Objectives

- Software Development
  - Agile
- Testing

1

Review

1. What does the compiler do?
   - How is compiling different from interpreting?
2. What are examples of compiler optimizations?
3. True or False: If the compiler is applying lots of code optimizations to my code, that means I wrote my code poorly.
4. What is the software testing process, formally?
   - What are the components?

2

Different Perspectives on the Program

To the Compiler

- This is my one shot to validate the program and optimize it!

To You/Developer

- The long view: I am compiling the program now, but I could change the program later.
  - It should be easy to update the program; otherwise, I could introduce bugs.

SOFTWARE DEVELOPMENT

Programming is not Software Engineering

"It's Programming if 'clever' is a compliment.
It's Software Engineering if 'clever' is an accusation."
-- Titus Winters, Google Software Engineer

- This course is software development…
  - We're moving from programming towards software engineering
  - One metric: how long you think before you code

https://twitter.com/tituswinters/status/1143595692113481728

5

No Silver Bullet: Essence and Accidents of Software Engineering

"Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.

"The familiar software project, at least as seen by the nontechnical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet—something to make software costs drop as rapidly as computer hardware costs do.

"But, as we look to the horizon of a decade hence, we see no silver bullet. **There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity**. In this article, I shall try to show why, by examining both the nature of the software problem and the properties of the bullets proposed."

by Frederick P. Brooks, Jr., 1986

6

Software Engineering

- Software Engineering is a relatively new field
  - Still learning best practices

Takeaway: We will employ lots of techniques that help make software development process more efficient without sacrificing software quality

8

How to Implement an Effective Solution

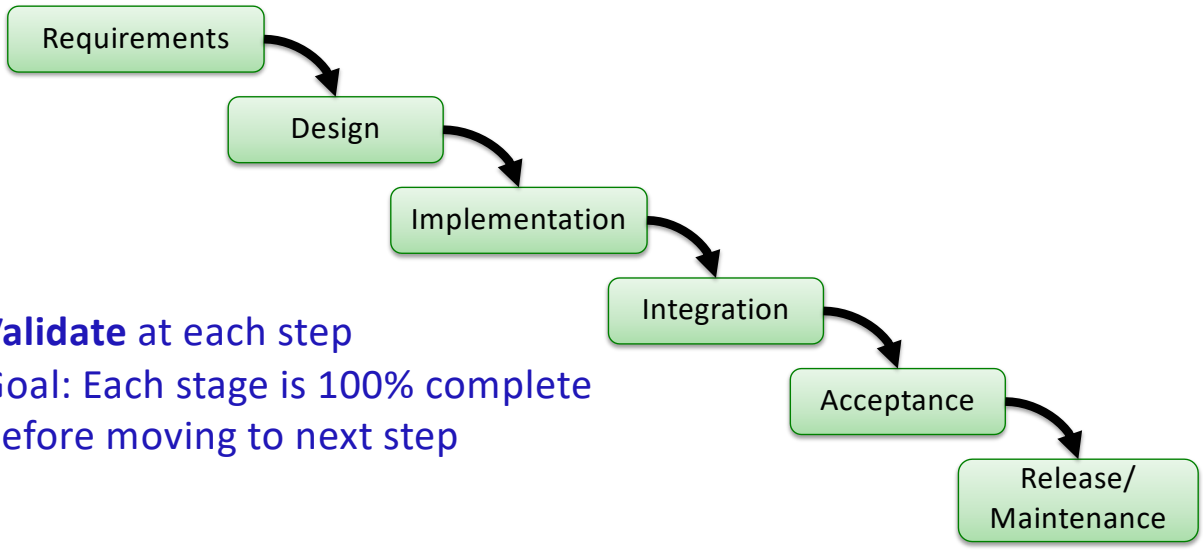
1. Understand the problem
2. Understand external constraints
3. Design an effective solution to the problem
4. While designing the solution, design some ***tests*** to verify that the problem is solved (and remains solved)
5. Code the effective solution to the problem
6. Teach other team members about your solution to the problem

