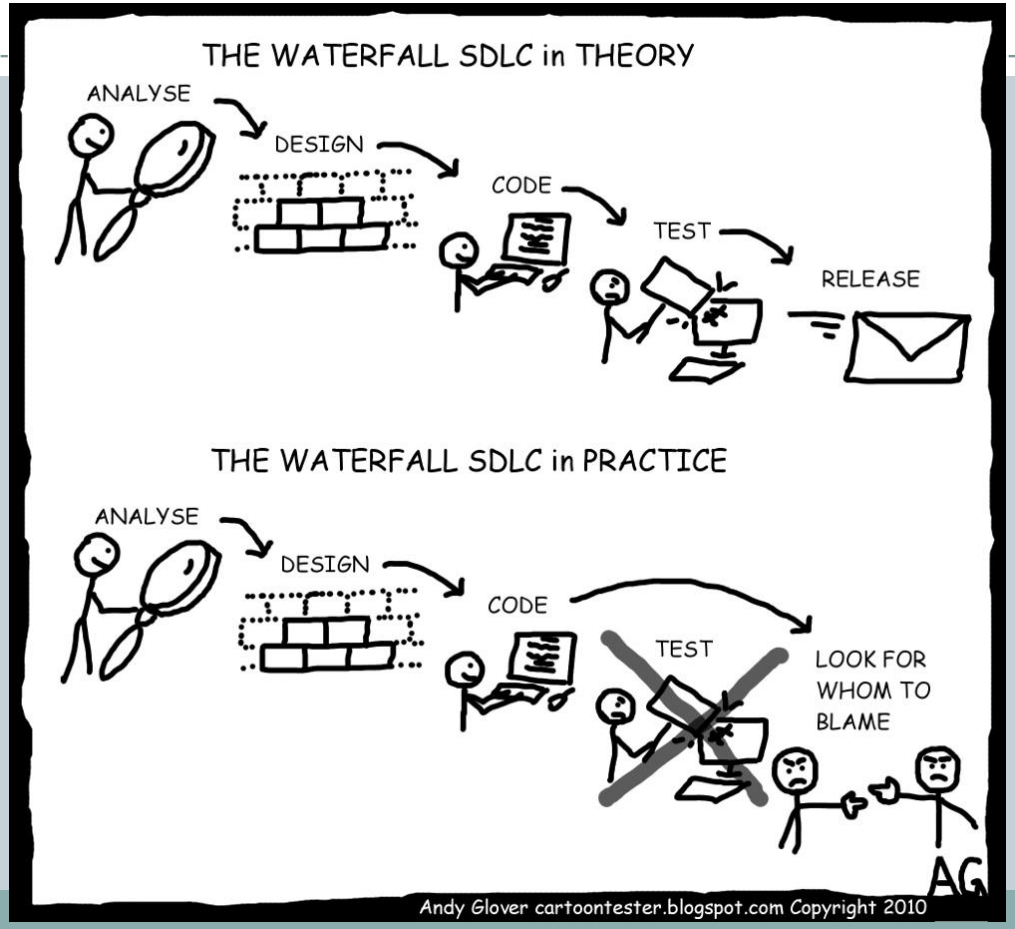
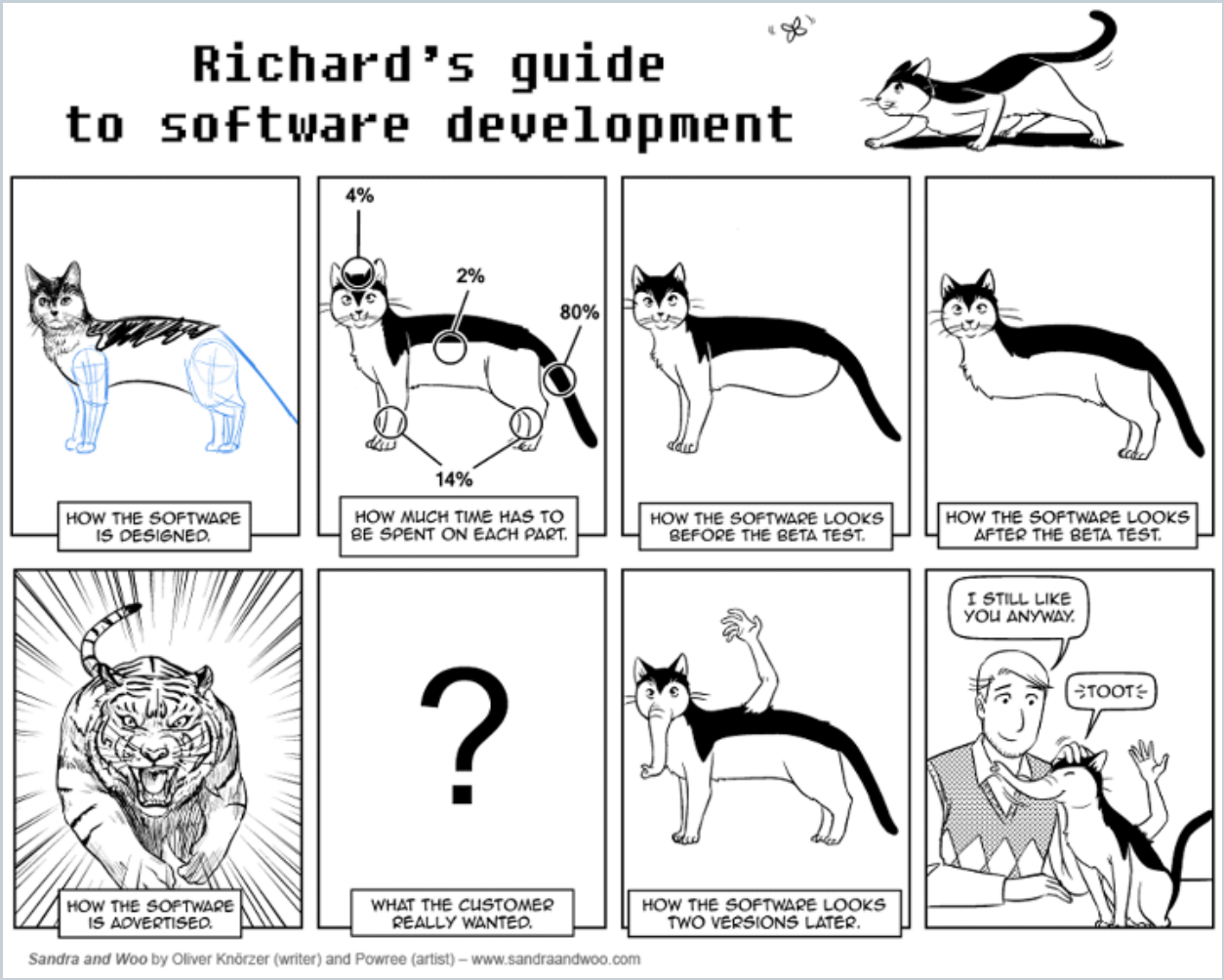
Test Driven Development (TDD)



