surefire present in the POM.
```

Build (Maven):

Software Engineering in a Meme



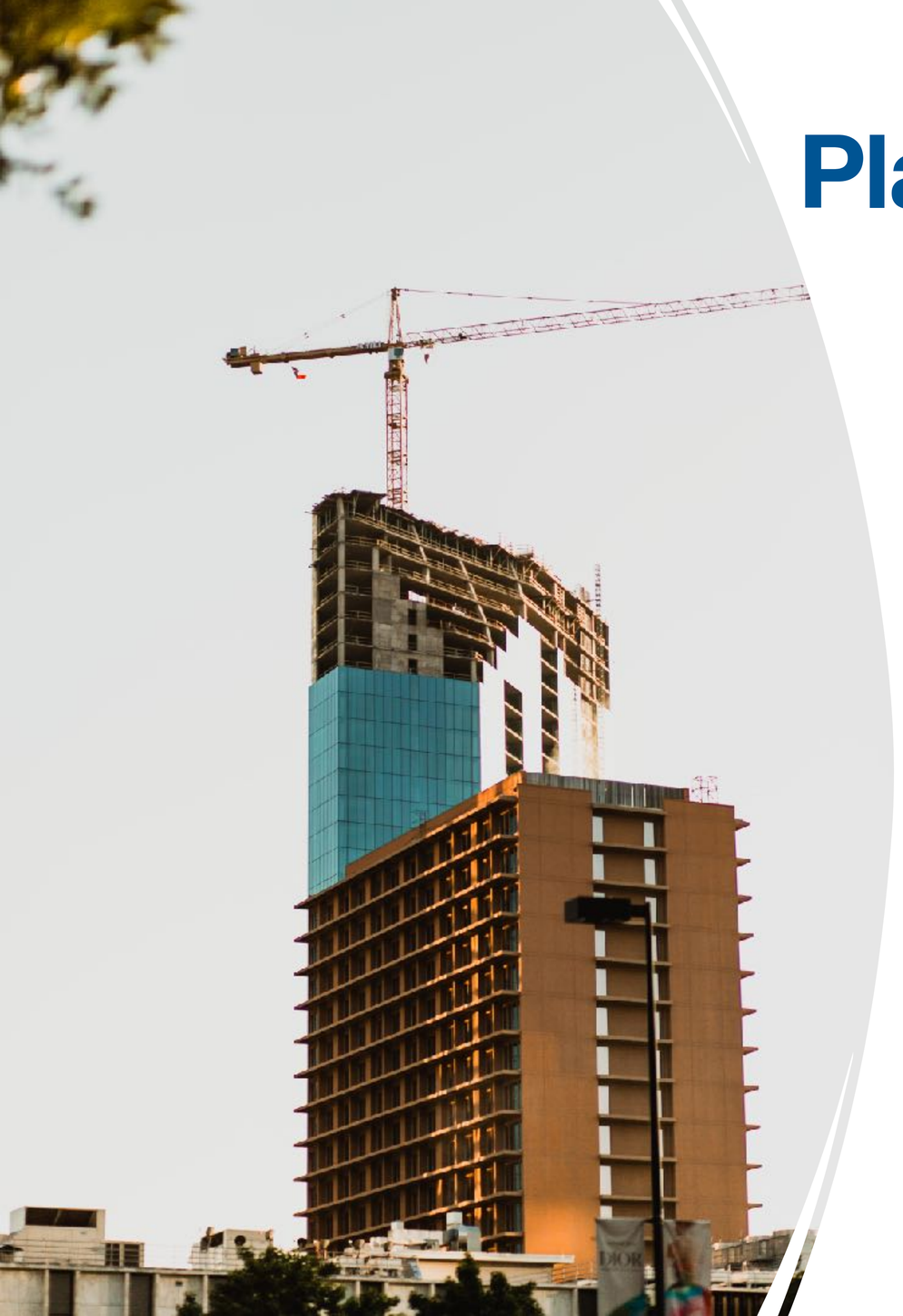
Software Engineering in an Ancient Meme



History of the tire swing meme

Brainstorm: What is a software process?

Why explicitly define and discuss the process?

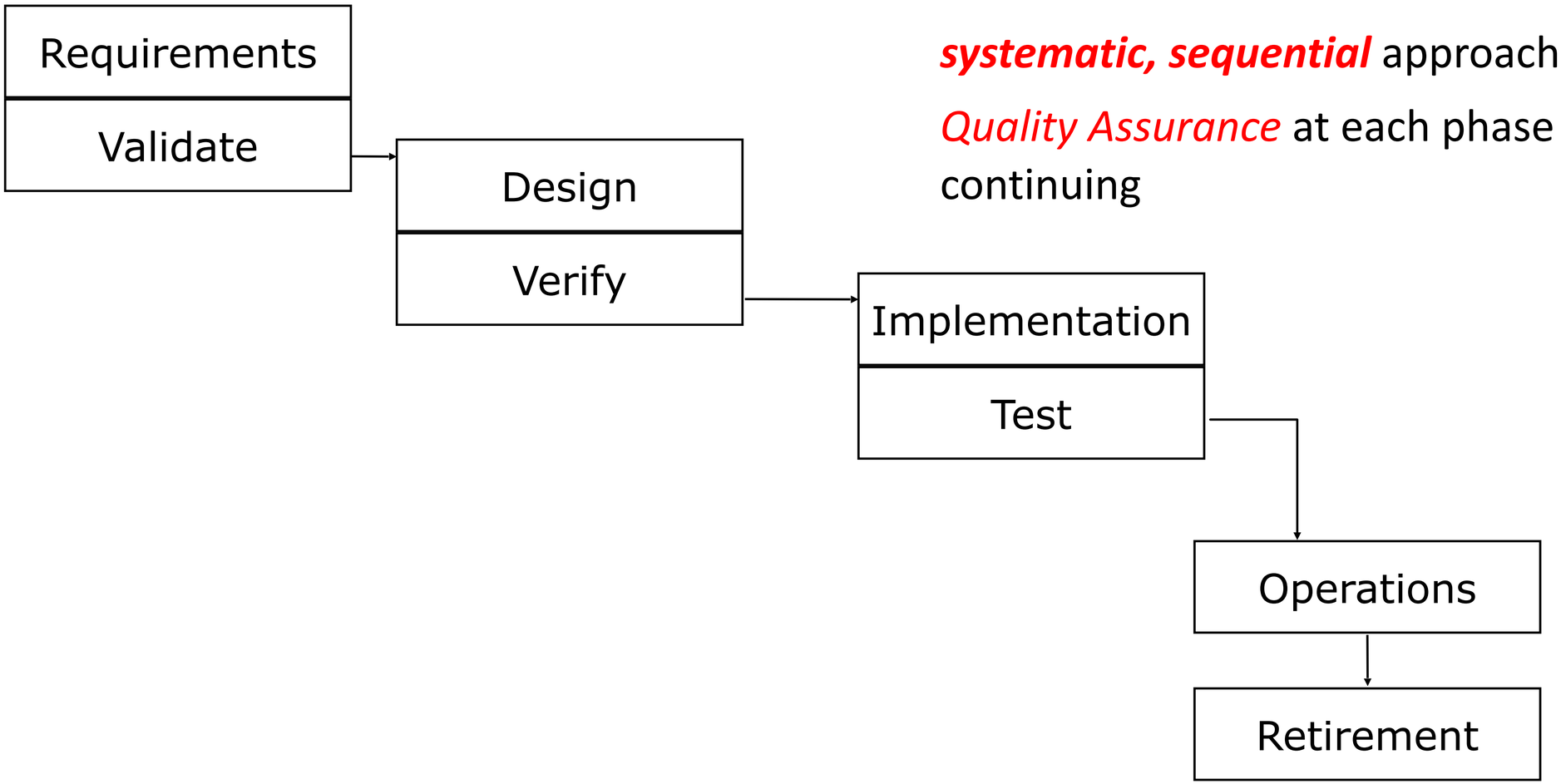


Planning Engineering Projects

In contrast to software:

- Mechanical in nature
- Highly standardized:
  - Design process