9

How to Implement an Effective Solution

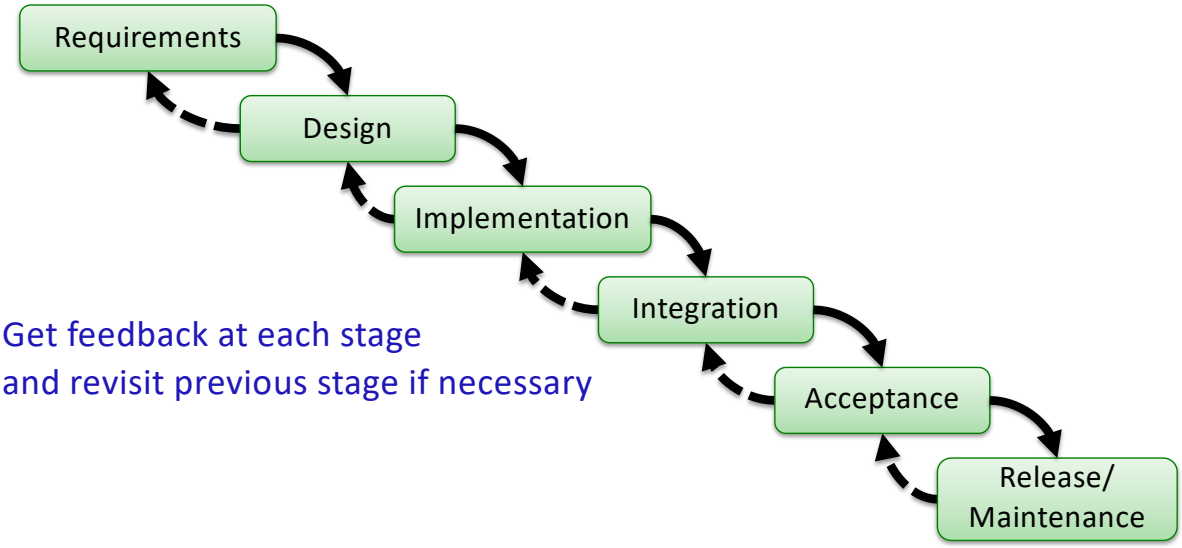
1. Understand the problem (interact with ***people***)
2. Understand external constraints (interact with ***people***)
3. Design an effective solution to the problem
4. While designing the solution, design some ***tests*** to verify that the problem is solved (and remains solved)
5. Code the effective solution to the problem
6. Teach other team members about your solution to the problem (interact with ***people***)

Traditional Software Engineering Process: Waterfall Model



Validate at each step
Goal: Each stage is 100% complete before moving to next step

