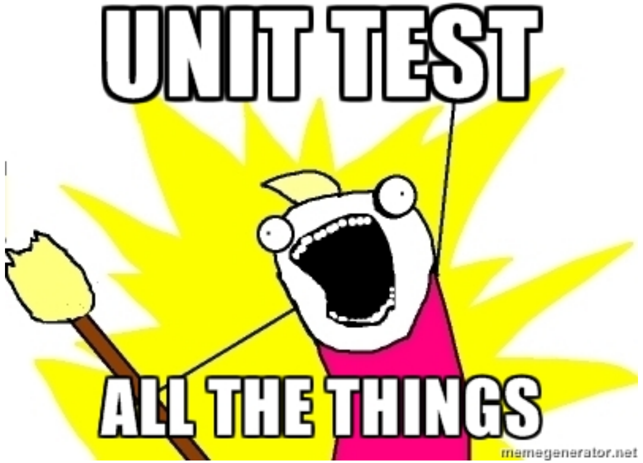


Software Testing for Continuous Delivery

Seminar 5: Unit Testing

Dr. Byron J. Williams

September 9, 2019

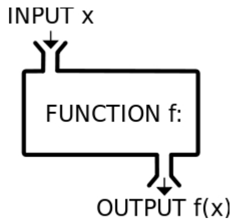
UF Herbert Wertheim College of Engineering
Department of Computer & Information Science & Engineering
UNIVERSITY *of* FLORIDA

Content adapted from "Developer Testing" by Alexander Tarlinder 1st Edition (2016)
Images courtesy Google Image Search

FICS Research

1. **It's consistent**—Given the same set of input data, it always returns the same output value, which doesn't depend on any hidden information, state, or external input.

2. **It has no side effects**—The function doesn't change any variables or data of any type outside of the function. This includes output to I/O devices.



CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

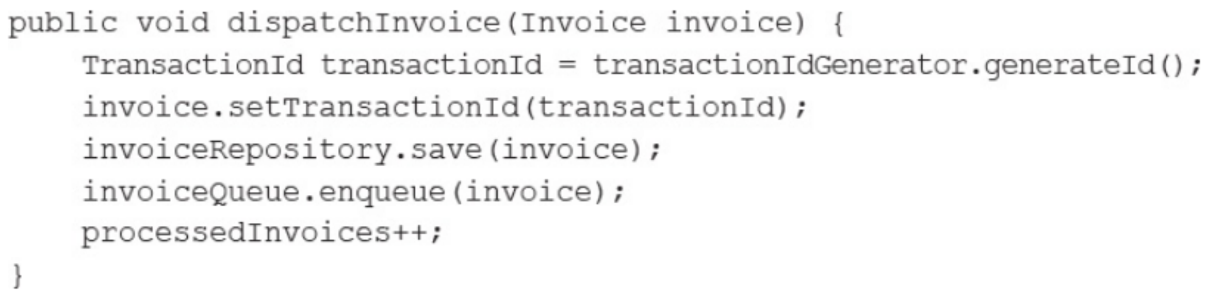
Indirect Input / Output

- Changing the value of a variable outside the scope of the function
- Modifying data referenced by a parameter (call by reference)
- Throwing an exception
- Doing some I/O

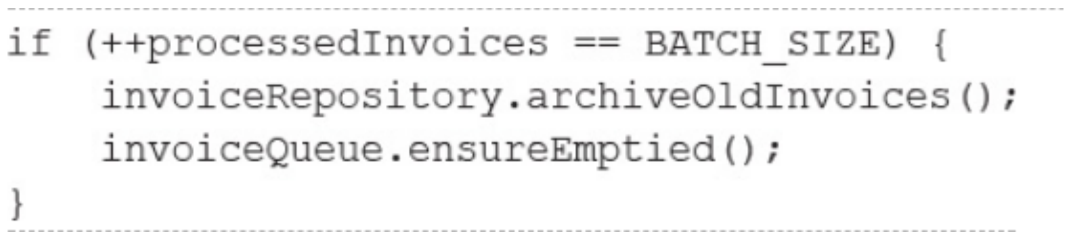
Pure Functions —> Functional Programming

RECAP

State



```
public void dispatchInvoice(Invoice invoice) {
    TransactionId transactionId = transactionIdGenerator.generateId();
    invoice.setTransactionId(transactionId);
    invoiceRepository.save(invoice);
    invoiceQueue.enqueue(invoice);
    processedInvoices++;
}
```



```
if (++processedInvoices == BATCH_SIZE) {
    invoiceRepository.archiveOldInvoices();
    invoiceQueue.ensureEmptied();
}
```

a program is described as **stateful** if it is designed to remember preceding events or user interactions; the remembered information is called the **state** of the system

"How do I set up a test so that I reach the correct state prior to verifying the expected behavior?"

RECAP

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.

