# CS 4530: Fundamentals of Software Engineering
# Module 7.2: Software Development Processes

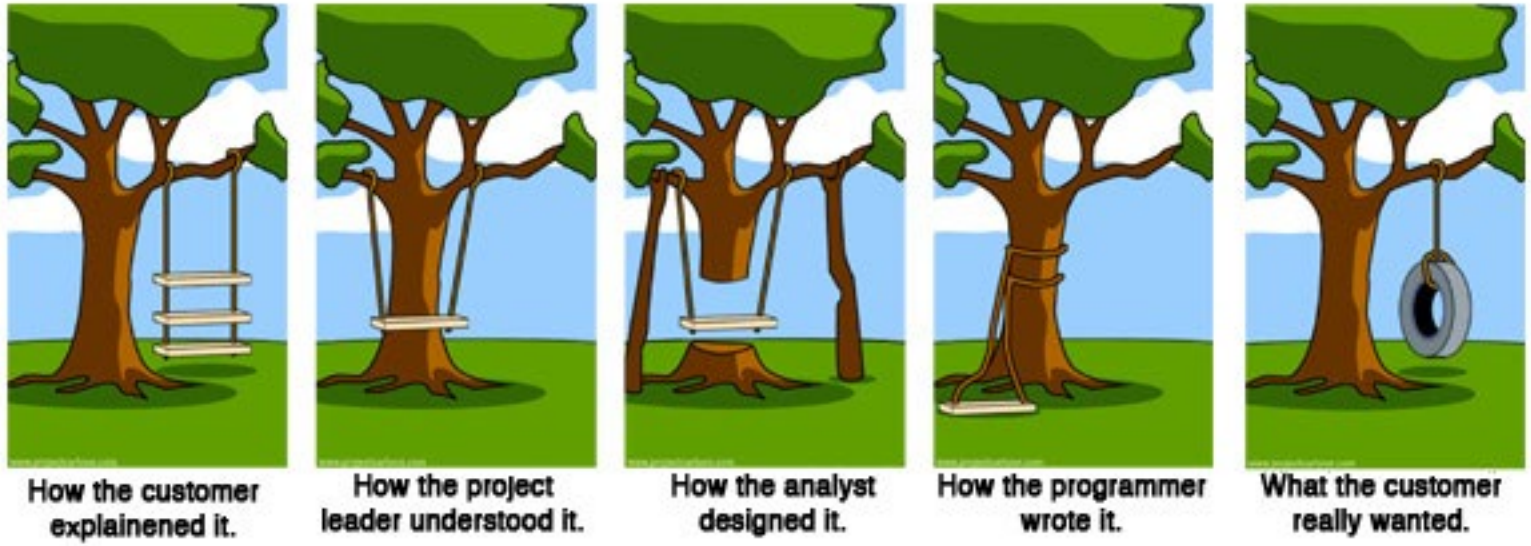
Adeel Bhutta, Rob Simmons and Mitch Wand

