CS 4350: Fundamentals of Software Engineering
Lesson 3.2: Software Development Processes

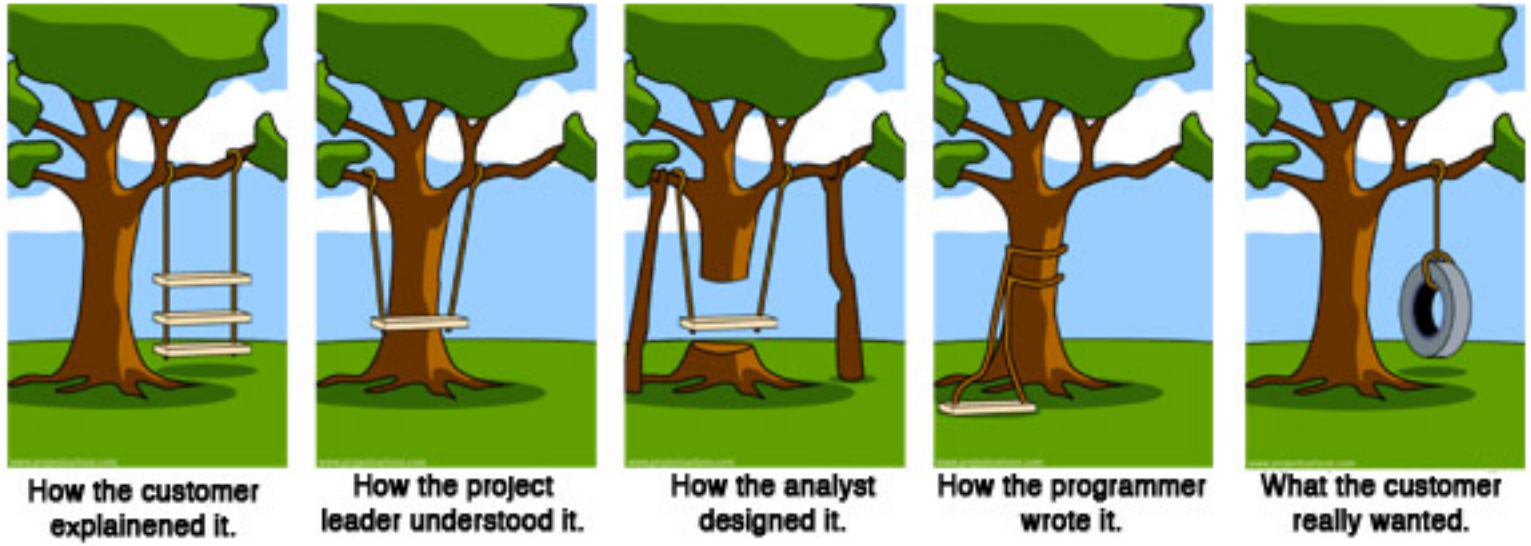
Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, Mitch Wand

Khoury College of Computer Sciences

Learning Goals for this Lesson

- At the end of this lesson, you should be able to
  - Know the basic characteristics of the waterfall software process model
  - Be able to explain when the waterfall model is appropriate and when it is not
  - Understand how the waterfall and agile models manage risk
  - Be able to explain how agile process instill quality, including through test driven development