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

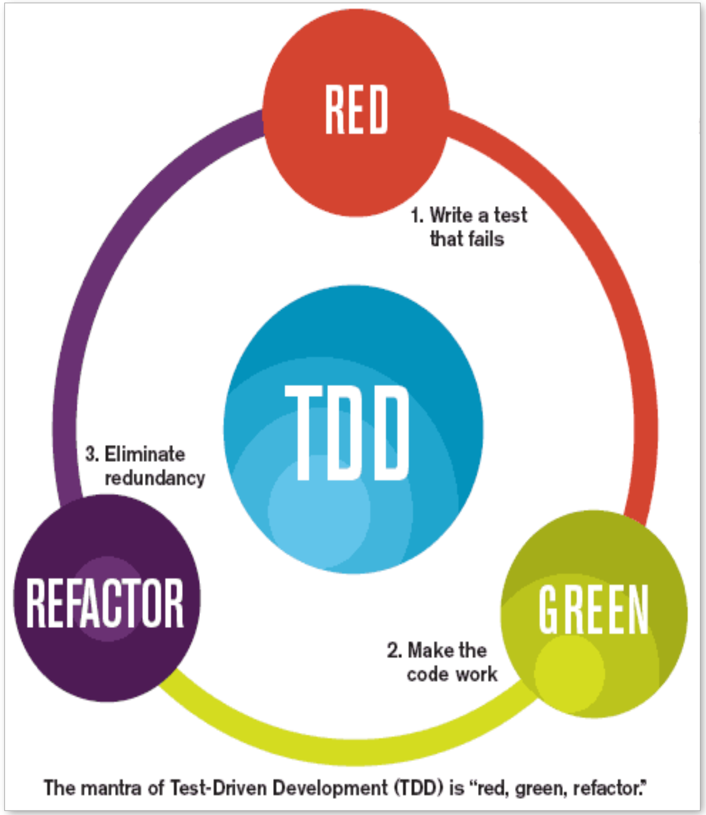

RED
1. Write a test that fails
TDD
GREEN
2. Make the code work
REFACTOR
3. Eliminate redundancy
The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Image from
abhishekmulay.com/blog/2014/12/13/javascript-unit-testing-with-jasmine/

RECAP

Herbert Wertheim College of Engineering
UNIVERSITY *of* FLORIDA

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

DEPARTMENT OF COMPUTER & INFORMATION SCIENCE & ENGINEERING
FLORIDA INSTITUTE FOR CYBERSECURITY RESEARCH (FICS)
Dr. Byron J. Williams

Unit Testing

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**

References: Some concepts from martinfowler.com/bliki/UnitTest.html and artofunittesting.com/definition-of-a-unit-test/

RECAP

Unit Testing

- Form of **white-box testing** that focuses on the implementation details
- 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
  - Many testing frameworks allow sets of unit tests to be **grouped**, allowing tests to be targeted at the function level and grouped by their parent object

References: Some concepts from martinfowler.com/bliki/UnitTest.html and artofunittesting.com/definition-of-a-unit-test/

RECAP

The Problem

How can we test this one function before it is used elsewhere in a program?

What if it was more complex?

What if it was in extremely large system?

What if we wanted to test it automatically so when it's modified, we can easily make sure it still works?

Answer: "Unit Testing"

Source Code

```python
def add_one(x):
    return x + 1
```

Unit Tests

```python
import pytest

def test_example_1():
    assert add_one(3) == 4

def test_example_2():
    assert add_one(-3) == -2

def test_example_3():
    assert add_one(5) == 20
```

Will Pass

A single "assertion"

Will Pass

Will Fail
(Need to fix this test…)

RECAP

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 (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)
  - Might also spin up mocks, a virtual environment, or any other resources the tests require
  - Often a basic test runner is built into the test framework, likely run via the command line
  - Example: Karma (for web application testing)

RECAP

More Tools

- **Mocks**
  - Provides a “mock” or **simulated implementation** of each external dependency or resource required by the methods being tested - aka. **stub**
  - May return random, dummy, or cached data
  - The need for mocks and their implementations varies between systems
  - *Example*: A simple program that returns dummy (fake but realistic) data for each outbound request made by a system that uses the Twitter API
- **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 ( )

Herbert Wertheim College of Engineering
UNIVERSITY *of* FLORIDA

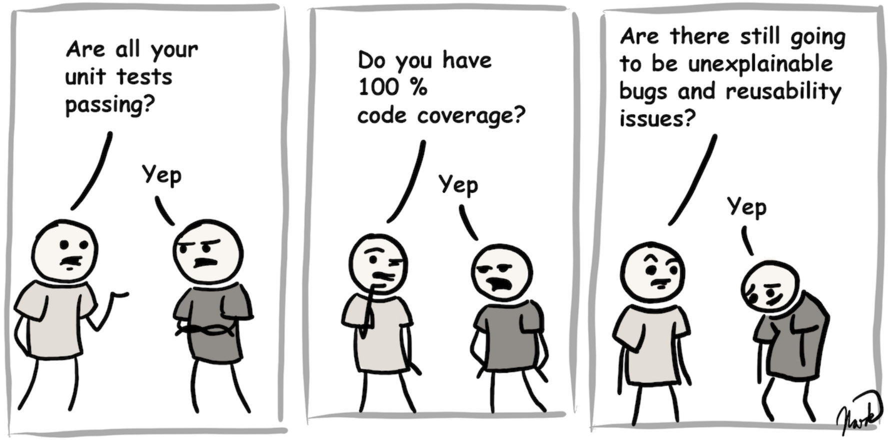
POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