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

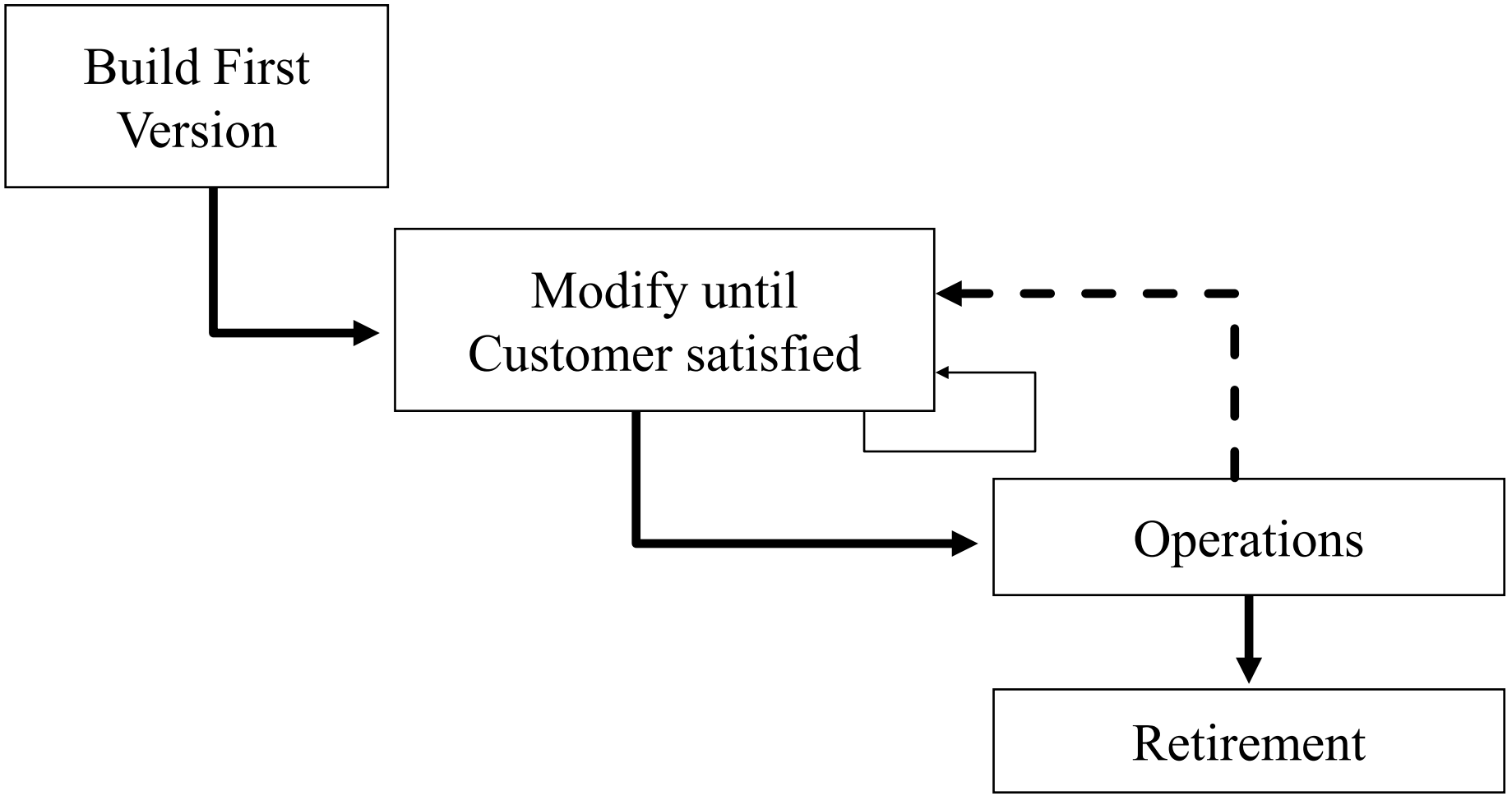



Requirements Analysis

Planning & Design

Implementation

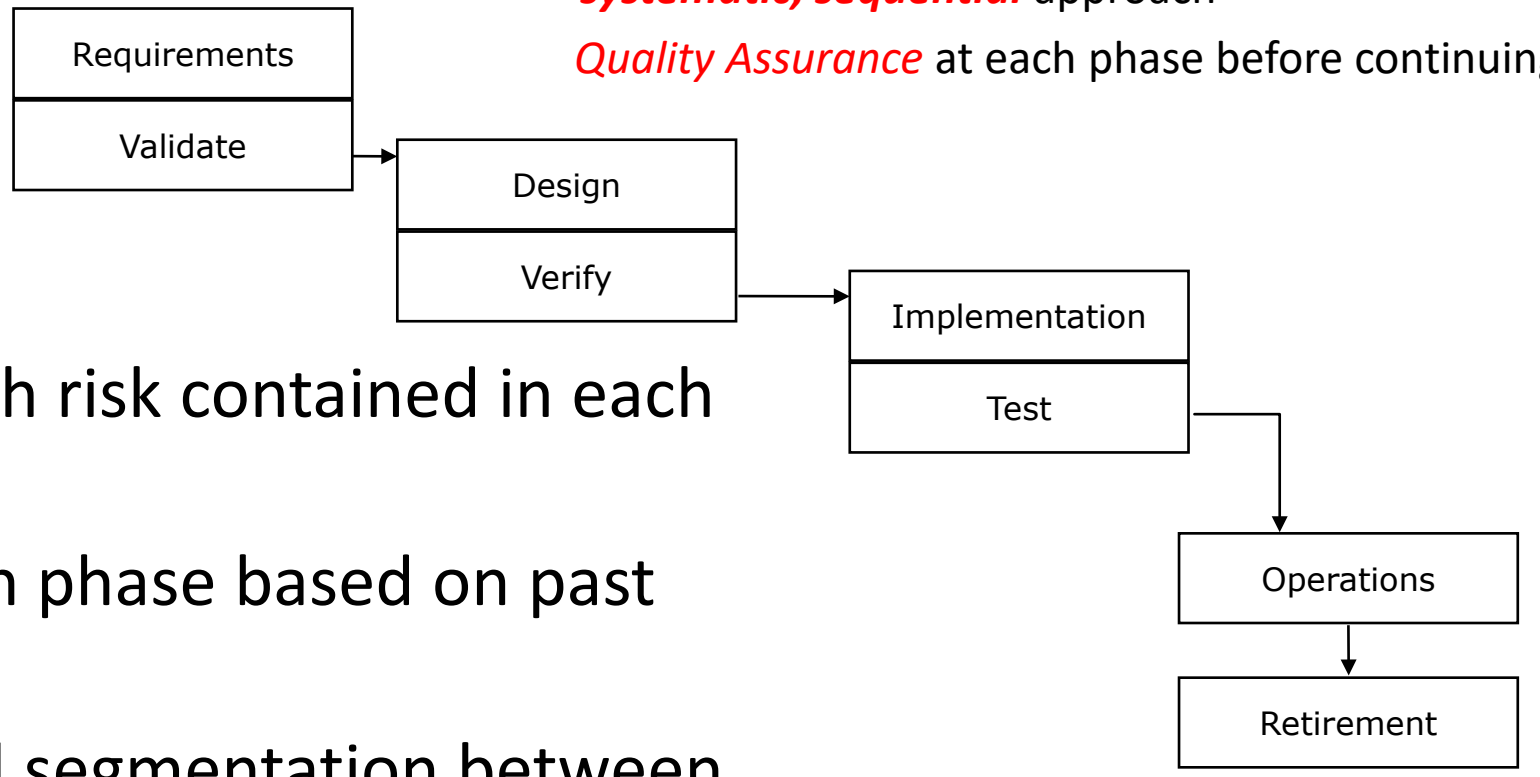
# Software Process: Code + Fix



# Waterfall Process Improves on Code + Fix (~1970s)

***systematic, sequential*** approach

*Quality Assurance* at each phase before continuing




- 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* and produced *Wasted* Work Product

Since formal quality assurance happens at each phase, it's necessary to produce extremely detailed documentation (requirements, design, testing)