Review:
How to make sure we are building the right thing

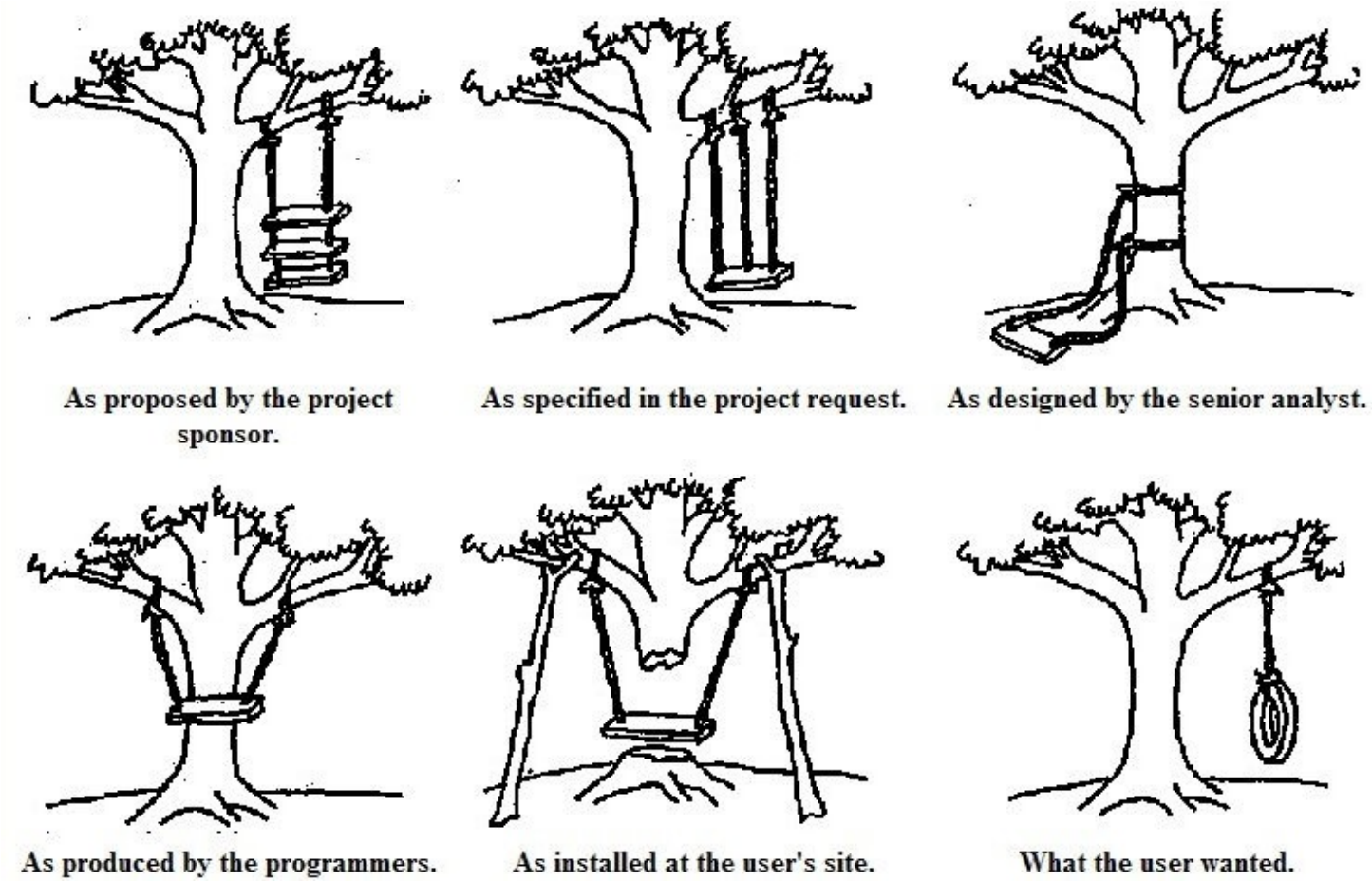


Requirements Analysis

Planning & Design

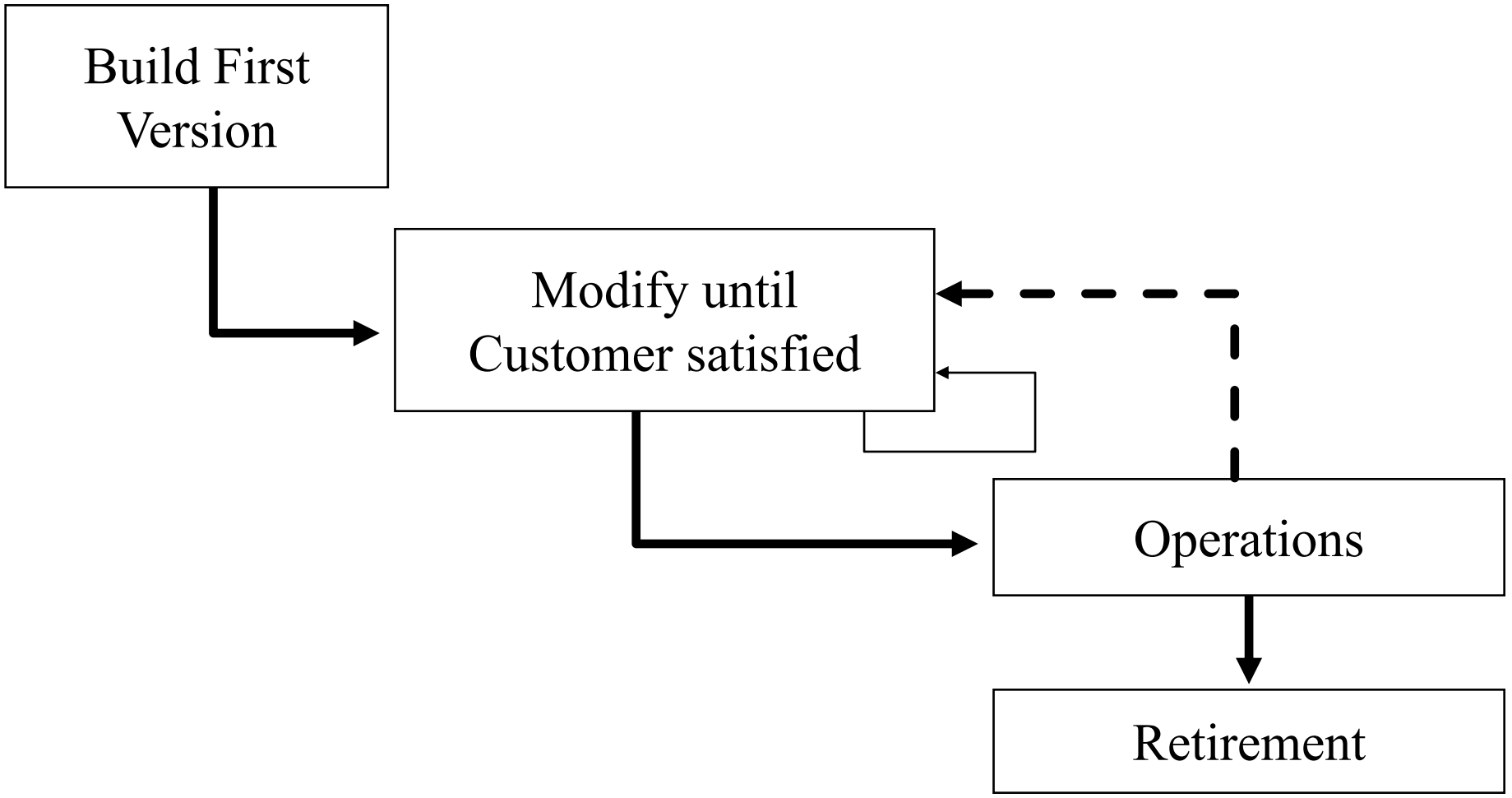
Implementation

What has changed in the past 50 years?



