SOFTWARE ENGINEERING II

LECTURE: 6

TDD AND UNIT TESTING

WHAT IS TDD?

Test-driven development (TDD) is a software development technique that uses short development iterations based on pre-written test cases that define desired improvements or new functions.

- Each iteration produces code necessary to pass that iteration's tests. Finally, the programmer or team refactors the code to accommodate changes.
- Note that test driven development *is a software design method*, not merely a method of testing.

TEST-DRIVEN DEVELOPMENT (TDD)

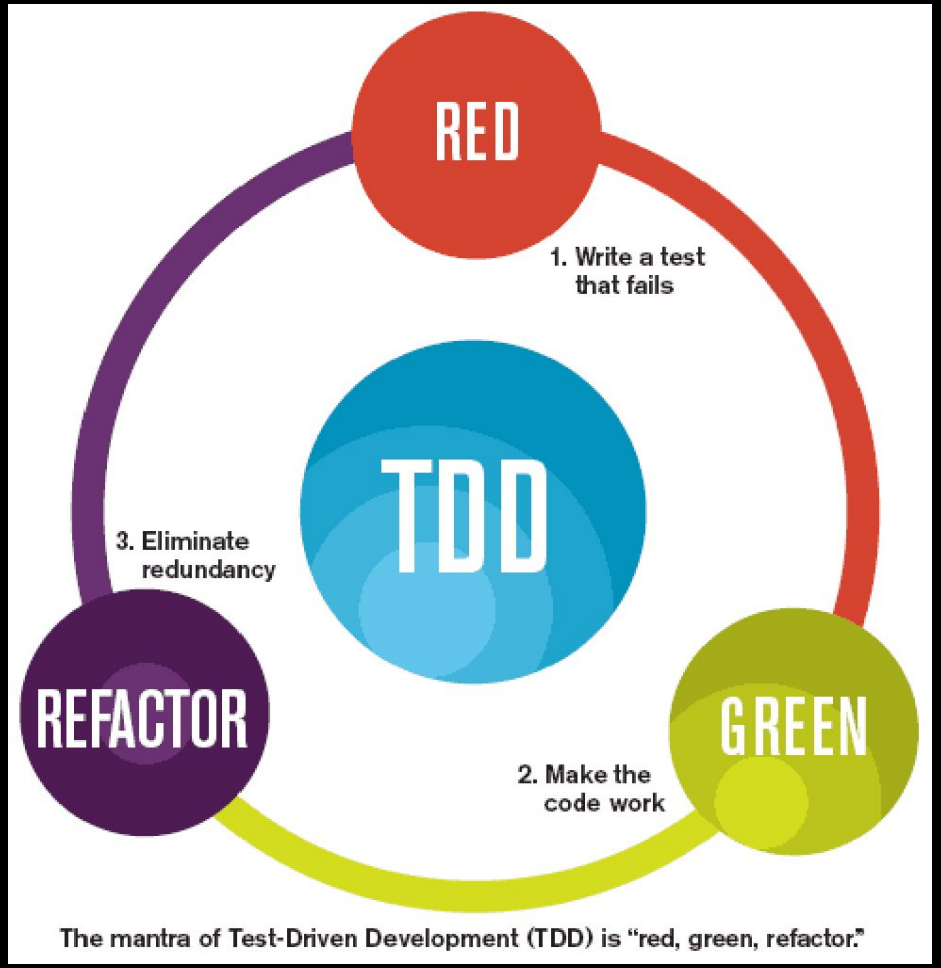
A software development process where a unit's tests are written before the unit's implementation and guide the unit's development as the tests are executed repeatedly until they all succeed, signaling complete functionality.

The TDD process steps are commonly shortened to "Red, Green, Refactor"

Used each time a new function, feature, object, class, or other software unit will be developed.

TEST-DRIVEN DEVELOPMENT (TDD)

Refactoring - process of restructuring existing computer code without changing its external behavior - refactoring improves nonfunctional attributes of the software



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

TEST DRIVEN DEVELOPMENT (TDD)

And repeat!

Red
Write a new test for a section of code (the “unit”)

Verify failure of the new test and the success of existing tests.

Green
Write some code to implement, modify, or develop the unit

Repeat until the tests pass. If one or more tests fail, continue coding until all tests pass. If the tests pass, the developer can be confident that the new/modified code works as specified in the test. Stop coding immediately once all tests pass.

Refactor
Improve non-functional code structure, style, and quality

Confirm tests pass. If one or more tests fail, the refactor caused problems; edit until all tests pass. If all tests pass, the developer can be confident that the refactor did not affect any tested functionality.