  - Materials
  - Construction process

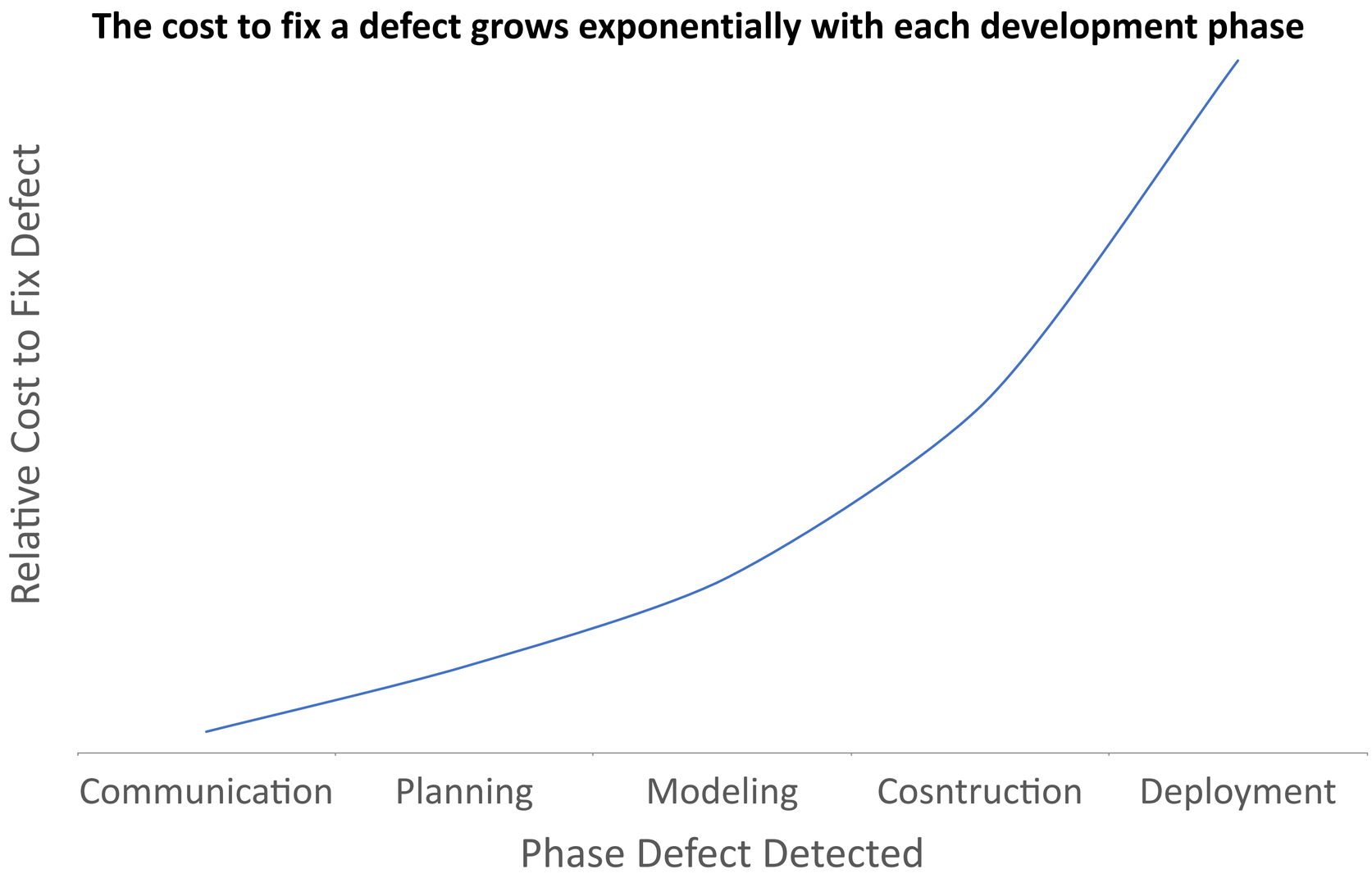
Software Process: Waterfall (~1970)



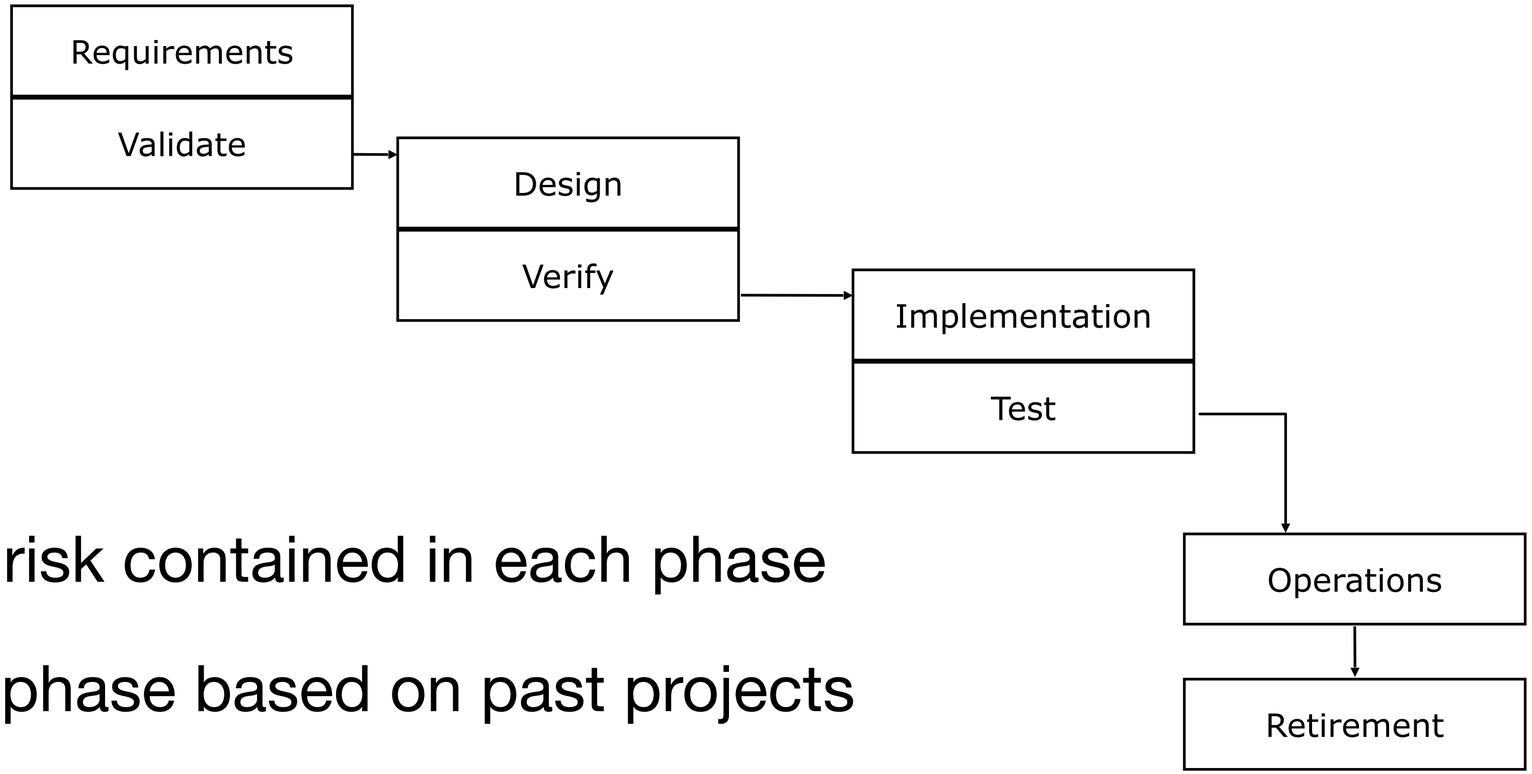
systematic, sequential approach

Quality Assurance at each phase before continuing

Waterfall Model: Risk Assumptions



Waterfall Process Improves on Code + Fix



- Measurable progress with risk contained in each phase
- Possible to estimate each phase based on past projects