History of the tire swing meme

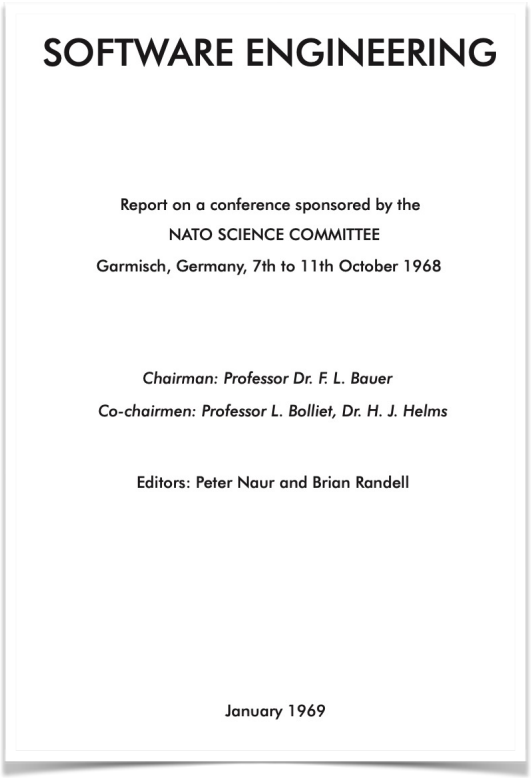
Software Process: Code + Fix



A brief history of software planning

NATO conference on Software Engineering + Outcomes

- 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

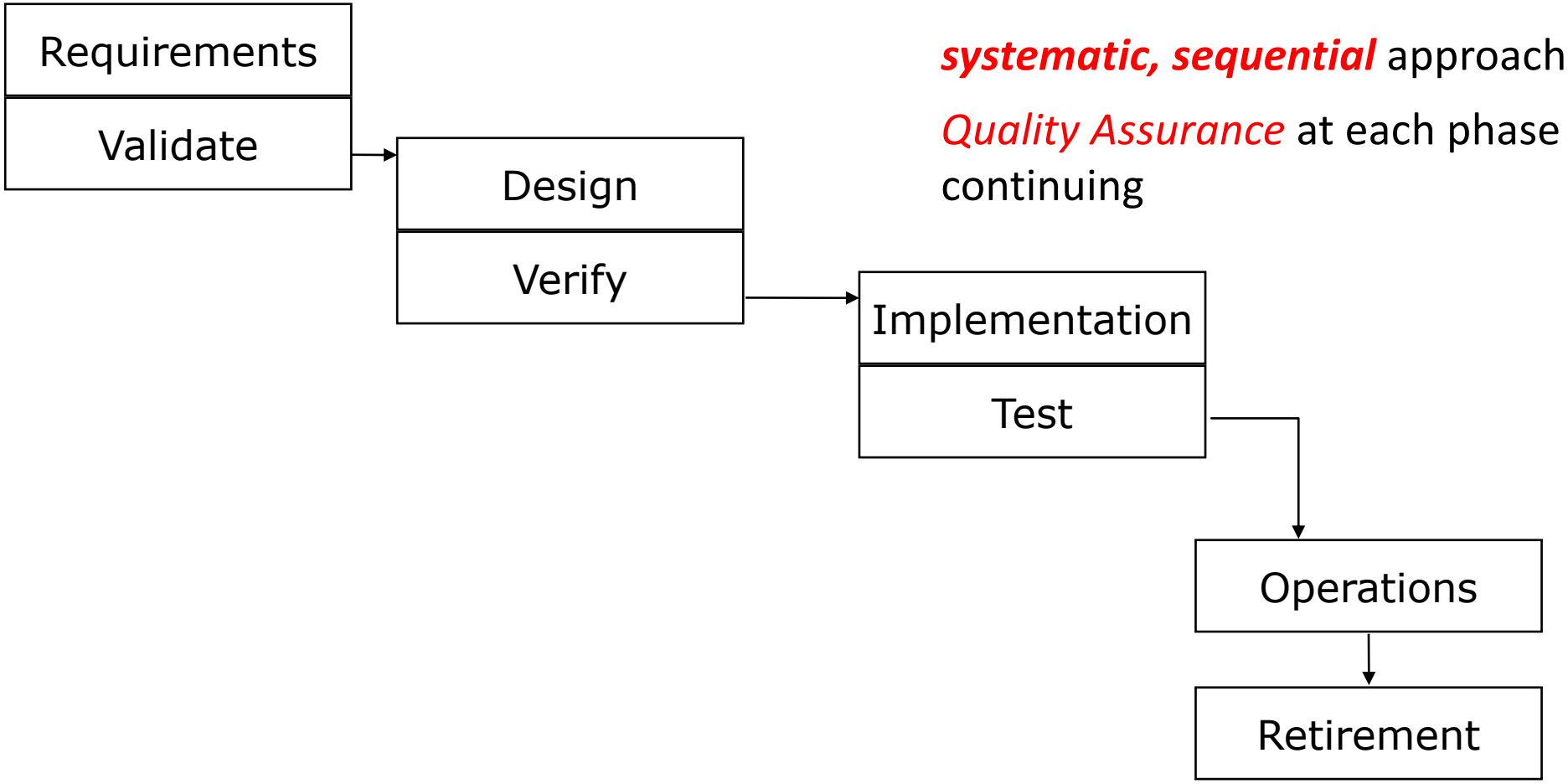
A call to action: We must study *how to build software*

Planning Engineering Projects

In contrast to software:

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

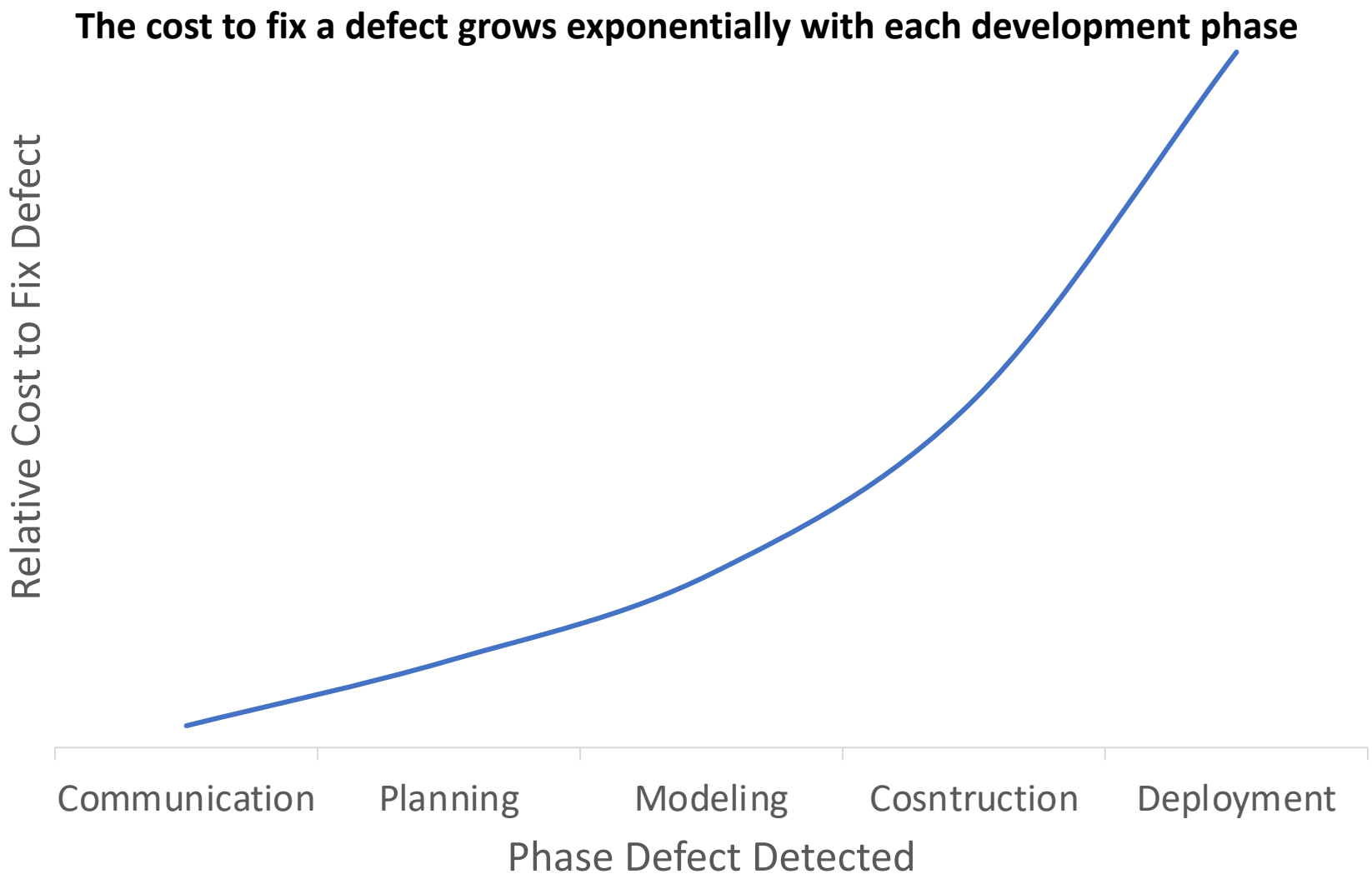
Software Process: Waterfall (~1970)