TEST-DRIVEN DEVELOPMENT (TDD)

Test-Driven Development (or test driven design) is a methodology

Common TDD misconceptions:

- TDD is not (just) about testing
- TDD is about design and development

By testing first, you design your code

UNIT TESTING

A test that invokes a **small, testable unit** of work in a software system and then checks a single assumption about the resulting output or behavior

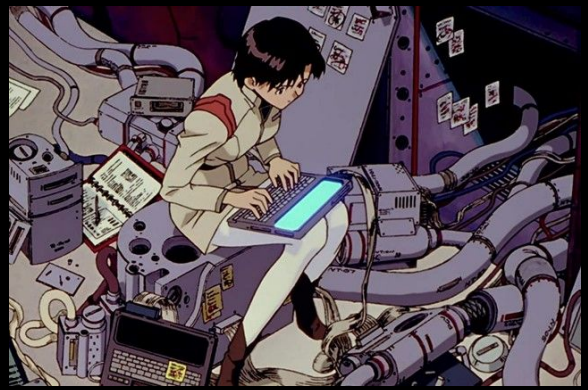
Key concept: **Isolation from other software components or units of code**

Low-level and focused on a tiny part or “unit” of a software system

Usually written by the programmers themselves using common tools

Typically written to be fast and run along with other unit tests in automation

UNIT TESTING



Typically uses coverage criteria as the exit criteria

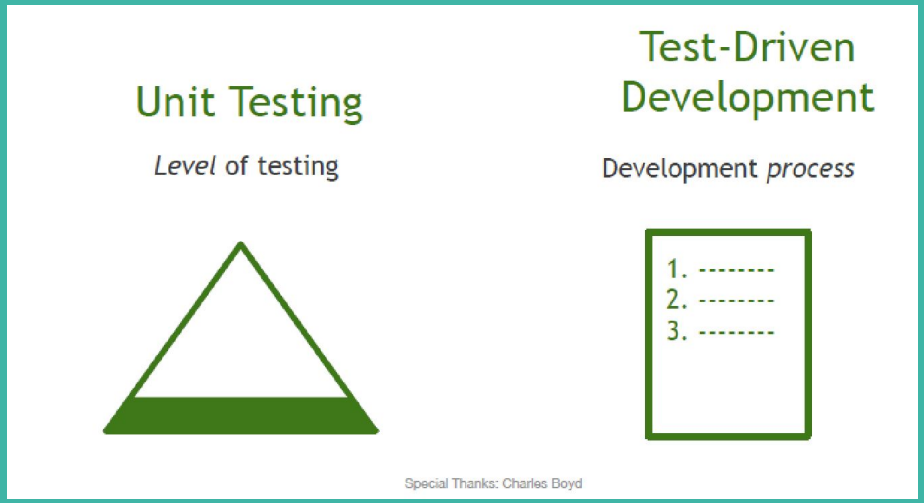
Definition of a “unit” is sometimes ambiguous

A unit is commonly considered to be the “smallest testable unit” of a system

Object-oriented (OO) languages might treat each object as a unit

Functional or procedural languages will likely treat each function as a unit

UNIT TESTING VS TDD



Unit testing and TDD are distinct concepts

While closely related and often used together, they could be used separately

EXAMPLE

```
import unittest
def add_one(x):
        return x+1

class Test_Add(unittest.TestCase):
    def test1(self):
        self.assertEqual(add_one(3), 4)
      def test2(self):
        self.assertEqual(add_one(3), 9)

if __name__ == '__main__':
    unittest.main()
```

Will pass

Will fail

UNIT TESTING TOOLS

Test Framework

- Defines the syntax that the tests are written
- Likely language-specific because it hooks into the system's execution
- Examples: Jasmine, Mocha, Jest (for JavaScript), PyTest or unittest (for Python), JUnit (for Java)

Test Runner

- Executes all (or a specific subset) of the system's unit tests and presents, displays, or otherwise outputs the results
- Could be a local test runner on a developer's computer or run on a server (e.g. a CI server)
- Often a basic test runner is built into the test framework, likely run via the command line
- Example: Karma (Java Script based for web application testing)

MORE TOOLS

Coverage Reporter

- Determines and provides a report on the test coverage metrics of a set of code
- May generate metrics such as statement, branch, function, executions per line, and line coverage grouped by file, class, component, or for the entire system
- Might be run independently or during each test executed by a test runner
- Example: Istanbul (for JavaScript), Coverage.py (python) → Tools like coveralls (https://coveralls.io/)

PROBLEMS WITH UNIT TESTING & TDD

Can seem to double development time (in the short run)

Practicing TDD can feel like it nearly doubles development time because the developer is writing twice the amount of code

No tests for the tests

Unit tests themselves are code and can be written incorrectly

Desire to make all tests “green”

The desire to see a passing or “green” result on all tests can cause developers to skip necessary tests or remove broken tests that should instead be fixed

Requires the initial and continuous development of “mocks”

Each external resource or dependency must be mocked to ensure isolation and this process can be time-consuming

PROBLEMS WITH UNIT TESTING & TDD

TDD does not necessarily result in “quality” tests and never guarantees proper code

Developers might skip edge cases, might write an untested branch by accident, etc.

TDD is not designed to build the best tests, it’s designed as a development process

Tedious

Sometimes TDD can feel like it gets in the way of “just coding.”

Advice: Try it anyway

CLASS ASSIGNMENT

Shortest Distance - Input 2 points (x1, y1) (x2, y2) [4 values - indicate to user which values are being input] and calculate the distance between the points using the distance formula (should be implemented without use of libraries with the exception of a square root library function). Floating point precision determination is key for testing.

Distance Formula: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Deliverable:

Runnable Source Code of the function and the test cases

Input a side x for a square. Write an area function that will calculate the area of the square with the given input x (using formula) and then return the area.

Deliverable:

Runnable Source Code of the function and the test cases

Screenshot of the output of the test file

END OF LECTURE 6

LECTURE: 6

TDD AND UNIT TESTING