Outline

- TDD Overview
- Test First vs. Test Last
- Summary



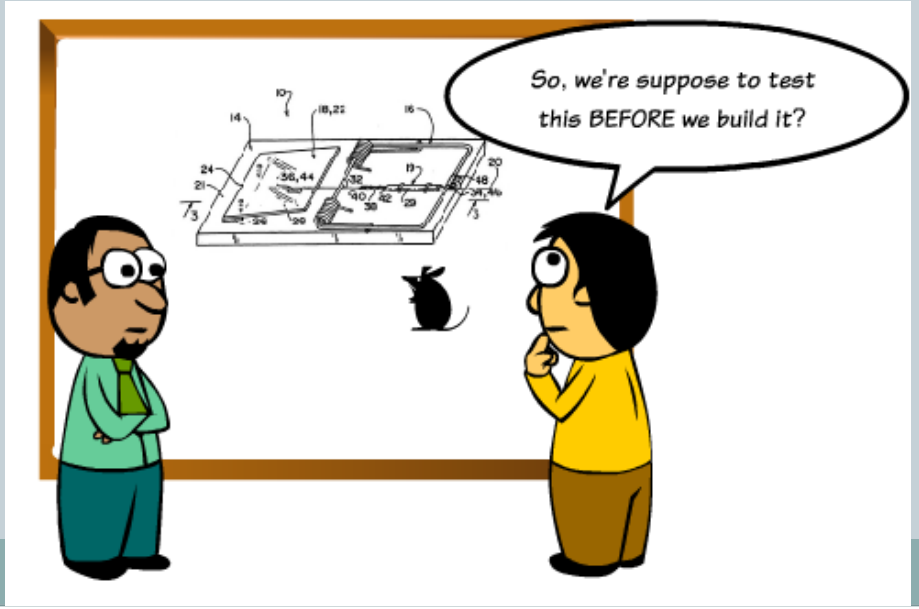
Quotes

Kent Beck said “Test-first code tends to be more cohesive and less coupled than code in which testing isn’t a part of the intimate coding cycle” [2]

“If you can’t write a test for what you are about to code, then you shouldn’t even be thinking about coding” [2]

TDD Overview (1 of 3)

- Made popular by Extreme Programming
- Method of developing software not just testing software
- Software is Developed in short iterations
- Unit Tests are developed FIRST before the code