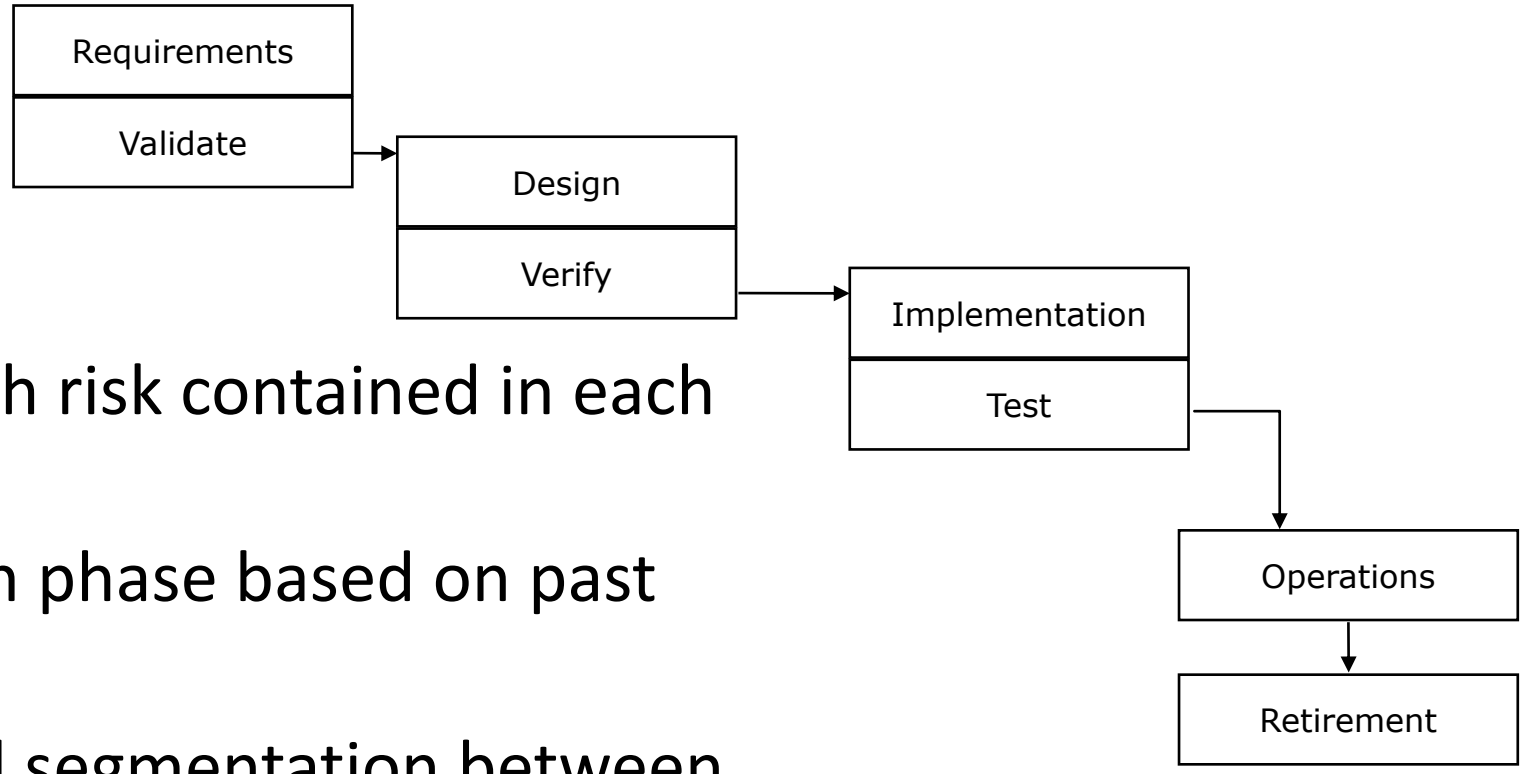
systematic, sequential approach

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 each phase, it's necessary to produce extremely detailed…

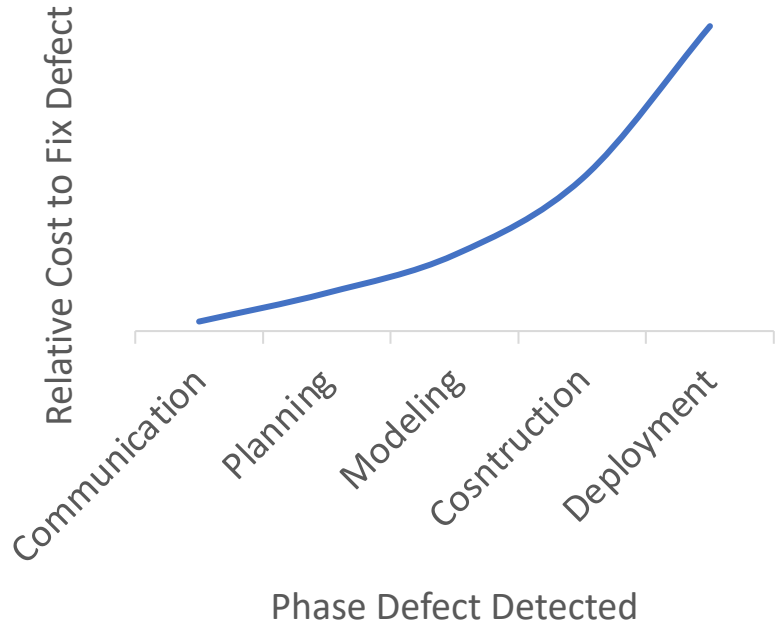
- Requirements documents
- Design documents
- Source code with documentation