- Division of labor: Natural segmentation between phases

Waterfall Model adds process overhead

Since formal quality assurance happens at extremely detailed…

- Requirements documents
- Design documents
- Source code with documentation

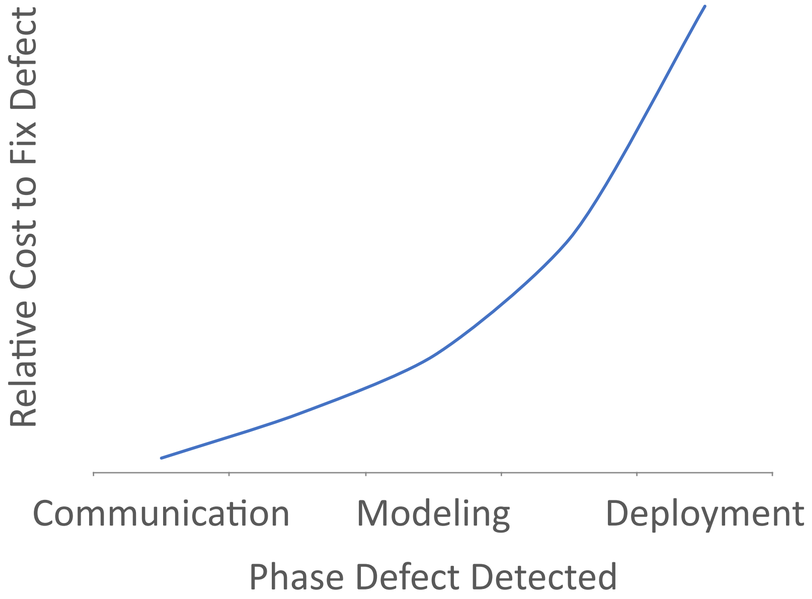
Waterfall Model Reduces Risk by Preventing Change

Traditional waterfall model: no way to go back “up”

“There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.”