TDD Overview (2 of 3)

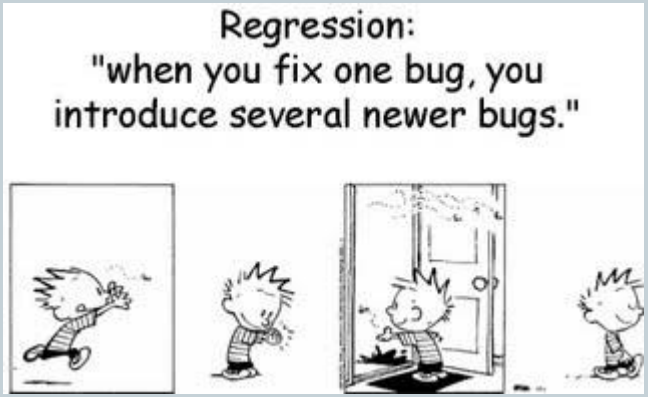
- How It Works –

1. Add a Test
   - Use Cases / User Stories are used to understand the requirement clearly
2. Run all tests and see the new one fail
   - Ensures test harness is working correctly
   - Ensures that test does not mistakenly pass
3. Write some code
   - Only code that is designed to pass the test
   - No additional functionality should be included because it will be untested [4]

TDD Overview (2 of 3)

4. Run the automated tests and see them succeed
   - If tests pass, programmer can be confident code meets all tested requirements
5. Refactor code
   - Cleanup the code
   - Rerun tests to ensure cleanup did not break anything

Repeat [4]



Why TDD?

- TDD can lead to more modularized, flexible, and extensible code
- Clean code
- Leads to better design
- Better code documentation
- More productive
- Good design

Unit Testing

- ..run completely by itself, without any human input. Unit testing is about automation.
- ...determine by itself whether the function it is testing has passed or failed, without a human interpreting the results.
- ...run in isolation, separate from any other test cases (even if they test the same functions). Each test case is an island

Goal -

- Isolate each part of the program and show that the individual parts are correct.

Benifits

- ..A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits. Unit tests find problems early in the development cycle

Limitations--

- -Testing cannot be expected to catch every error in the program

Mechanics of TDD

- Always start with a failing test
- Quickly write the simplest code needed to pass the test
- Remove duplication (AKA Refactor)
- Repeat as needed to meet requirements
- Test everything that could possibly break

What can be tested?

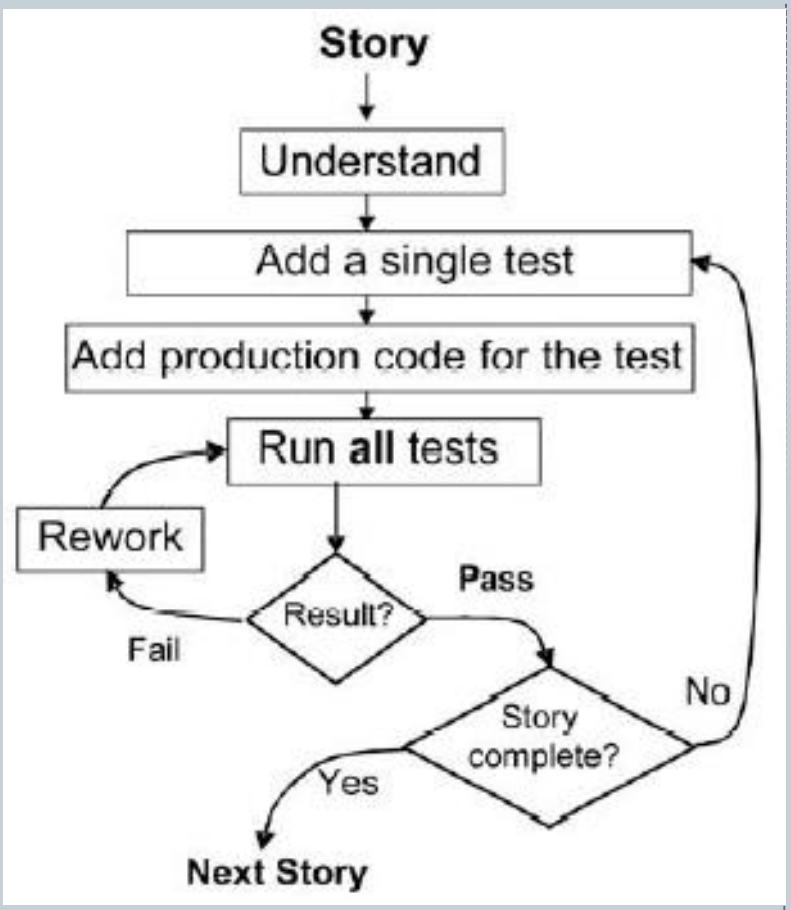
- Valid inputs
- Invalid inputs
- Errors, exceptions, and events
- Boundary conditions
- Everything that could possibly break!