Waterfall Model Reduces Risk by Preventing Change

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

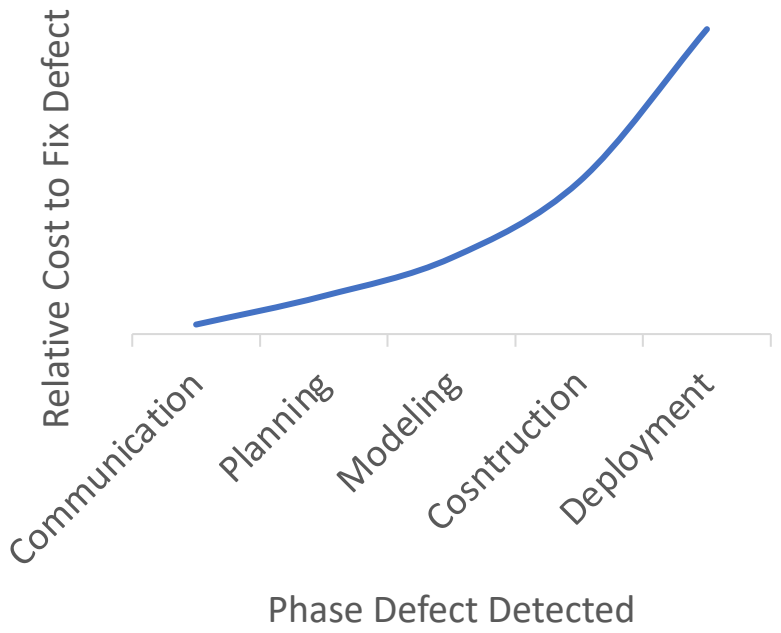
Waterfall Model: Applications

- 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?