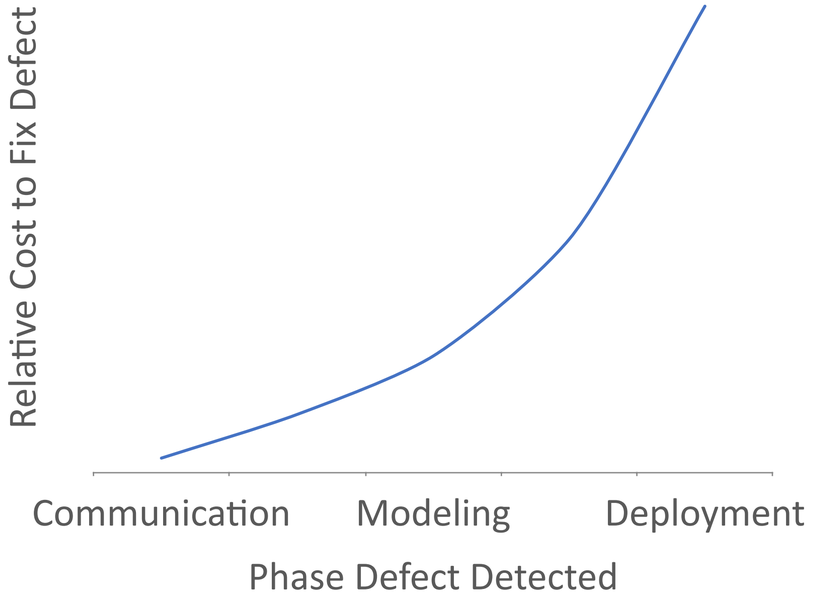
Waterfall Model: Applications

- What projects would this work well in?
  - Projects with tremendous uncertainty
  - Projects with long time-to-market
  - Projects that need extensive QA of requirements and design
  - Projects for which the expense of the planning is worth it
  - Classic examples: military/defense
    - Warship that needs to have component interfaces last 80 years
    - Spacecraft?

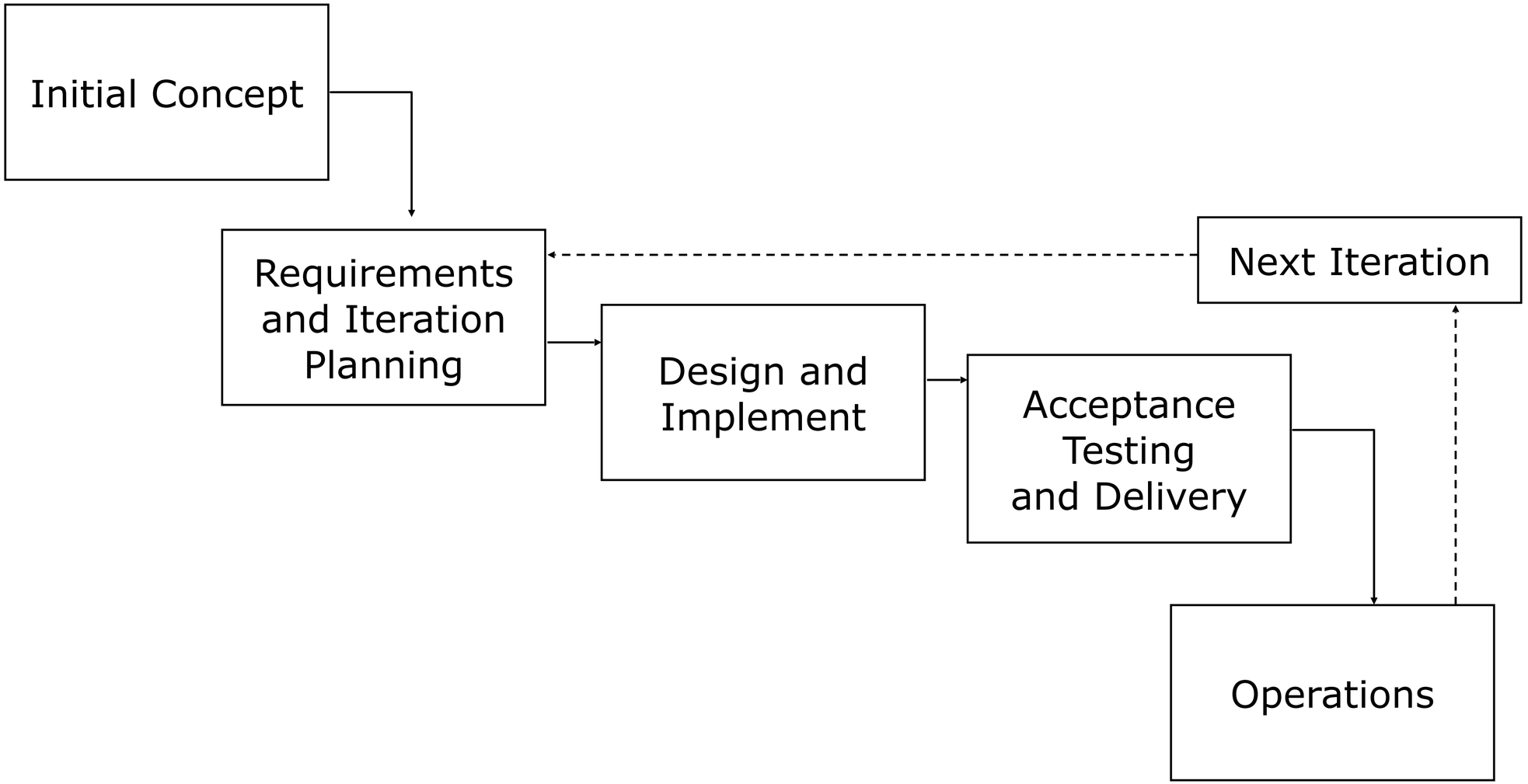


Waterfall Model: Wasted Work Product

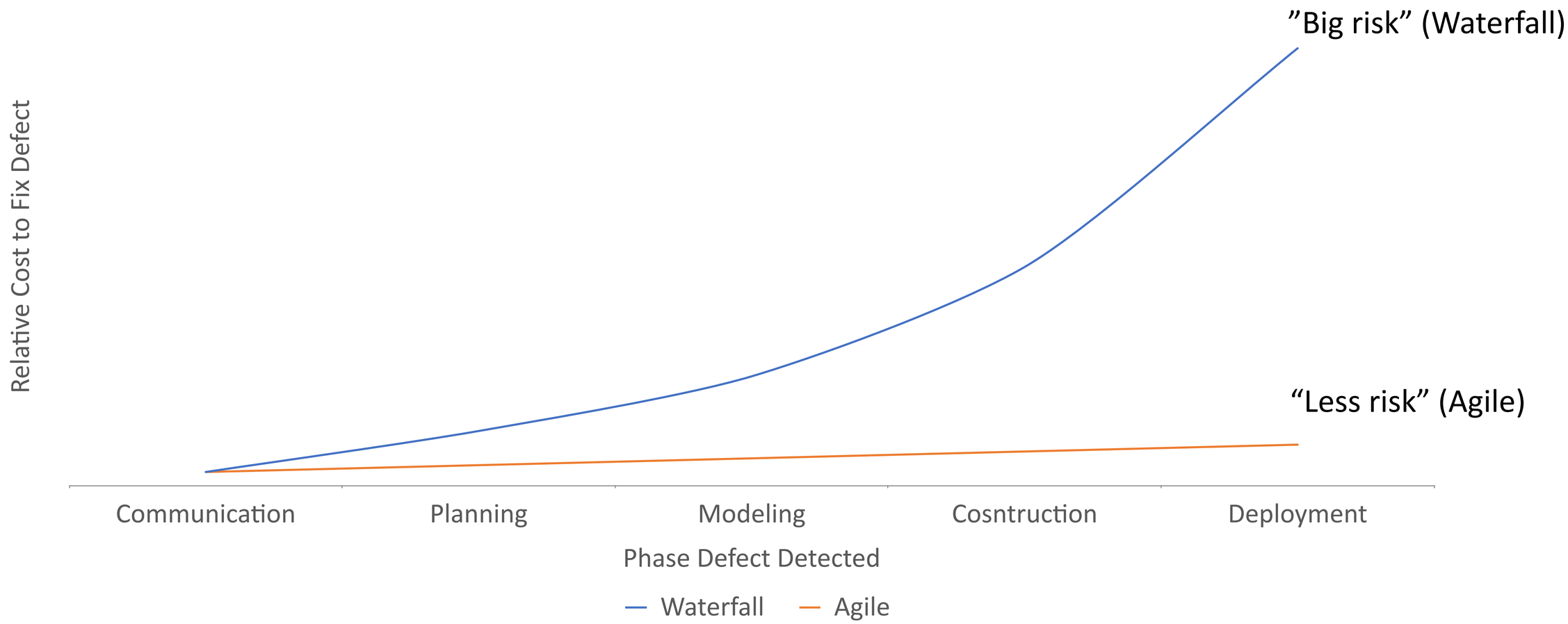
- Wasted productivity can occur through each phase's QA process:
