Putting Testing First

Idaho State University | Computer Science

Isaac Griffith

CS 4422 and CS 5599
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Outcomes

At the end of Today's Lecture you will be able to:

- Understand the basic concepts of TDD
- Understand the basic concepts of CI
- Understand how to marry TDD and CI

ROAR

Inspiration

"If you don't like unit testing your product, most likely your customers won't like to test it either." – Anonymous

WHAT IS TDD?

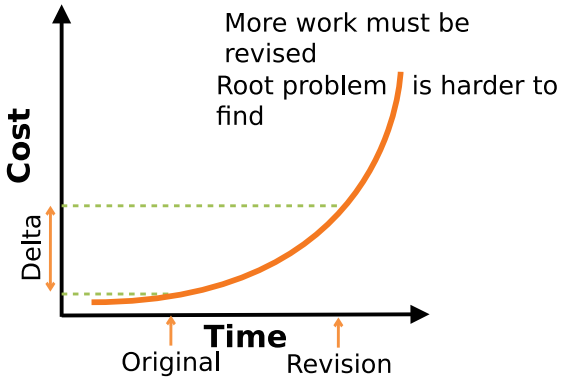
what is

UNIT TESTING?

ROAR

The Increased Emphasis on Testing

- Philosophy of **traditional** software development methods
  - **Upfront** analysis
  - Extensive **modeling**
  - Reveal **problems** as early as possible



Idaho State University
Computer Science

Traditional Assumptions

1. Modeling and analysis can identify potential problems early in development
2. Savings implied by the cost-of-change curve justify the cost of modeling and analysis over the life of the project

- These are true if requirements are always complete and current
- But customers always change their minds!
  - Humans are naturally good at approximating
  - But pretty bad at perfecting
- These two assumptions have made software engineering frustrating and difficult for decades

Thus, Agile Methods…

ROAR

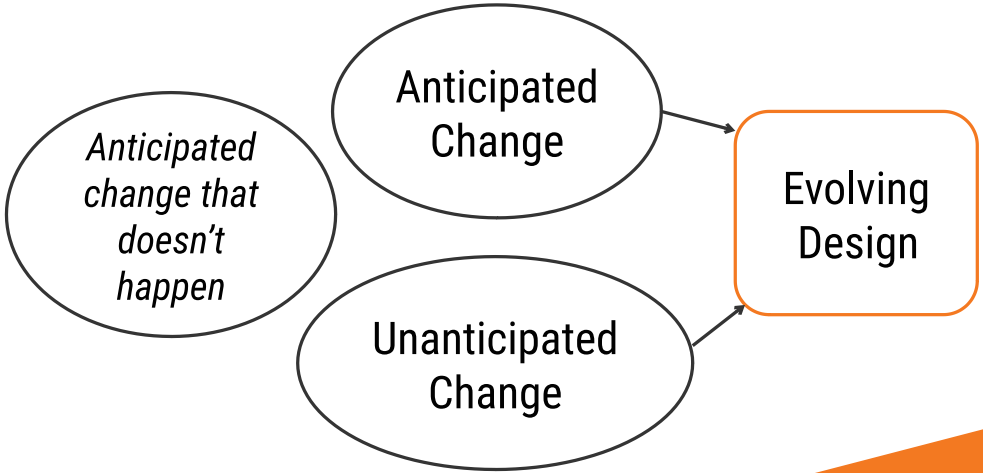
Why Be Agile?

- Agile methods start by recognizing that **neither assumption** is valid for many current software projects
  - Software engineers are **not good at developing requirements**
  - We do not anticipate many **changes**
  - Many of the changes we do anticipate are **not needed**
- Requirements (and other “non-executable artifacts”) tend to go **out of date** very quickly
  - We seldom take time to **update** them
  - Many current software projects **change continuously**
- Agile methods expect software to **start small and evolve** over time
  - Embraces **software evolution** instead of fighting it.

Idaho State University
Computer Science

Supporting Evolutionary Design

- Traditional design advice says to anticipate changes
- Designers often anticipate changes that don't happen



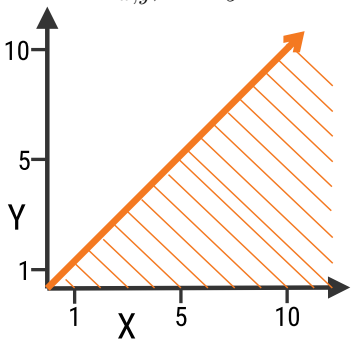
ROAR

The Test Harness as Guardian

What is Correctness?

Traditional Correctness
(Universal)

$$\forall_{x,y}, x \geq y$$



Agile Correctness
(Existential)

```
{ (1, 1) -> T
  (1, 0) -> T
  (0, 1) -> F
  (10, 5) -> T
  (10, 12) -> F }
```

Idaho State University
Computer Science

A Limited View of Correctness

- In **traditional** methods, we try to define **all correct behavior** completely, at the beginning
  - What is **correctness**?
  - Does "correctness" **mean anything** in large engineering products?
  - People are **VERY BAD** at completely defining correctness
- In **agile** methods, we redefine correctness to be **relative** to a specific set of tests
  - If the software behaves correctly **on the tests**, it is "correct"
  - Instead of **defining all** behaviors, we **demonstrate some** behaviors
  - **Mathematicians** may be disappointed at the lack of completeness

But software engineers ain't mathematicians!

ROAR

Idaho State University
Computer Science

In-Class Exercise

Group Discussion: **Limited Correctness**

- Do you understand the distinction?
- How does limited correctness relate to evolutionary design?

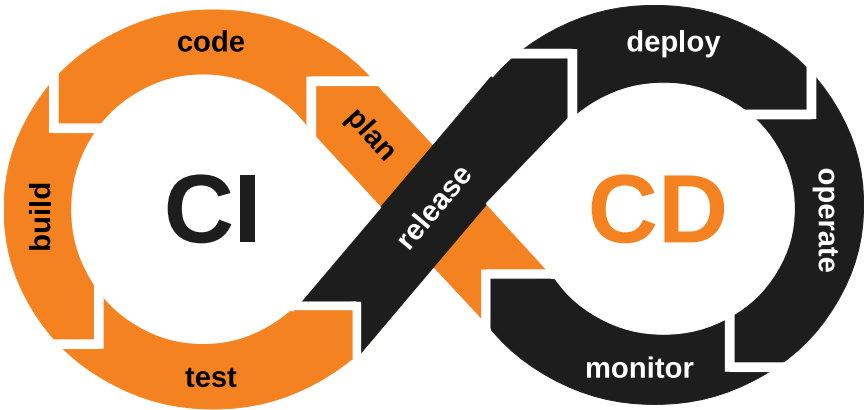
You have ten minutes

ROAR

Test Harnesses Verify Correctness

A **test harness** runs all automated tests and reports results to the developers

- Tests must be **automated**
  - Test automation is a **prerequisite** to test driven development
- Every test must include a **test oracle** that can evaluate whether that test executed correctly
- The tests replace the **requirements**
- Tests must be **high quality** and must **run quickly**
- We run tests **every time** we make a change to the software



Continuous Integration

- Agile methods work best when the current version of the software can be run against all tests at any time

A **continuous integration server** rebuilds the system, returns, and re-verifies tests whenever any update is checked into the repository

- Mistakes are caught earlier
- Other developers are aware of changes early
- The rebuild and re-verify must happen as soon as possible
  - Thus, tests need to execute quickly

A **continuous integration server** doesn't just run tests, it decides if a modified system is **still correct**

CI Reduces Risk

TDD encourages incremental integration of functionality

Non-integrated functionality is dangerous

Are there any questions?