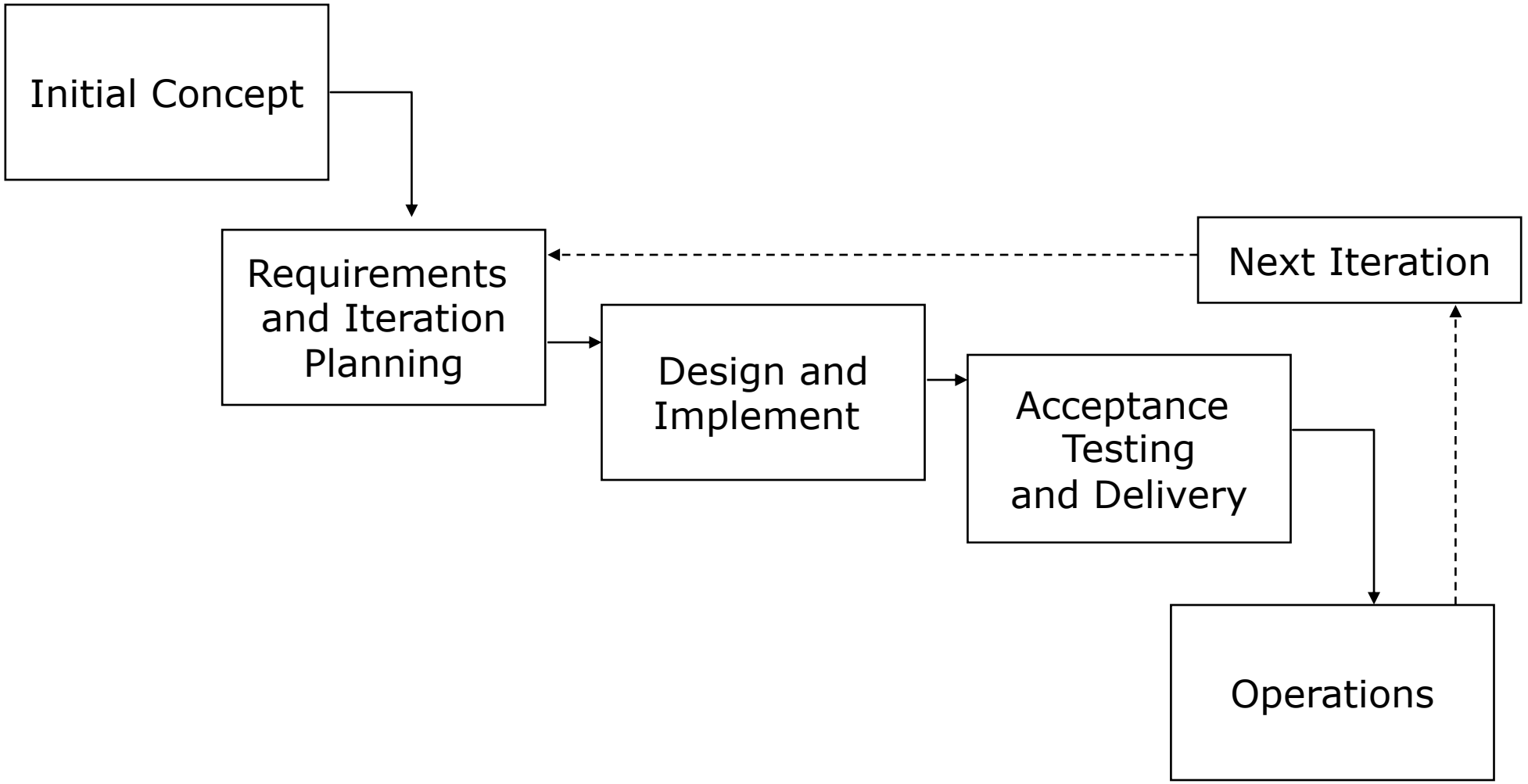


Waterfall Model: Wasted Work Product

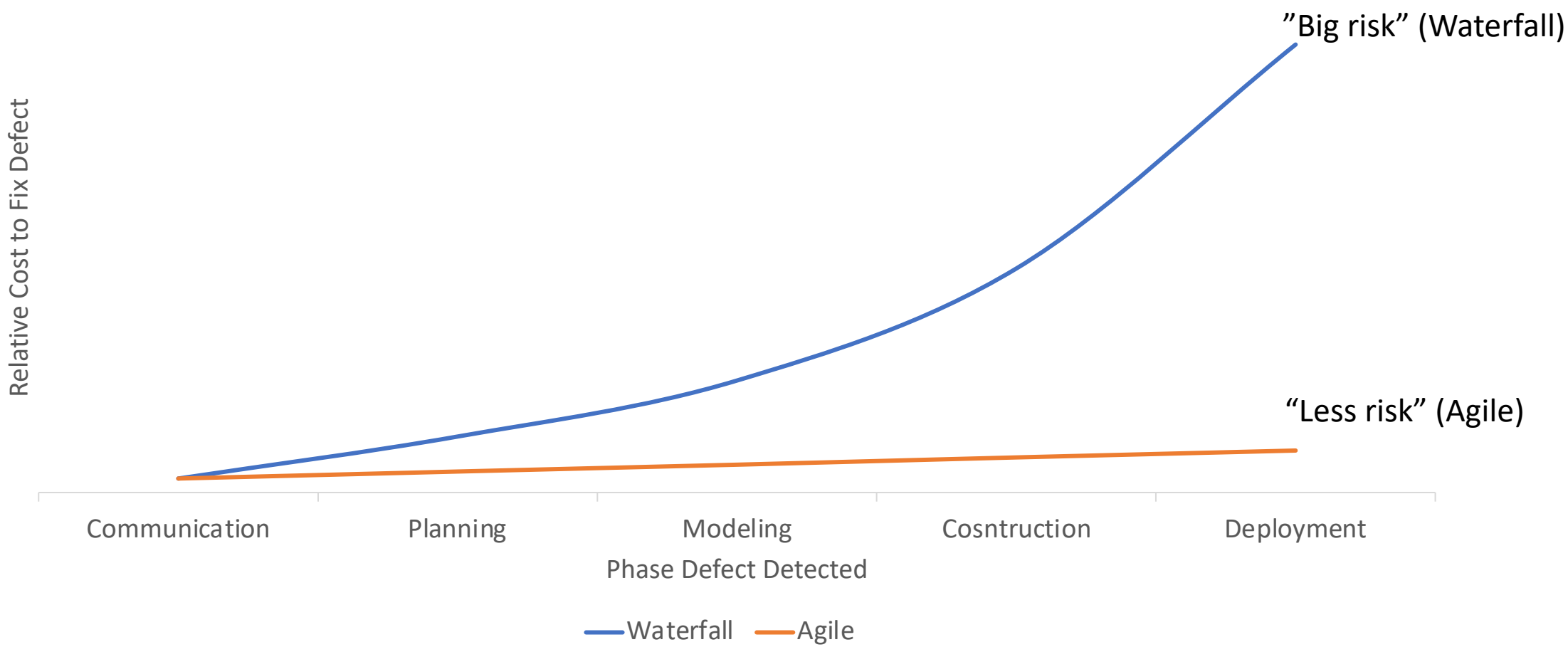
- Wasted productivity can occur through each phase's QA process:
  - Requirements that become obsolete
  - Elaborate architectural designs never used
  - Code that sits around not integrated and tested in production environment, eventually discarded
  - Documentation produced per requirements, but never 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 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

Example Agile Process: XP

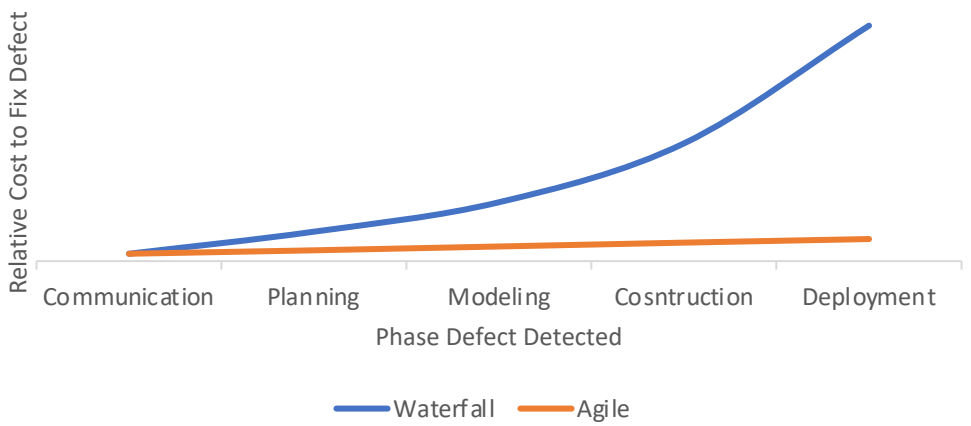
"The development of a piece of software changes its own requirements. As soon as the customers see the first release, they learn what they want in the second release...or what they really wanted in the first. And it's valuable learning, because it couldn't have possibly taken place based on speculation. 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 Values Increase Efficiency and Embrace Change

Compare to problems in waterfall:

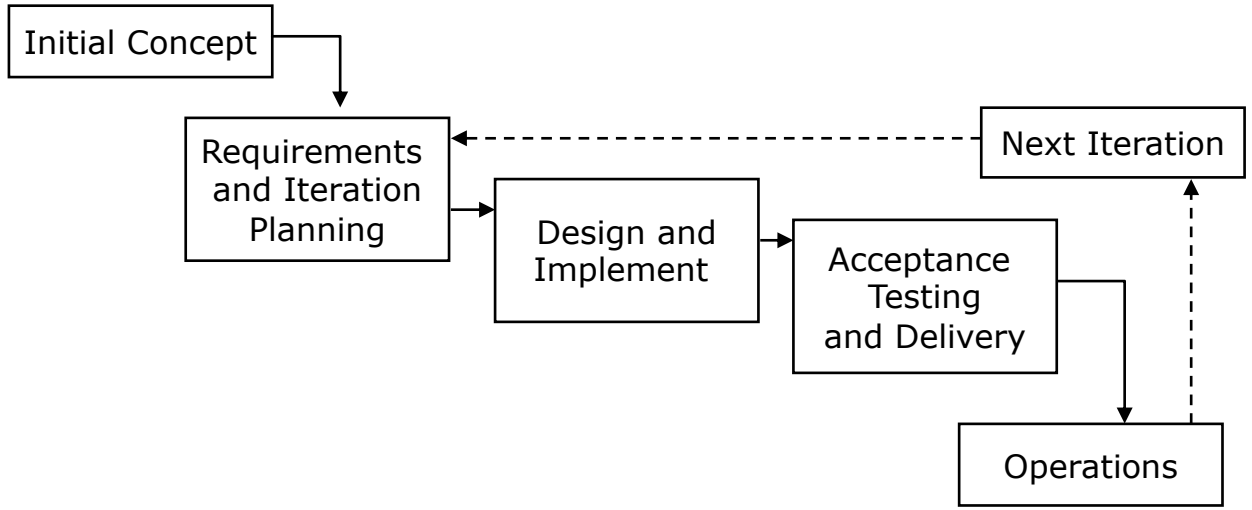
- Requirements that become obsolete
  - Don’t make detailed requirements until you need them
- Elaborate architectural designs never used
  - Don’t design until you need
- Code that sits around not integrated and tested in production environment, eventually discarded
  - Integrate and test continuously
- Documentation produced per requirements, but never read
  - Don’t require documentation