  - Requirements that become obsolete
  - Elaborate architectural designs never used
  - Code that sits around not integrated and tested in production environment, eventually discarded
  - Documentation produced per requirements, but never read
- What if we could eliminate that waste, and reduce the cost of defects later in development cycle?
  - Example: with shorter time-to-market?



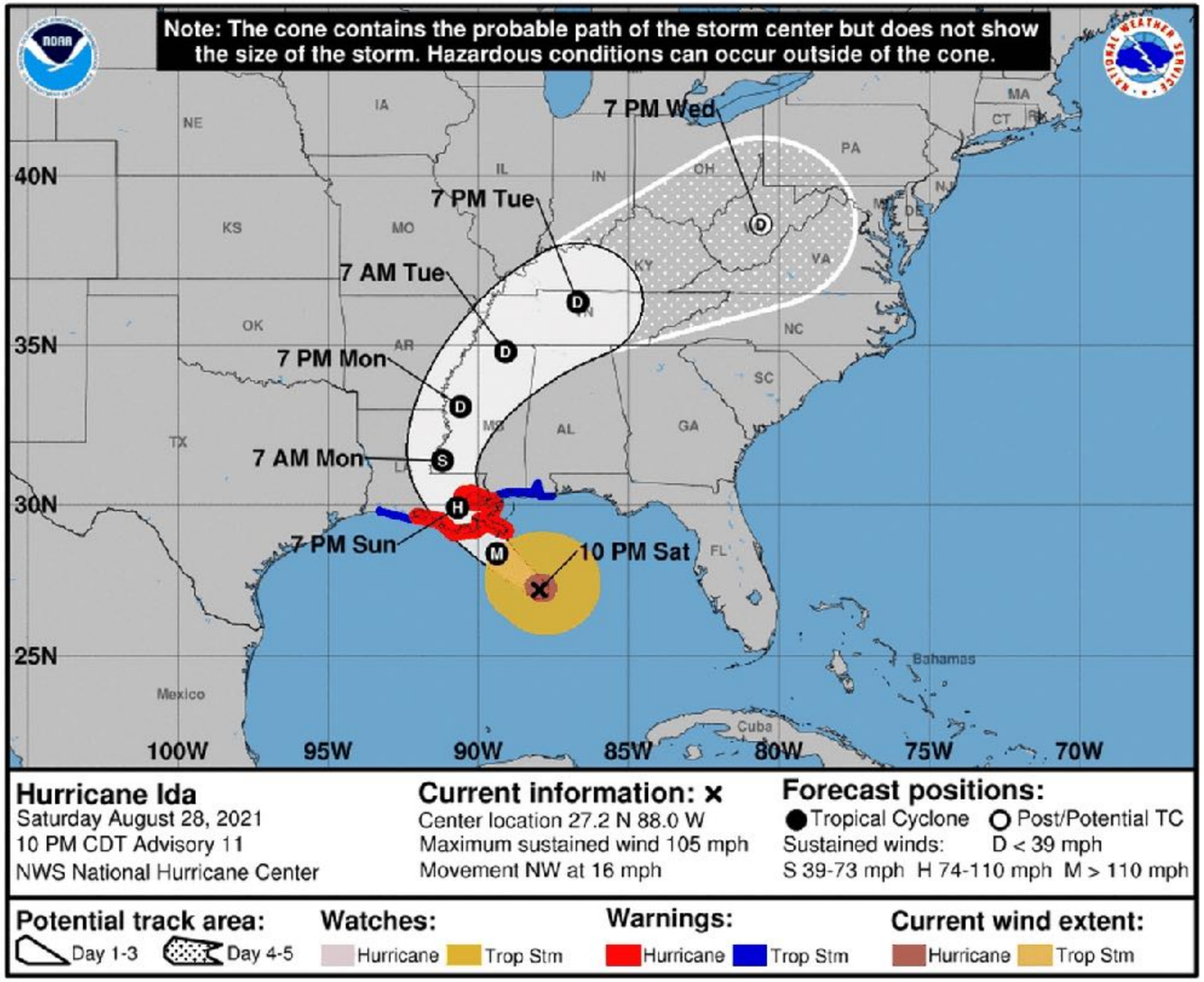
Waterfall Variation: Iterative Process (~1980s)



The Agile Model Reduces Risk by Embracing Change (~2000)

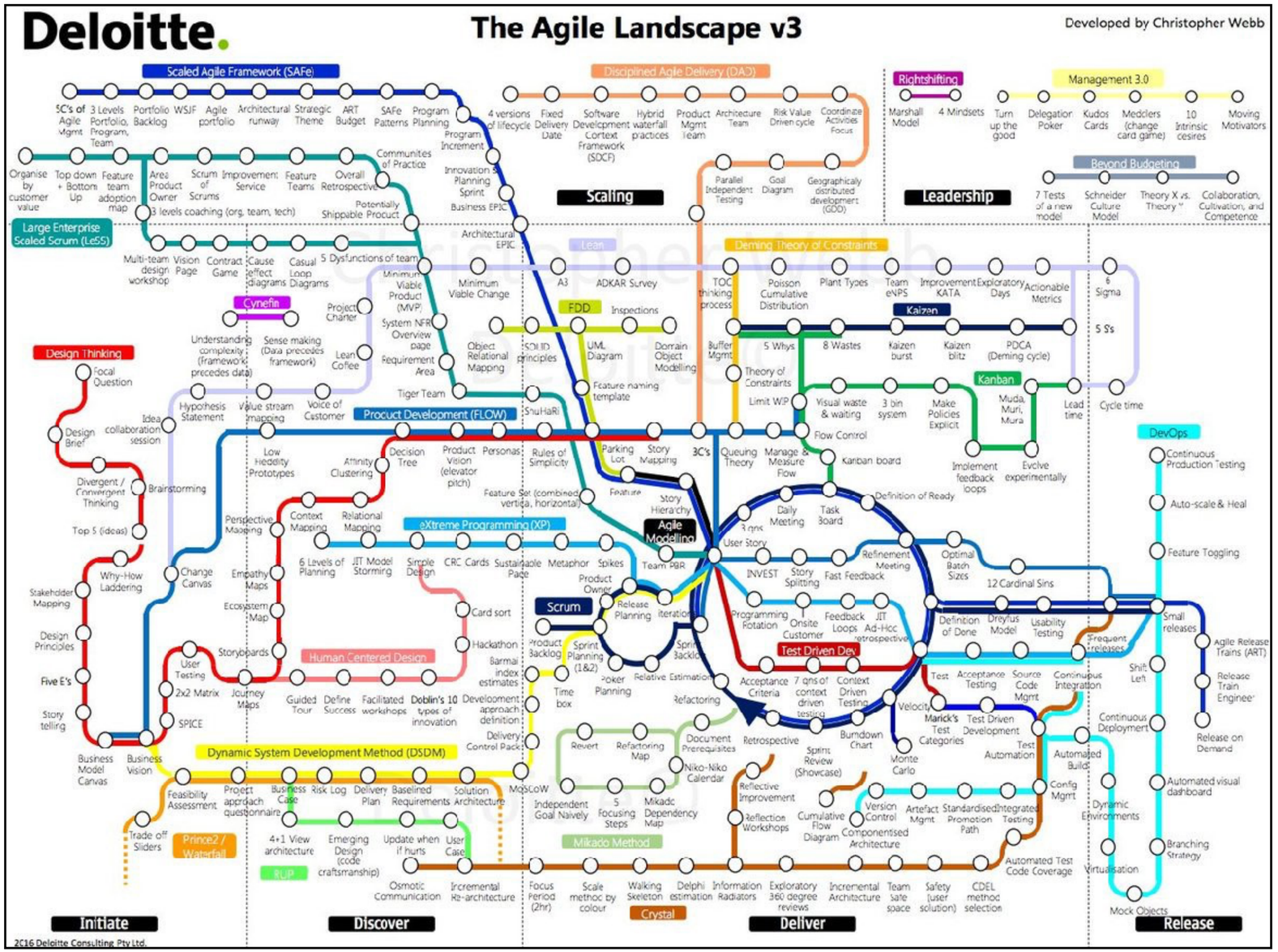


Agile Recognizes that Long-Term Planning is Hard.



Agile Empowers Workers to Improve Processes: Toyota Production System (1990’s)

Agile can be a buzzword



Graphic © Christopher Webb

What is a good software engineering process?

Disclaimer: Software Engineering is full of opinions

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand”

- Martin Fowler

Agile Manifesto

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to 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

That is, while there is value in the items on
the right, we value the items on the left more.