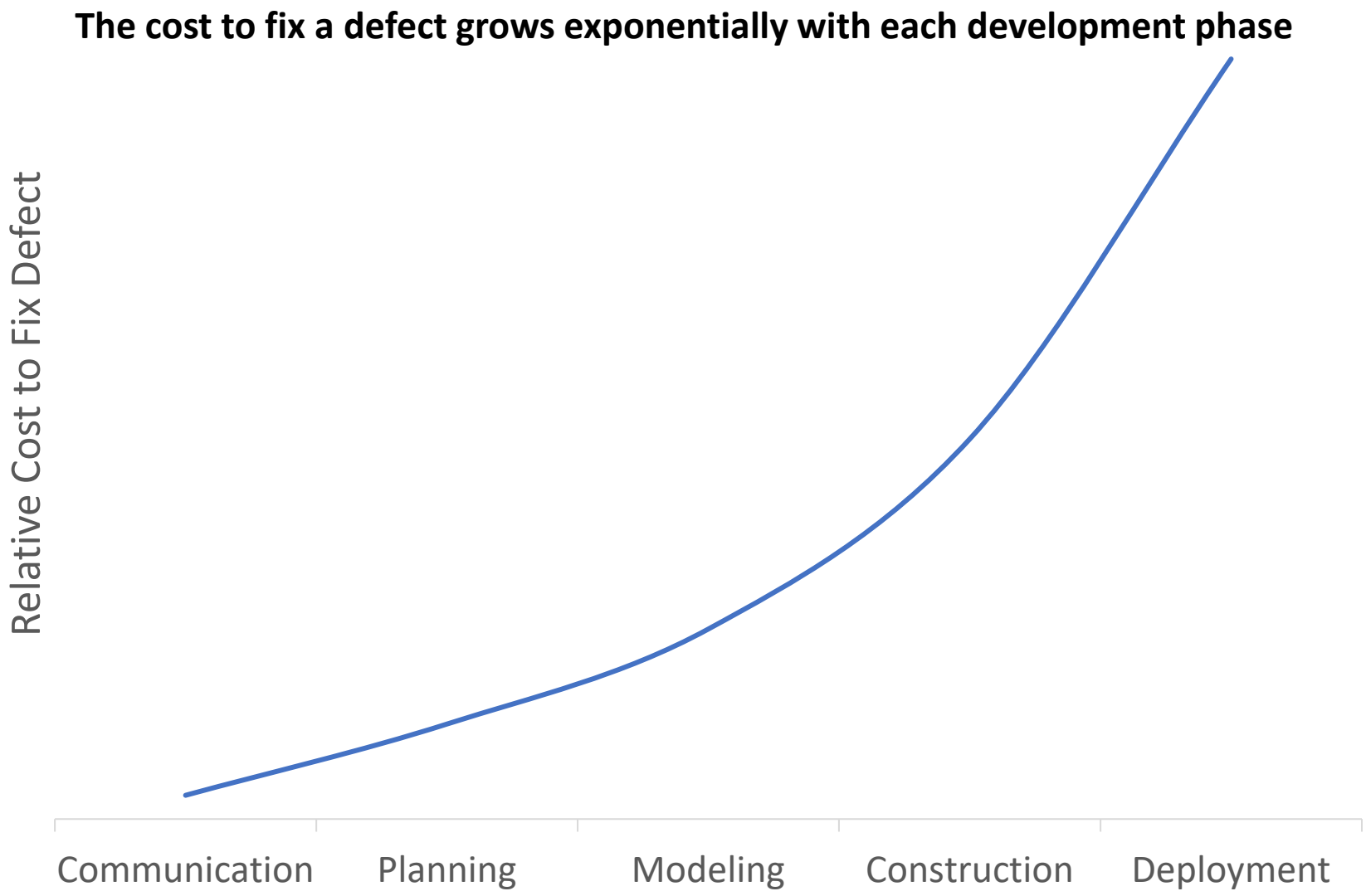
But

Documentation produced per requirements was never read, Elaborate architectural designs never used