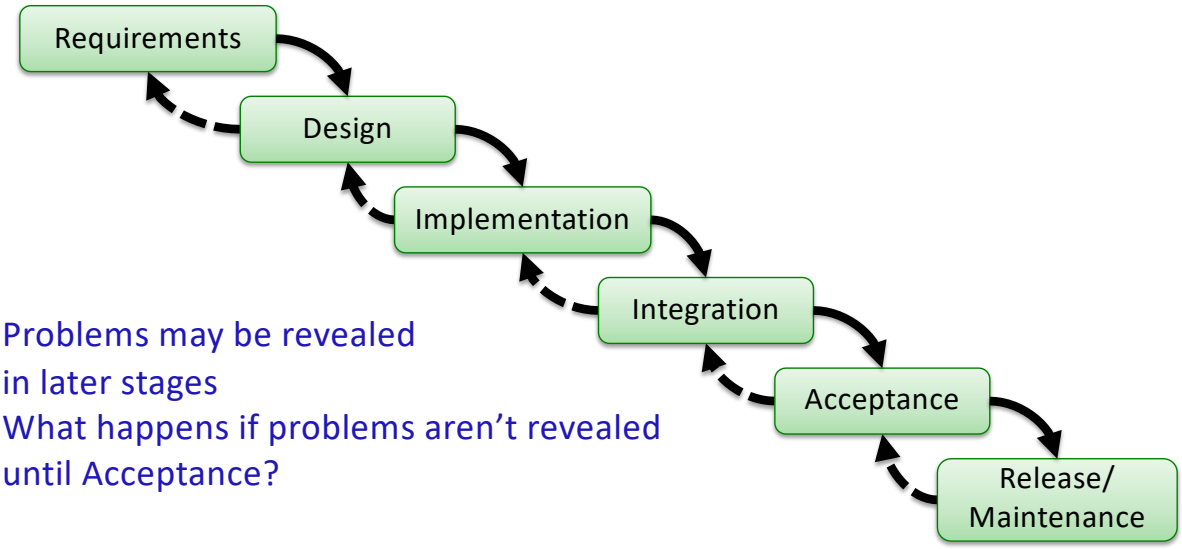
Feedback in Waterfall Model



- Get feedback at each stage and revisit previous stage if necessary

12

Feedback in Waterfall Model



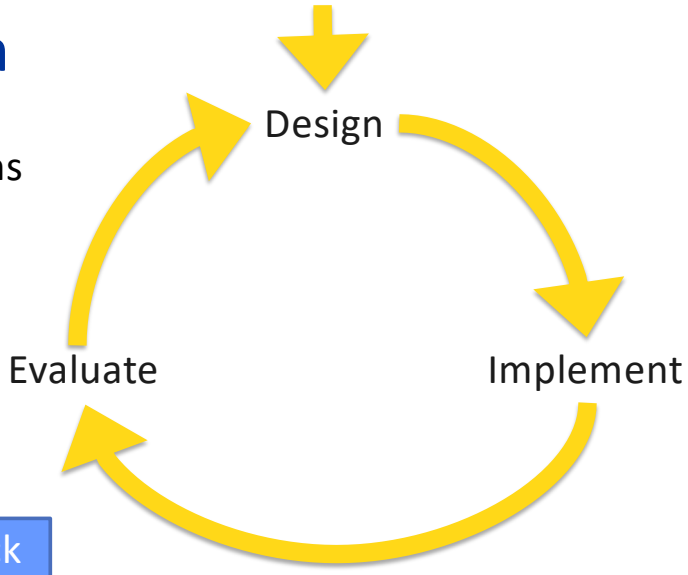
- Problems may be revealed in later stages
- What happens if problems aren't revealed until Acceptance?

13

Iterative Design

Various implementations

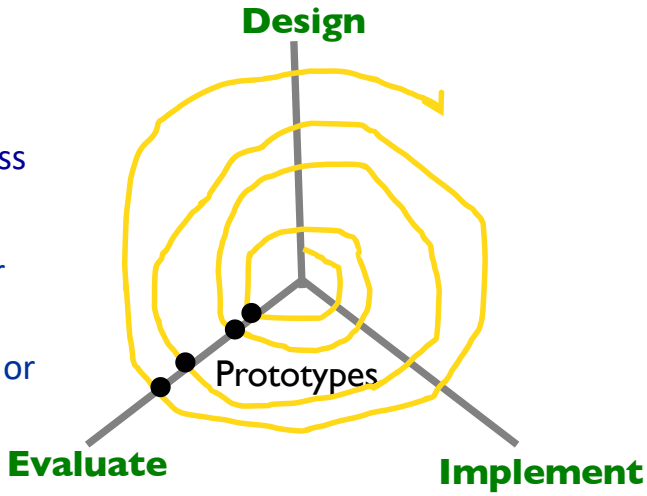
Get feedback/requirements from users/clients



Goals: Frequent feedback
→Identify problems early
→Higher quality product

Spiral Model

- Idea: smaller prototypes to test/fix/throw away
  - Finding problems early costs less
- In general…
  - Break functionality into smaller pieces
  - Implement most depended-on or highest-priority features first



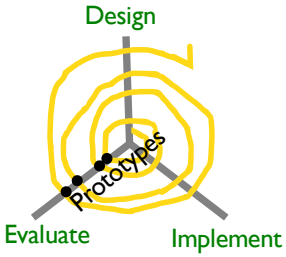
Radial dimension: cost

[Boehm 86]

Prototypes, In Brief

- Sample of application
  - Often: Demonstrate one part/purpose
    - Focus on one thing, not the whole thing
- Purpose/Dimensions
  - Functionality
  - Interaction
  - Implementation

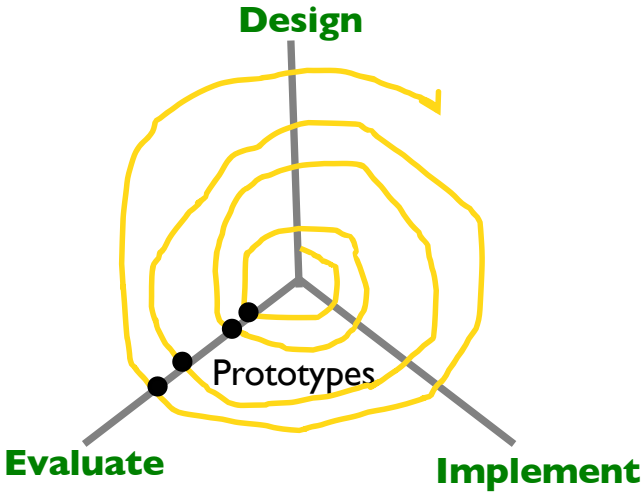
Spiral Model/Iterative Design Model Benefits