https://agilemanifesto.org

Continuous Improvement is a Key Agile Value

- “Collective ownership”
- Functional and non-functional correctness is checked on the cheap, and often
- Regular checkpoints to evaluate the efficacy of the process and solicit improvements

Example Agile Process: XP

Building Software For Shifting/Unknown Requirements

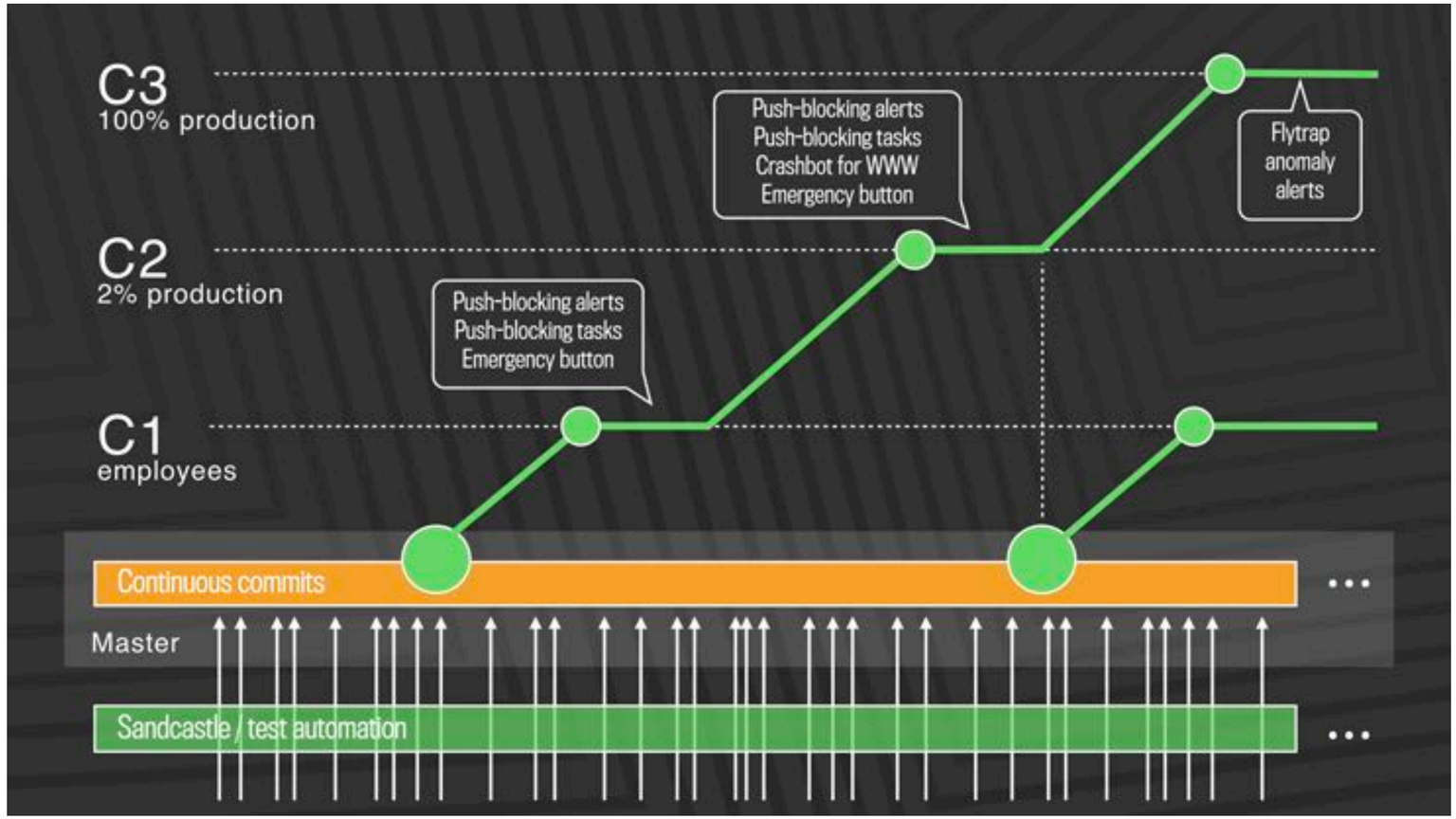
"The development of a piece of software changes its own requirements. As soon as the customers see the first release, they learn what they want in the second release...or what they really wanted in the first. And it's valuable learning, because it couldn't have possibly taken place based on speculation. It is learning that can only come from experience. But customers can't get there alone. They need people who can program, not as guides, but as companions."

- Kent Beck, in “eXtreme Programming eXplained”

Agile Practice: Small, Continuous Releases

Checking correctness and quality of the complete system

- System is put into production before solving the full problem – new releases that add value happen fast (monthly, daily, hourly…)
- Multiple release phases for fast-feedback
- More in weeks 8 & 11



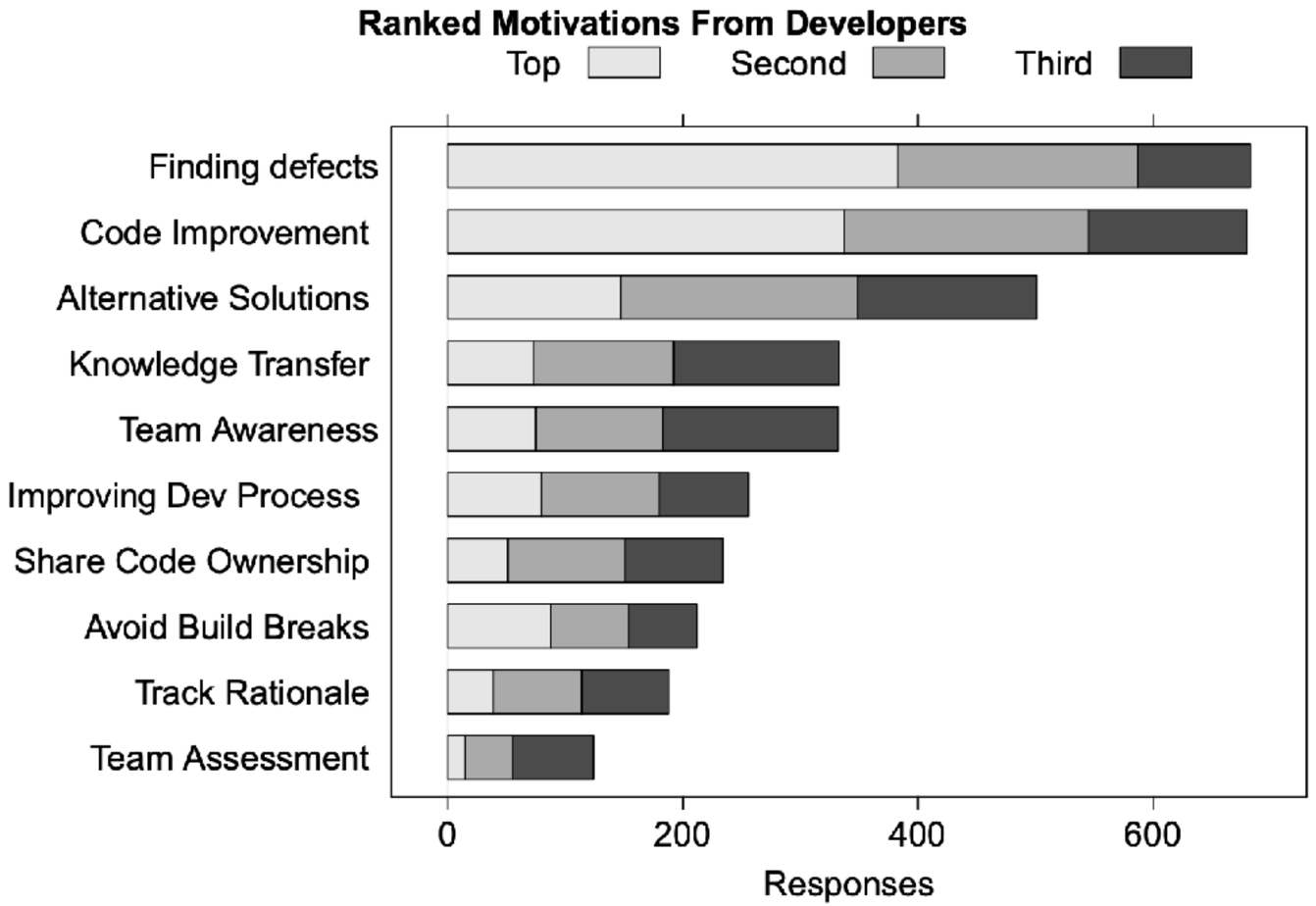
https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

Agile Practice: Code Review

A Formal Process

- A code review is the process in which the author of some code is asked to explain it to their peers:
  - What purpose the code has;
  - How the code accomplishes this purpose;
  - How the author is confident of this information,
    - E.g., show results of running tests (CI results)