# 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?

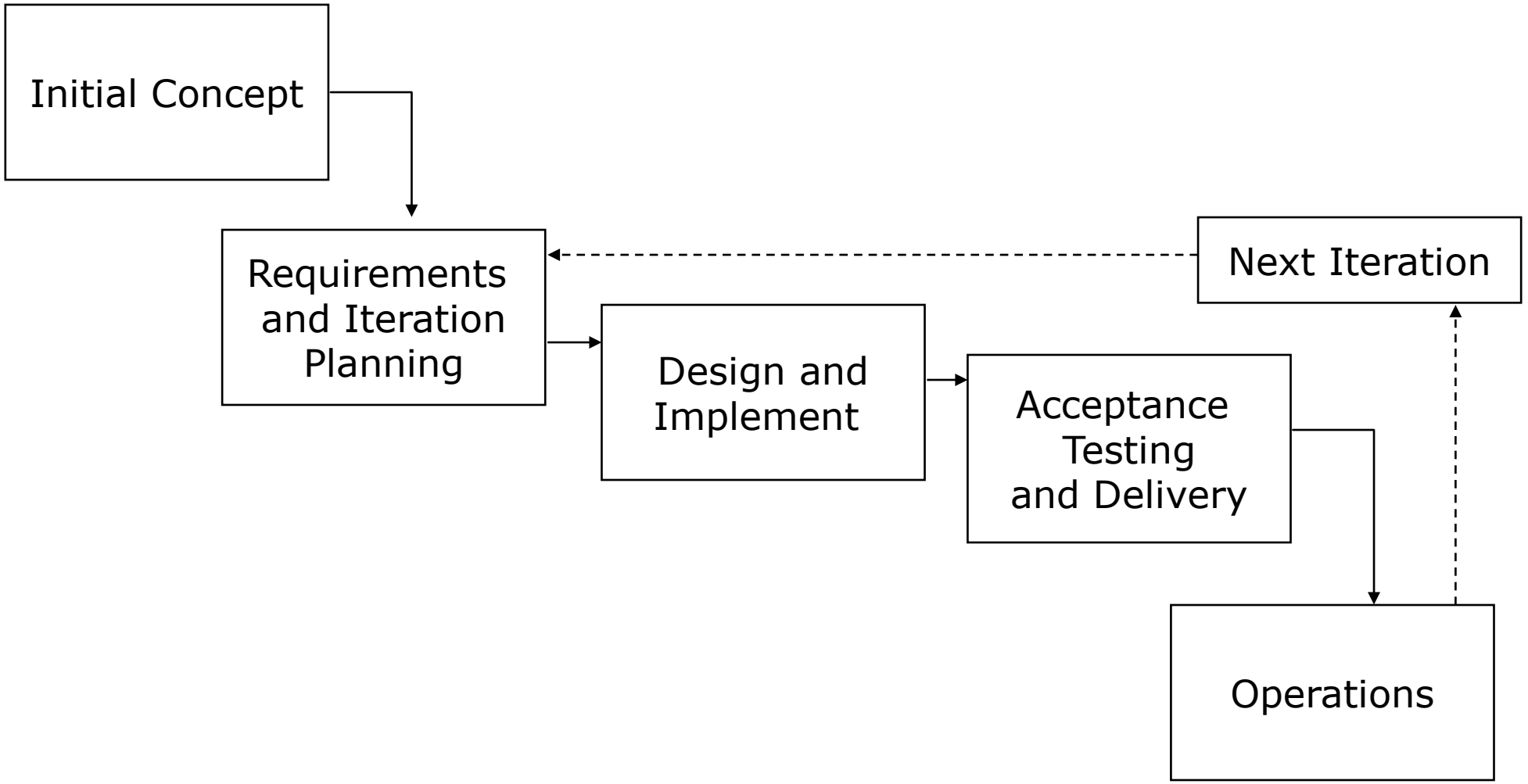
# Waterfall Model: Risk Assumptions



# Waterfall Model Reduces Risk by Preventing Change

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

# Iterative Process (~1980s) are Waterfall Variations



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

- The Waterfall philosophy:
  - "The project is too large and complex, and it will take months (or years!) to plan, so once we come up with the plan, that plan can not change"
  - Reduce risk by proceeding in stages
- The Agile philosophy:
  - The project is too large and complex, it is unlikely that we will know exactly what we need right now, and to some extent, we are inventing something new. We think that as we make it, we will figure it out as we go”
  - Reduce risk by limiting time on any one stage; then reassess. (“time-boxing”)

