Test Driven Development

Objectives

- Explain why you might want to Test
- Install a test framework
- Red-Green-Refactor
- Write basic tests that check for equality
- Practice pairing techniques

You are about to take a test!

Why test our code?

Why test our code?

- Saves time
- Prevent regressions
- Can help you design better code
- Serve as executable documentation

Testing isn't something you just have to do, it is a mindset that helps you write more robust code.

Test Frameworks

Test Frameworks

- Mocha (often used with Chai)
- Jasmine
- QUnit
- many many others…

Parts of a test framework

- Assertions/matchers
- Test runner
- Utilities
  - Mocks
  - Fixtures

Test Frameworks

- Don’t get too caught up in the specifics!
- Deeply grok the principles
- It’s OK to lookup syntax at first! (even on the job)

Testing Main Idea

- Write code that tests code
- Determine what the expected result should be
- Run your code
- Check the actual result against the expected result

Test Driven Development

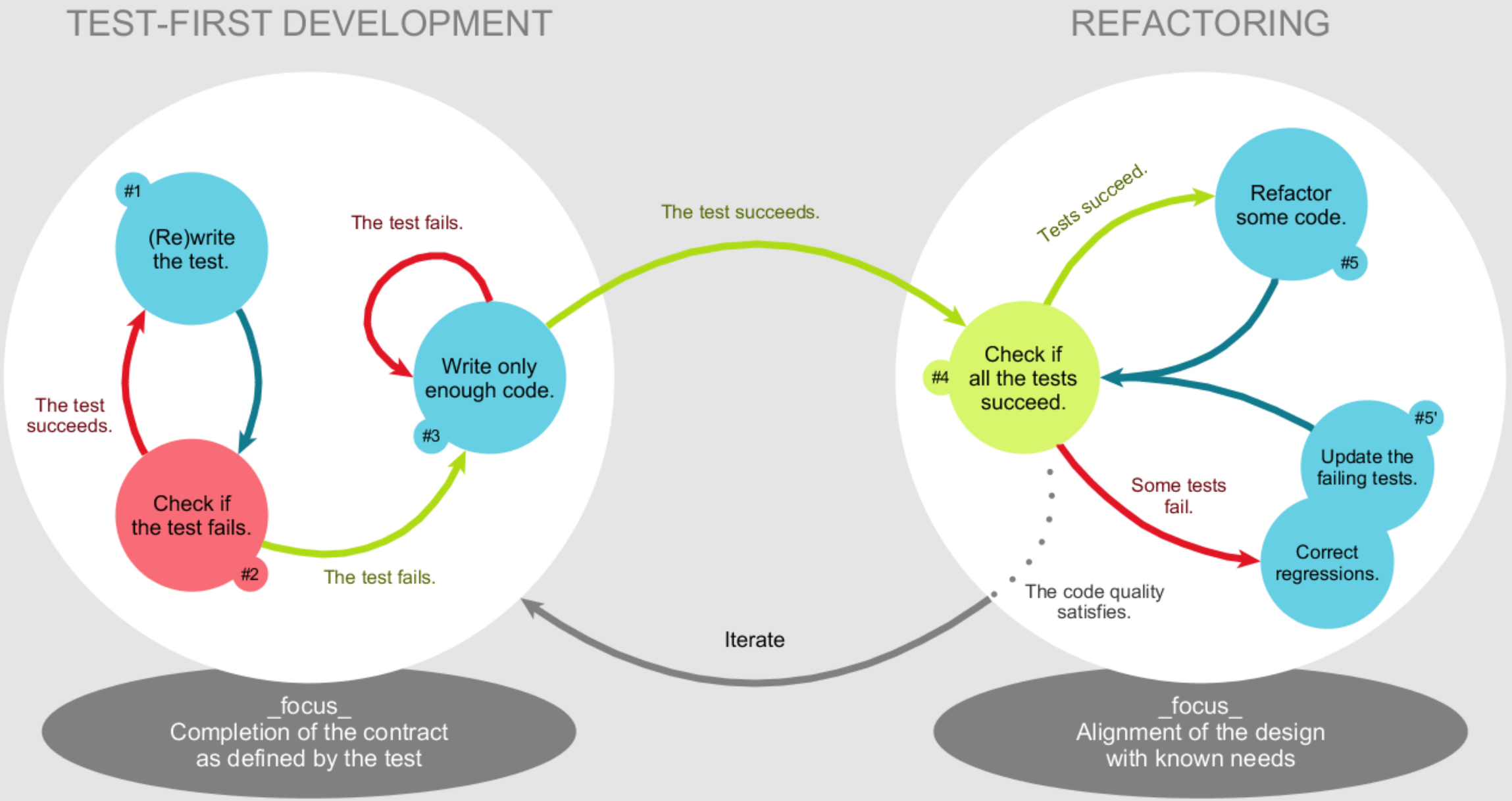
Origins

Test-driven development is related to the test-first programming concepts of extreme programming, which began in 1999

History of TDD

What is the basic flow of Test Driven Development (TDD)?

1. Write a test *first*
2. Run the test and watch it fail
3. Write the minimum amount of code required to make the test pass
4. Re-run the test at each change of code to see if the changes have made the test pass
5. Refactor if there are any opportunities to
6. Use tests to ensure that refactoring did not change behavior



First goal of TDD is to satisfy a requirement

```
A User:
    Has a first name, which:
      Is not a blank String
      Does not contain any special characters
```

Documentation

A secondary goal of TDD is to serve as documentation of the code's behavior in a human comprehendible (not just human readable) format.

The first statement, "A User", frames what the test group is talking about. Translating this into Mocha:

```
describe('User', function () {
});
```

At the next level, now that the test specifies what entity is being described, specific behaviors can be specified (one at a time) and tested as well:

```
describe('User', function () {
  describe('firstName', function () {
    it('is not a blank String', function () {
      // test code here
    });
    // additional tests for firstName
  });
});
```

How to setup a test suite?

Given the following rules about a leap year, what tests can we write to check if a given year is a leap year?

1. Every year whose number is perfectly divisible by four is a leap year.
2. Except for years which are both divisible by 100 and not divisible by 400.
3. 1600 and 2000 are leap years, but the century years 1700, 1800, and 1900 are not.

Give the following tax system, what kind(s) of tests can we write?

1. The first $10 is taxed at 10%
2. The second $10 is taxed at 7%
3. The third $10 is taxed at 5%
4. Everything after that is taxed at 3%

Bob

Ping Pong Pairing

In this technique, two partners, A and B work together writing tests and code in one of two ways:

Sequence of Events:
1. Person A writes a test
1. Person B writes the code to satisfy the test Person A authored
1. Person B writes a test
1. Person A writes the code to satisfy the test Person B authored

Sequence of Events:
1. Person A writes a test

Restricting techniques

Begin by asking the class to think of some restrictions you could impose when writing code.

To start off, begin with things like:
* no iteration/loops
* no conditional logic
* no recursion
* no use of mouse (only keyboard)
* swapping of roles for every written line of code (every time the enter key has been hit)
* using a timer and swap roles e.g every 3 minutes

TDD-Game

More exercises

- Editor OOP TDD with User Stories
- OOP TDD

Review

- Explain why you might want to Test
- Install a test framework
- Red-Green-Refactor
- Write basic tests that check for equality
- Practice pairing techniques