- A code review often concerns a code change (“diff”)

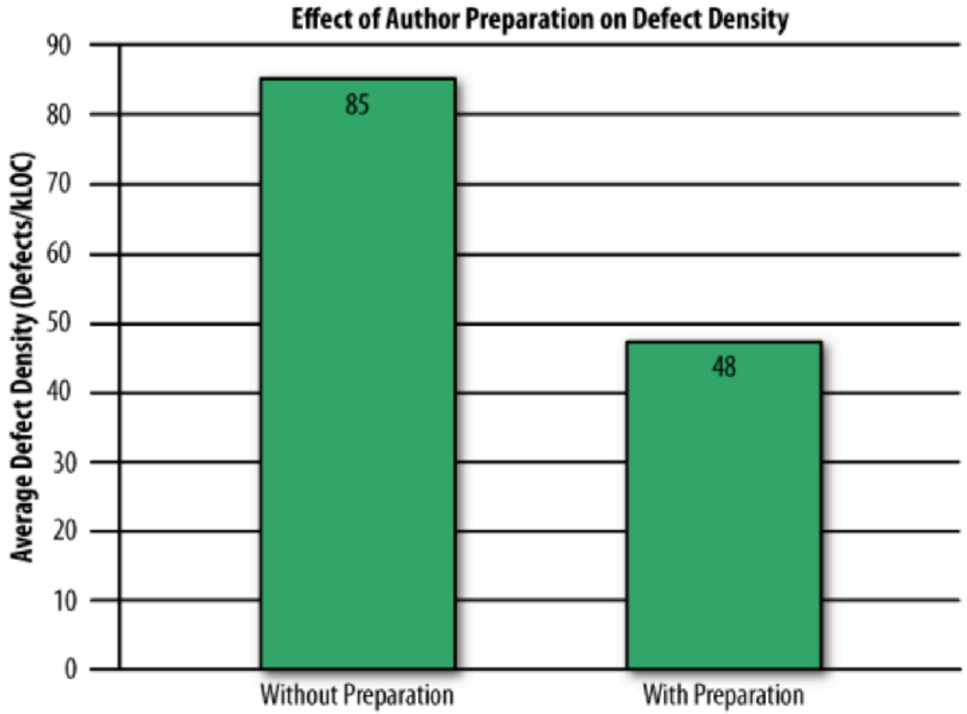
SE Research Question: Why Do Code Review?



"Expectations, Outcomes, and Challenges of Modern Code Review", Bacchelli & Bird, ICSE 2013

SE Research Question: Does Self Review Detect all Defects?

Study of 300 reviews at Cisco in 2006



Even if developers pre-review their code, many defects still found in peer review

"Best Kept Secrets of Peer Code Review", Jason Cohen, SmartBear Software, 2006

Topics we'll address this term

- Software Process
- Modularity and Design
- Mining Software Repositories and Open Source Culture
- Testing, Continuous Integration and Devops
- Expertise and knowledge sharing
- Security
- Software engineering in specific domains

Paper Reading Advice

- Note: Might need to re-read an article multiple times, especially if you are not familiar with background material
- As you read, consider the following questions:
  1. What is the motivation for this work?
  2. What is the problem that is being solved?
  3. What is hard about that problem?
  4. What is the proposed solution?
  5. How is that solution achieved?
  6. How is that solution evaluated?
- After reading, reflect:
  1. Is this a problem worth solving?
  2. Is the solution a good idea?
  3. Do you see limitations to the problem, or the solution?
  4. Is future work needed to fit this research prototype into the real world problem domain?
  5. What questions does this paper leave you with?
- More advice at https://cseweb.ucsd.edu/~wgg/CSE210/howtoread.html

Topic Interest Poll/Discussion