# Agile Manifesto

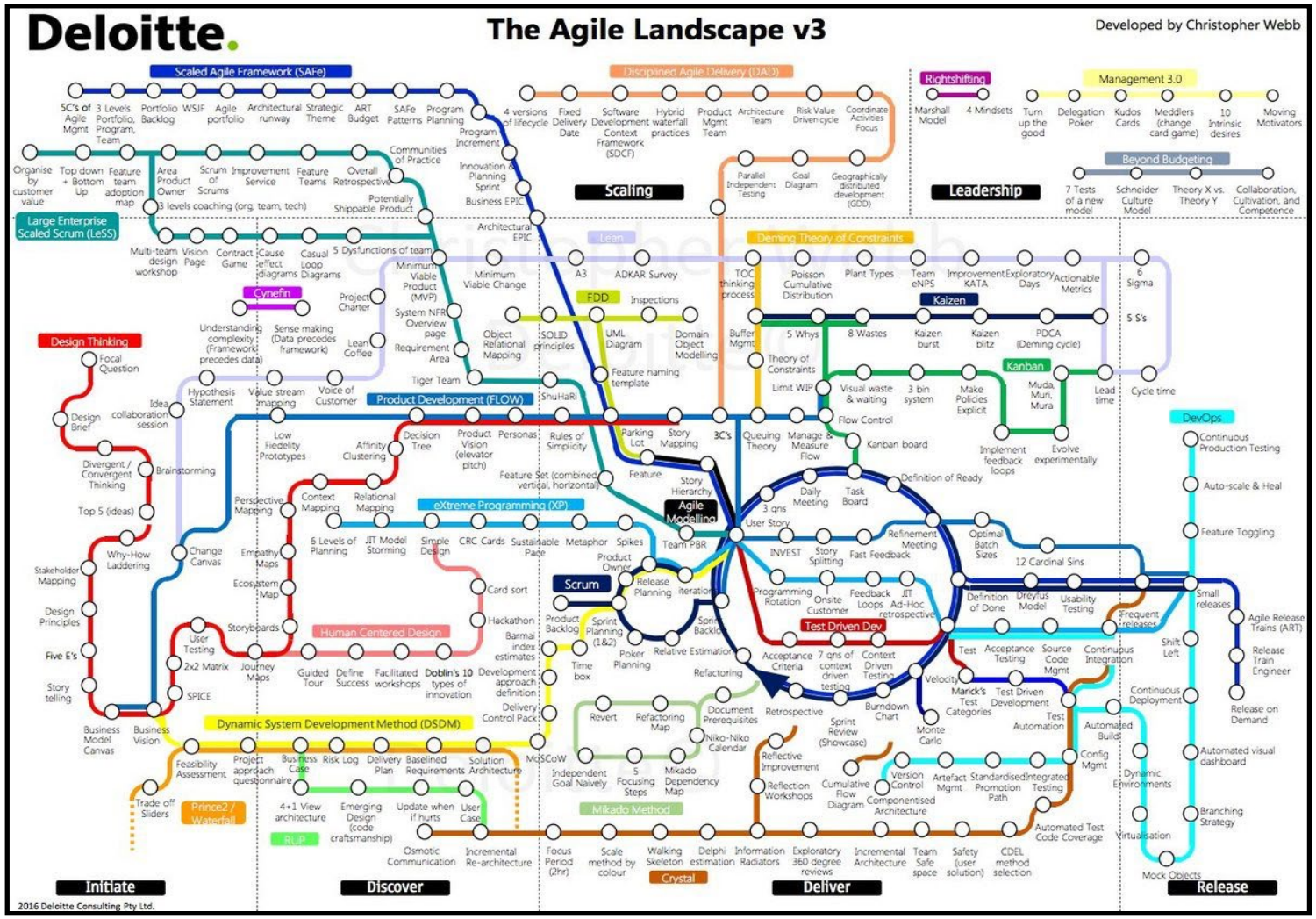
We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

| | |
|---|---|
| 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

# Warning: Agile can be a buzzword



# Agile Values 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

Or only as much documentation as you really need.

# 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 requires quality assurance processes

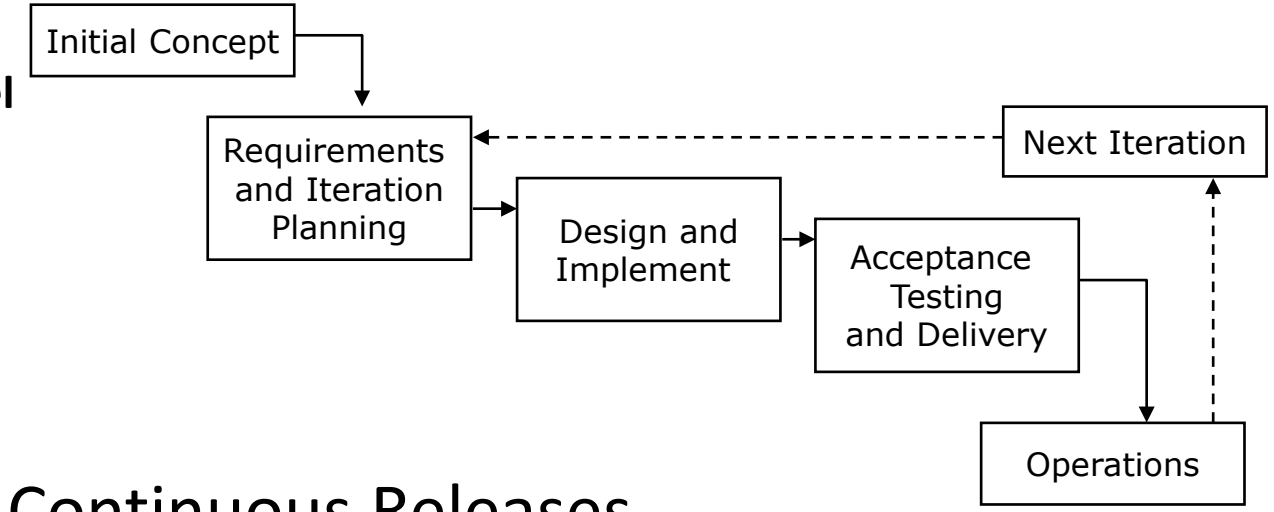
- Quality is everyone's responsibility
- Multiple processes work together to ensure quality:
  - unit testing/TDD
  - mix of unit tests & integration tests (we'll see more of this)
  - code review
  - continuous integration
  - continuous deployment (A/B, canaries, etc.)
  - quality includes non-functional requirements (resource consumption, response time) or generally speaking extensibility, maintainability, etc.