- Builds in getting feedback from client
  - Demo prototypes or working versions of [parts of] application
  - Clients' requirements may change
  - Clients' requirements may be ambiguous or were misinterpreted
- Makes project development more ***agile***
  - Goal: find problems early
  - Easier to throw away cheaper early prototypes
  - Adjust/adapt to changes

Spiral Model: Breaking Down Further

- Project's development process: Spiral Model
- What does this look like day to day?
  - Agile development is a common implementation



Radial dimension: cost

[Boehm 86]

Agile Development

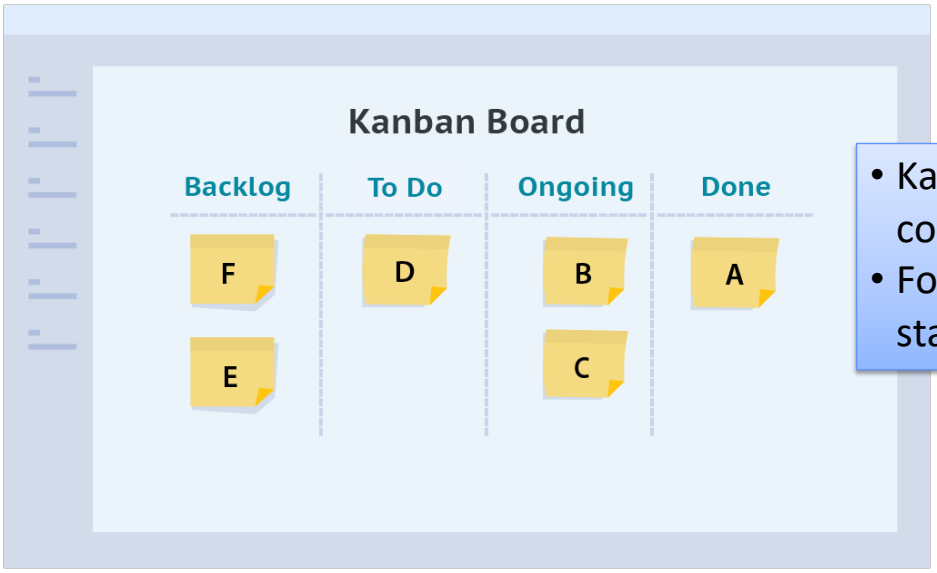
- Iterative approach to project management and software development
  - Work in small, launchable increments
  - Frequent review of requirements, plans, results
- Goals:
  - Respond to change quickly
  - Deliver application faster
  - Fewer conflicts about requirements
- Lots of variations – often company- or team-specific

Agile Development Framework: Scrum

- Product owner creates prioritized wish list: *a product backlog*
- Team works in a *sprint*, usually 2-4 weeks
  - During planning, team picks a subset of wish list, *a sprint backlog*, and decides how to implement those pieces
  - Daily Scrum: team meets daily to assess its progress
    - ScrumMaster keeps the team focused on its goal
  - At end of sprint, work should be potentially shippable:
    - ready to hand to a customer, put on a store shelf, or show to a stakeholder
  - The sprint ends with a sprint review and retrospective
- Repeat sprint

https://www.scrumalliance.org/why-scrum

Tools to Help: Kanban Board

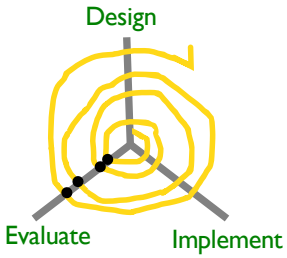


- Kanban is continuous, fluid.
- Focus on short start to finish time

 https://www.digite.com/kanban/what-is-kanban/

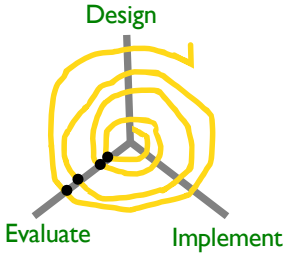
Really Zoomed In:
Iterative Development Steps

1. Design a {method, class, package}
2. Implement the {method, class, package}
3. Test the {method, class, package}
4. Fix the {method, class, package}
5. Deploy the {method, class, package}
6. Get feedback
   - Probably will require modifications to design
   - May even need to rollback a previous version
7. Repeat, building up