Agile Processes are Iterative

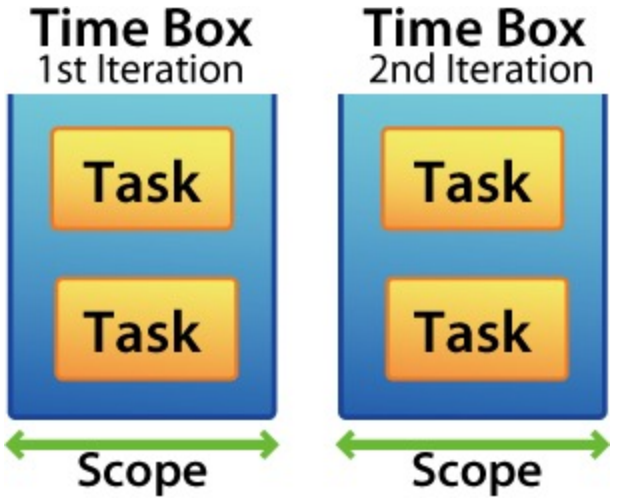
Agile Process Model

Iterative Waterfall Model



Agile Processes Reduce Risk by Time Boxing

- Each “iteration” is called a “sprint”
- Each sprint has a fixed duration
- Scope of features in a sprint is determined by the team
- Key insight: planning might be a guess at first, but gets better with time
- More on agile planning & estimation in Lesson 3.3

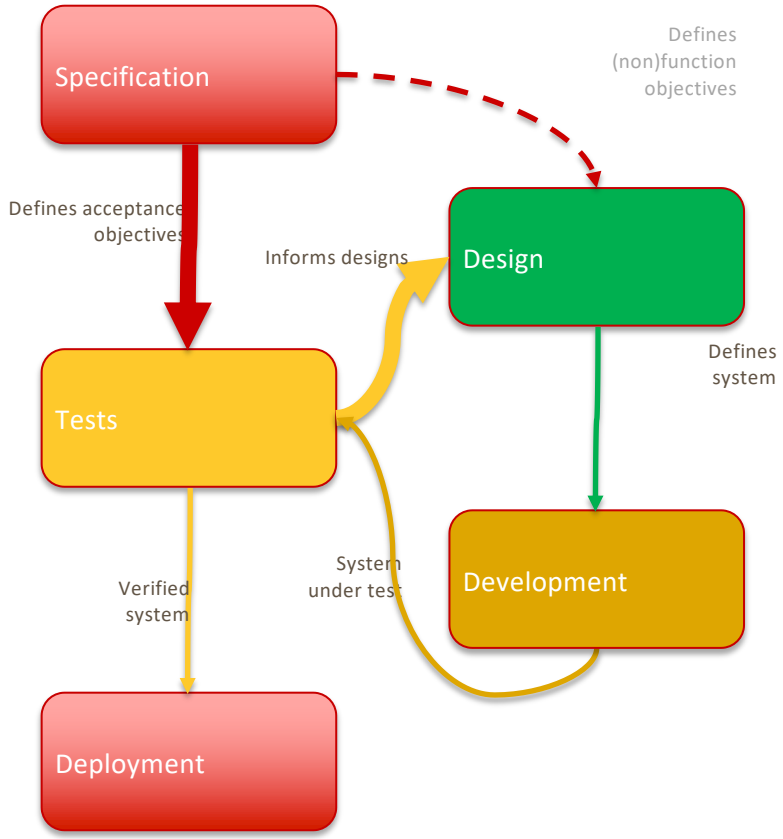


Agile Practice: Everyone is Responsible for Quality

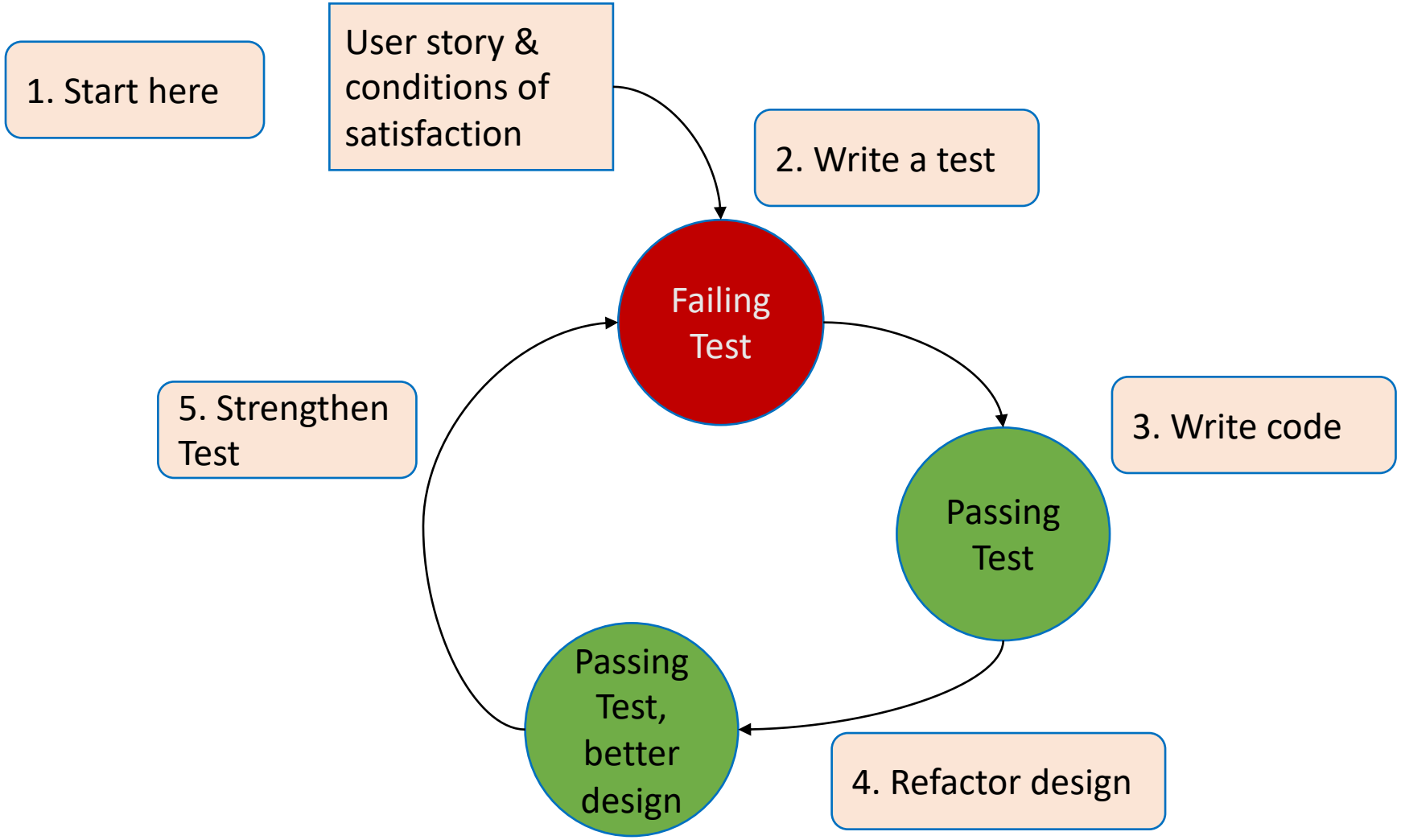
- “Collective ownership”
- Requirements (user stories) are developed collaboratively with customer, and are *negotiable* (INVEST qualities)
- Functional and non-functional correctness is checked *on the cheap*, and often
- Developers improve code anywhere in the system if they see the opportunity
- Many parallels with “Toyota Process System;” a variety of other software processes developed in the 90’s share these basic values