Test First vs. Test Last

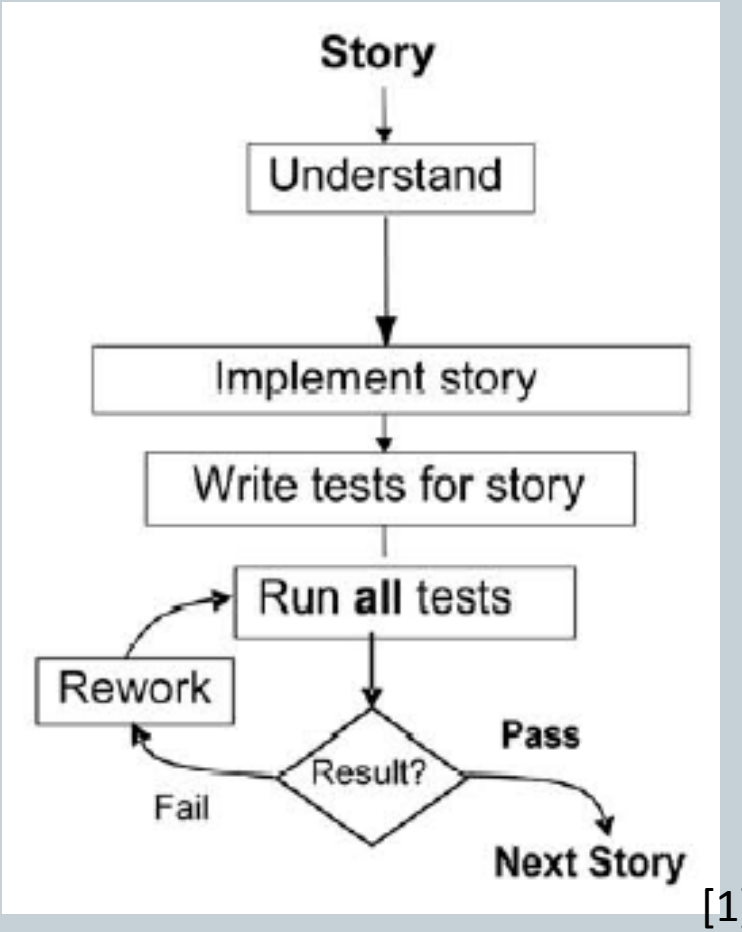
- Pick a piece of functionality
- Write a test that expresses a small task that fails
- Write production code until test passes
- Run all tests
- Rework code until all tests pass
- Repeat [1]

- Pick a piece of functionality
- Write production code that implements entire functionality
- Write tests to validate all functionality
- Run all tests
- Rework code until all tests pass [1]

Test First vs. Test Last



[1]



[1]

Got Bugs?

- Go after your most painful defect
- Write a test to expose it
- Write code needed to fix it
- Refactor
- Repeat
- Measure the results

TDD Benefits (1 of 3)

- Instant Feedback
  - Developer knows instantly if new code works and if it interferes with existing code [1]
- Better Development Practices
  - Encourages the programmers to decompose the problem into manageable, formalized programming tasks [1]
  - Provides context in which low-level design decisions are made [1]
  - By focusing on writing only the code necessary to pass tests, designs can be cleaner and clearer than is often achieved by other methods [4]

TDD Benefits (2 of 3)

- Quality Assurance
  - Having up-to-date tests in place ensures a certain level of quality [1]
  - Enables continuous regression testing [2]
  - TDD practices drive programmers to write code that is automatically testable [2]
  - Whenever a software defect is found, unit test cases are added to the test suite prior to fixing the code [2]

TDD Benefits (3 of 3)

- Lower Rework Effort
  - Since the scope of a single test is limited, when the test fails, rework is easier
  - Eliminating defects early in the process usually avoids lengthy and tedious debugging later in the project [4]
  - "Cost of Change" is that the longer a defect remains the more difficult and costly to remove [3]

TDD Limitations (1 of 2)

- Counterproductive and hard to learn [1]
- Difficult in Some Situations
  - GUIs, Relational Databases, Web Service
  - Requires mock objects
- TDD does not often include an upfront design [2]
  - Focus is on implementation and less on the logical structure

TDD Limitations (2 of 2)

- Difficult to write test cases for hard-to-test code
  - Requires a higher level of experience from programmers [2]
- TDD blurs distinct phases of software development
  - design, code and test [2]

Summary

- TDD Overview
- Test First vs. Test Last
- Summary