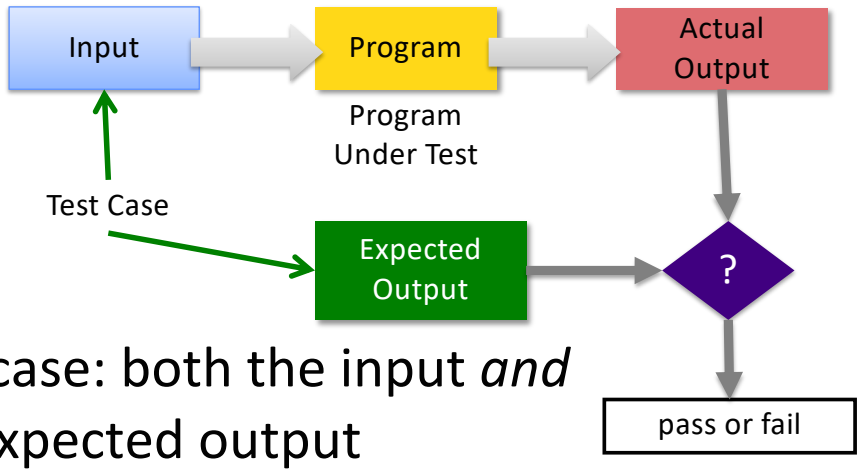
Really Zoomed In:
Iterative Development Steps

1. Design a {method, class, package}
2. Implement the {method, class, package}
3. ***Test*** the {method, class, package}
4. Fix the {method, class, package}
5. Deploy the {method, class, package}
6. Get feedback
   - Probably will require modifications to design
   - May even need to rollback a previous version
7. Repeat, building up



SOFTWARE TESTING PROCESS

Review: Software Testing Process



- Test case: both the input *and* the expected output
- Test Suite: set of test cases

Type 1 Bugs: Compile-Time

- Syntax errors
  - Missing semicolon, parentheses
- Compiler notifies of error
- Cheap, easy to fix

Type 2 Bugs: Run-Time

- Usually logic errors
- Expensive to locate, fix

Aside: Objections to "Bug" Terminology

- "Bug"
  - Sounds like it's just an annoyance
    - Can simply swat away
  - Minimizes potential problems
  - Hides programmer's responsibility
- Alternative terms
  - **Defect**
  - **Fault**

Tenor of Conversation

How do we detect bugs and fix them before the user sees them?

- NOT: how do we *never* write bugs?
  - We're human!
  - I mean, don't *try* to write bugs/be sloppy…
  - There's a balance.

Discussion: Your Testing Process

- How do you test?
- Categorize what you test/look for
- Are you a good tester? Why or why not?
  - What do you do well?
  - What do you need to get better at?

30

Common Bad Development Approaches

- Run the code. Did it do what you expect? <shrug/>
- Identify bug. Fix the bug on the test case that revealed the error. Don't test the other cases.
  - Similar: made a change to code (famous last words: "it shouldn't affect anything") and don't retest
- Tests don't help you identify the problem
  - A good set of tests will help you narrow the scope of the problem
- Random (only) testing

31

Microsoft  Testing

- Beyond their internal testing …
  - 5 million people beta tested
  - 60+ years of performance testing
  - 1 Billion+ Office 2007 sessions
- Still, users found correctness, stability, robustness, and security bugs

OSS Fuzz Project

- Continuous Fuzzing for Open Source Software
  - Fuzzing is a testing technique
- "Google has found thousands of security vulnerabilities and stability bugs by deploying guided in-process fuzzing of Chrome components"
- Also found 40K+ bugs in 600+ projects

https://github.com/google/oss-fuzz

Conclusion: Software Testing is Hard!

- Need to use a lot of different approaches
  - Different approaches catch different defects

Types of Testing

(Non-Exhaustive)

- Black-box testing
- White-box testing
- Non-functional testing
- Acceptance testing

Ideas about or definitions of any of these?
What is the approach? Or, what problems are they trying to reveal?