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

# Agile Processes are Iterative

**Agile Process Model**

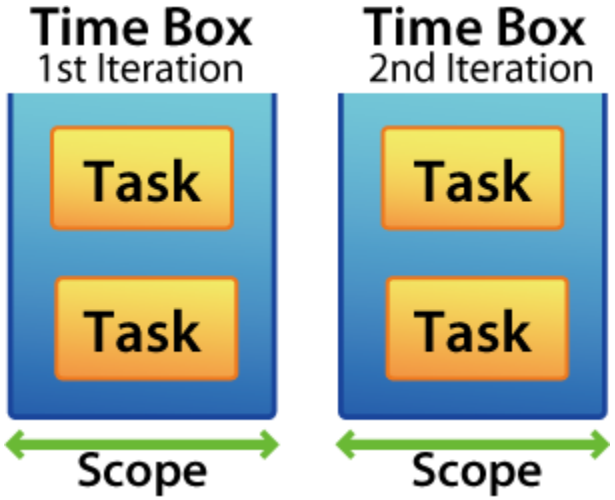

**Iterative Waterfall Model**



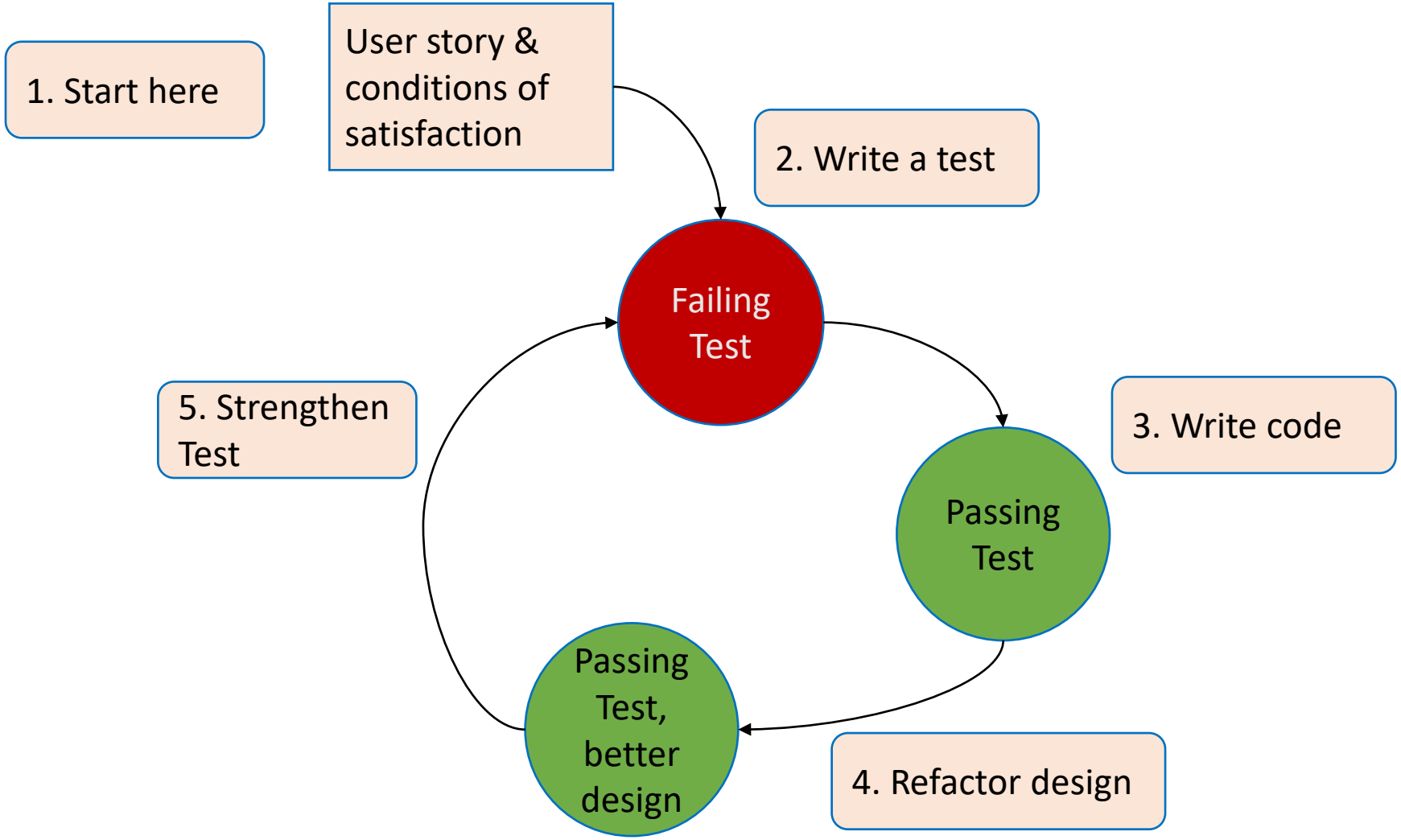

Key Idea: Small Continuous Releases

# 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 the next module

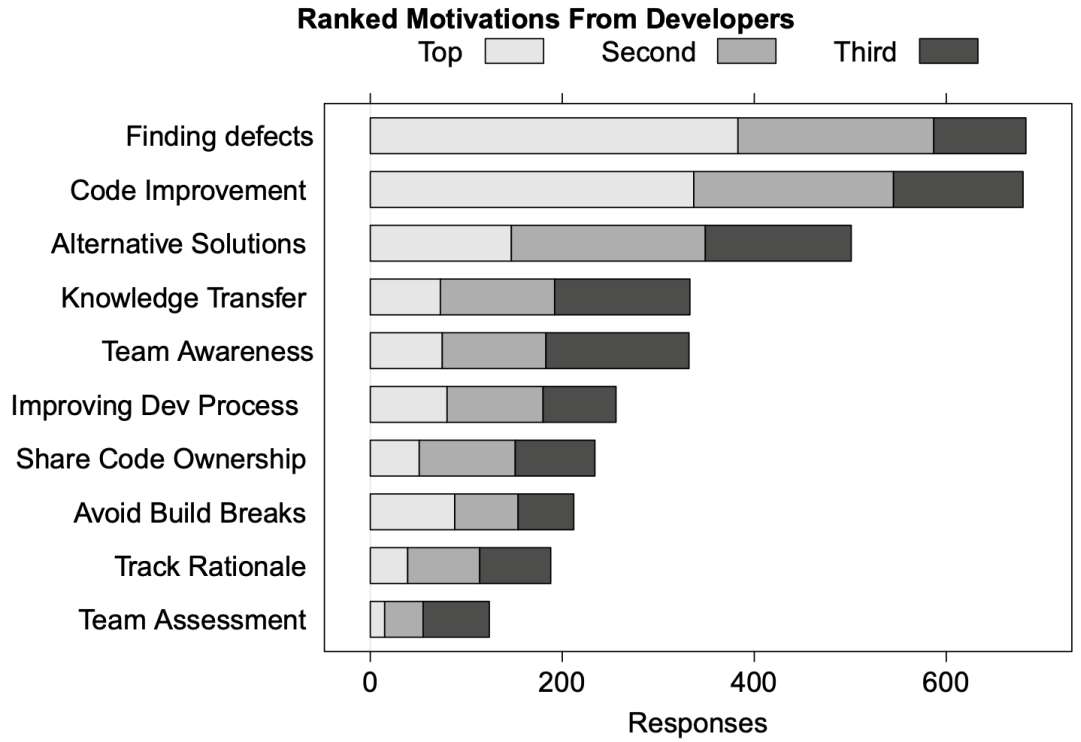


# Agile Practice: Test Driven Development (TDD)



# Code Review is Agile Practice that's used to Refactor

- 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")



# Agility and You

- In your project, you can display agility in some of the following ways:
  - Renegotiate specs
  - Reorder priorities
  - Alter implementation strategy
  - Improve team communication patterns
- If you are agile, you can adjust these things to deliver your product on time and get a good grade ☺

# 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