Agile Practice: Test Driven Development (TDD)

- Puts test specification as the critical design activity
  - Understands that deployment comes when the system passes testing
- The act of defining tests requires a deep understanding of the problem
- Clearly defines what success means
  - No more guesswork as to what “complete” means

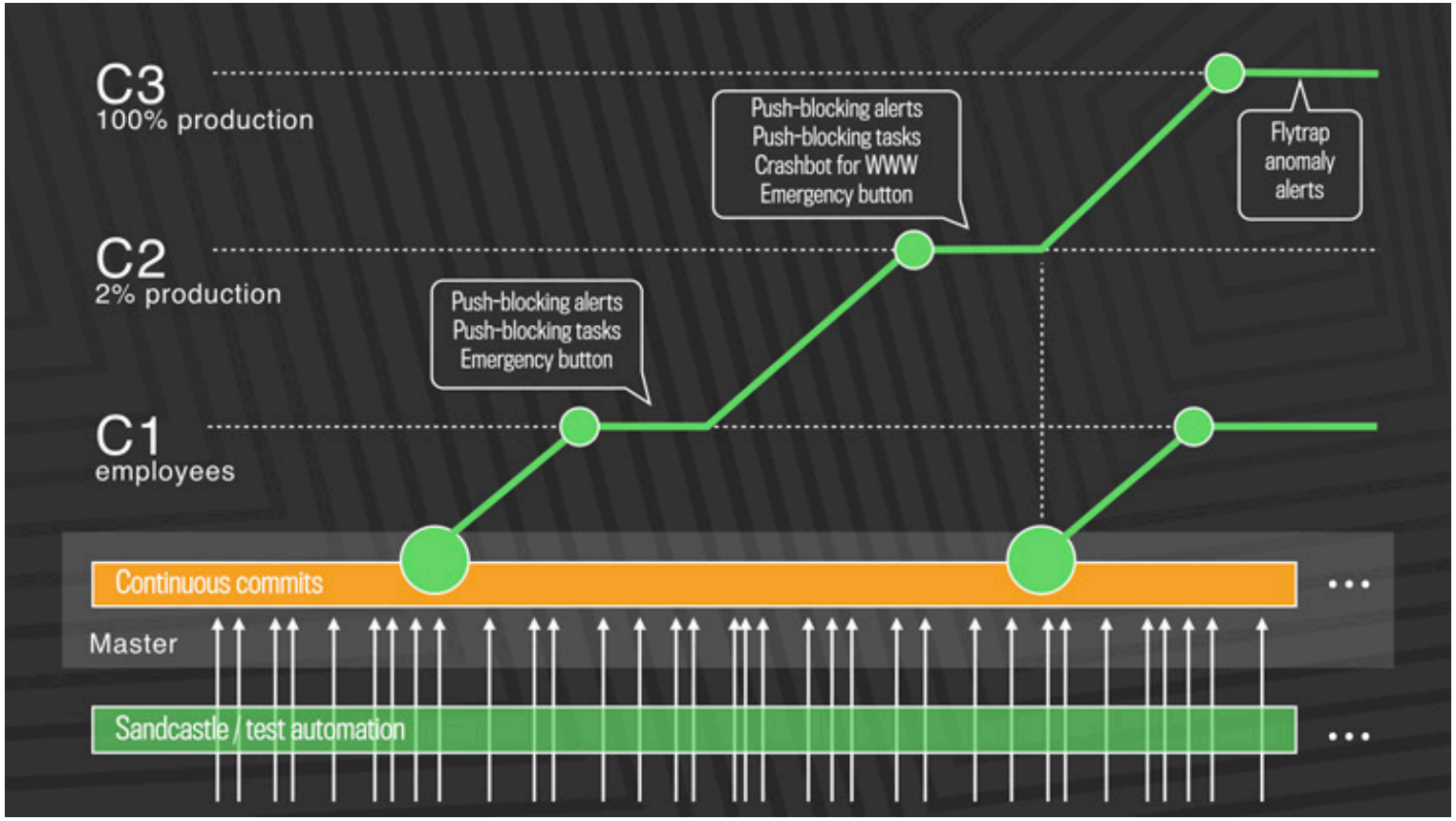


Agile Practice: Test Driven Development (TDD)



Agile Practice: Small, Continuous Releases

- 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 week 8



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

Learning Goals for this Lesson

- At the end of this lesson, you should be able to
  - Know the basic characteristics of the waterfall software process model
  - Be able to explain when the waterfall model is appropriate and when it is not
  - Understand how the waterfall and agile models manage risk
  - Be able to explain how agile process instill quality, including through test driven development