Activity 11-2 - Test Process

Remember the Three Rules of TDD?

1. You are not allowed to write any production code unless it is to make a failing test pass.
2. You are not allowed to write any more of a test than is sufficient to fail; compilation errors are failures.
3. You are not allowed to write any more production code than is sufficient to pass the one failing test.

Because it is by writing tests that you drive the development of your production code, TDD requires you to spend a lot of time thinking about what the next test is that you should write.

So how do we do this?

Techniques and Strategies for the Process of Writing Tests

1. Fake it 'til you make it

When you start to write a function, instead of writing a correct implementation, just return the result needed to make the test pass.

- Where have we seen an example of this?
- Which rule(s) of TDD does this help to enforce?
- **Useful for getting you started.**

2. Stairstep tests

Tests can be like stairs – sometimes the whole purpose of a test is to allow you to write the next test. Once the first test leads you to where you are going, you can delete it.

- Example: video at 20:00
  - Want to start testing a class, but class is not yet written. So write a test that uses the class ==> failure to compile forces you to write enough of the class to at least get your code to compile.
- Which rule(s) of TDD does this help to enforce?
- **Useful for getting you started.**

3. Assert first

Write test backwards – first write the assert, then iteratively fix one error at a time by adding only enough code needed to make the error go away.

- Example: video at 27:55
- Which rule(s) of TDD does this help to enforce?
- Useful if you need help figuring out the setup for a test

4. Triangulation

Add a second specific test that will force you to modify the code you are currently testing to be more general.

- Example: video at 34:17
- two tests + code being tested = triangle
- May lead to extracting an abstract class.
- Which rule(s) of TDD does this help to enforce?

Thought Experiment

Your team has just completed a large project using TDD. All the production source code is on one hard drive, and all the testing code is on a second hard drive.

One hard drive has a catestrophic failure.

Which drive is the one you would hope would crash? The one with the production code? Or the one with the testing code?

- **Discuss**
- Uncle Bob's answer at 59:10

Final thoughts

- What does a good test look like?
  - **The best tests read like well-written specifications.**
- Golden Rule of Writing Tests:
  - **Write the test that you'd like to read.**
- Test behavior, not APIs.

  - **Write tests to express the behavior of the system, independent of the API, and design the API based on the behaviour that your tests expect.**

- Four rules of simple design:

  1. Code must pass all its tests.
  2. Contains no duplicate code.
  3. Expresses all author's ideas.
  4. Minimize classes and variables.

  Or in other words,

  "First make it work, then make it right, then make it fast and small." –Kent Beck