Types of Testing

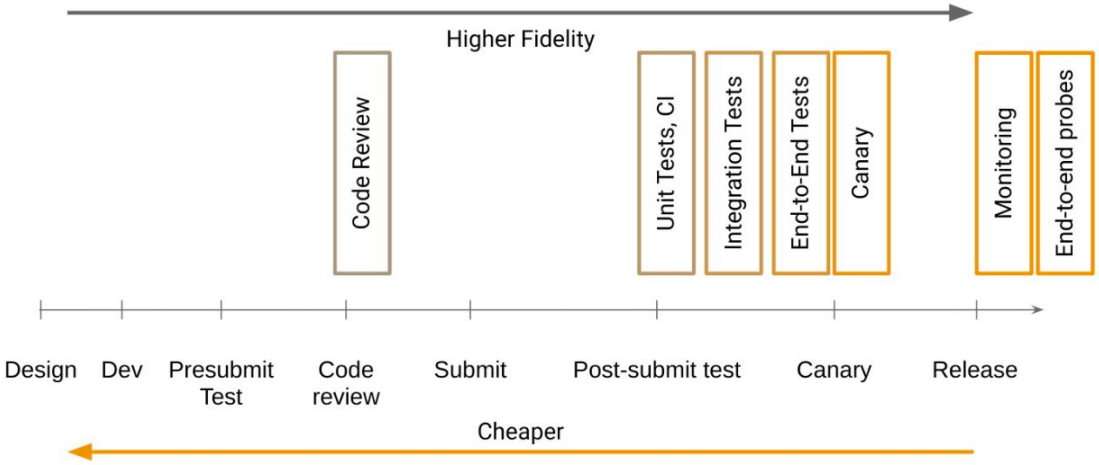
(Non-Exhaustive)

- Black-box testing
  - Test *functionality* (e.g., the calculator)
  - No knowledge of the code
  - Examples of testing: boundary values
- White-box testing
  - Have access to code
  - **Goal**: execute *all* code
- Non-functional testing
  - Performance testing
  - Usability testing (HCI)
  - Security testing
  - Internationalization, localization
- Acceptance testing
  - Customer tests to decide if they accept the product

Levels of Testing

- Unit
  - Tests minimal software component, in isolation
  - For us, Class-level testing
  - Web: Web pages (Http Request)
- Integration
  - Tests interfaces & interaction of classes
- System
  - Tests that completely integrated system meets requirements
- System Integration
  - Test system works with other systems, e.g., third-party systems

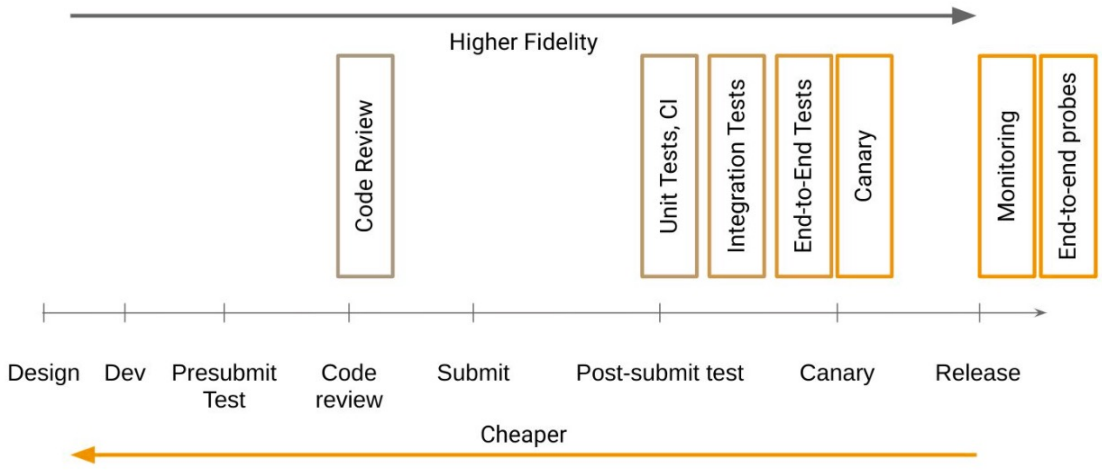
Software Development Process



A Bad Role Model

 http://imgur.com/HBSbn

Software Development Process



Shift Left's Goal: Catch defects earlier

Software Testing Issues

- How should you test? How often?
  - Code may change frequently
  - Code may depend on others' code
  - A lot of code to validate
- How do you know that an output is correct?
  - Complex output
  - Human judgment?
- What caused a code failure?

➥ Need a *systematic*, *automated*, *repeatable* approach

Some Approaches to Testing Methods

- Typical case
  - Test typical values of input/parameters
- Boundary conditions
  - Test at boundaries of input/parameters
  - Many faults live "in corners"
- Parameter validation
  - Verify that parameter and object bounds are documented and checked
  - Example: pre-condition that parameter isn't null

➥ All black-box testing approaches

Looking Ahead

- Read slides about testing, JUnit before Monday's class
  - Canvas quiz
- Goal: Hands-on lab in class on Monday
  - Prep: Clone the repository