DEPARTMENT OF COMPUTER & INFORMATION SCIENCE & ENGINEERING
FLORIDA INSTITUTE FOR CYBERSECURITY RESEARCH (FICS)
Dr. Byron J. Williams

Code Coverage

Code Coverage

- Used to find untested parts of your code
  - oft referred to as “test” coverage

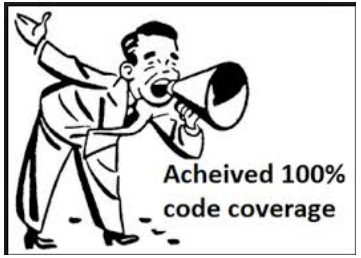

Are all your unit tests passing?
Yep
Do you have 100 % code coverage?
Yep
Are there still going to be unexplainable bugs and reusability issues?
Yep

- Measures the ‘amount’ of code executed by your test suite (i.e., has an assertion against it)
- Does not indicate how good your tests are
- Can be used as *part* of an exit criteria along with reliability measures (e.g., mttf)
- Low numbers are more telling than high coverage numbers (e.g., 99)
- What are good values for code coverage?

https://martinfowler.com/bliki/TestCoverage.html

Code Coverage Challenges

- If you make a certain level of coverage a target, people will try to attain it
- High coverage numbers are too easy to reach with low quality testing
- High percentage of coverage could still be problematic if critical parts of the application are not being tested
- Quality is considered by giving time to think about testing from a user perspective and not just by looking at lines of code
- Make code coverage apart of your CI workflow (if certain % not reached, fail build)
- Reasonable target ~80%



https://martinfowler.com/bliki/TestCoverage.html

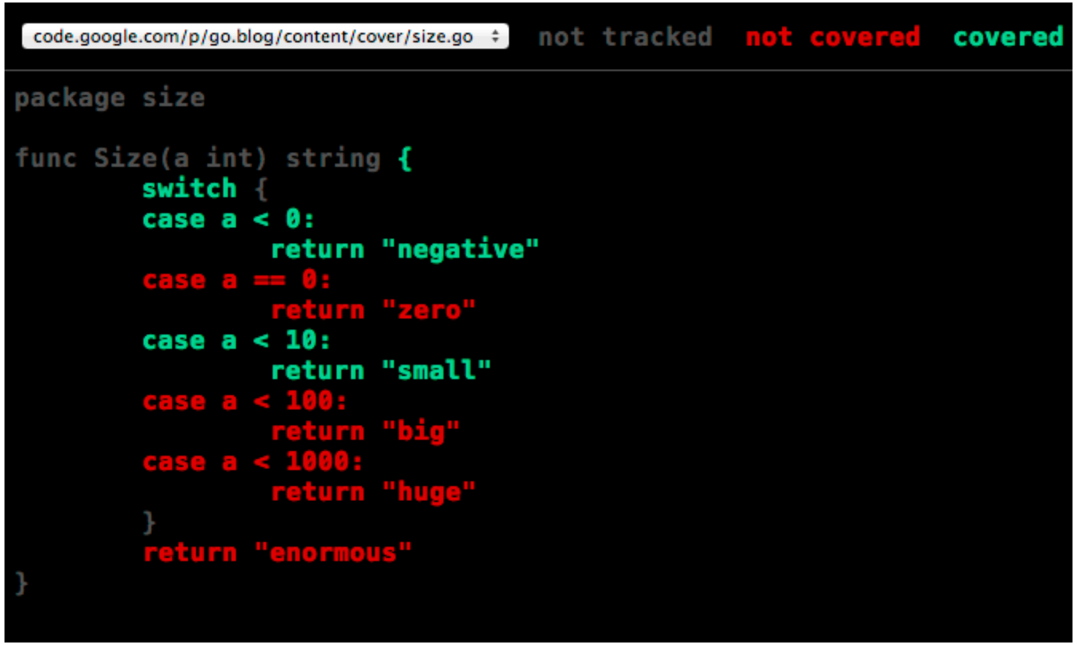
Code Coverage Metrics

- **Function coverage:** how many of the functions defined have been called.
- **Statement coverage**: how many of the statements in the program have been executed.
- **Branches coverage:** how many of the branches of the control structures (if statements for instance) have been executed.
- **Condition coverage:** how many of the boolean sub-expressions have been tested for a true and a false value.
- **Line coverage:** how many of lines of source code have been tested.

https://www.atlassian.com/continuous-delivery/software-testing/code-coverage

```
package size

func Size(a int) string {
    switch {
    case a < 0:
        return "negative"
    case a == 0:
        return "zero"
    case a < 10:
        return "small"
    case a < 100:
        return "big"
    case a < 1000:
        return "huge"
    }
    return "enormous"
}
```



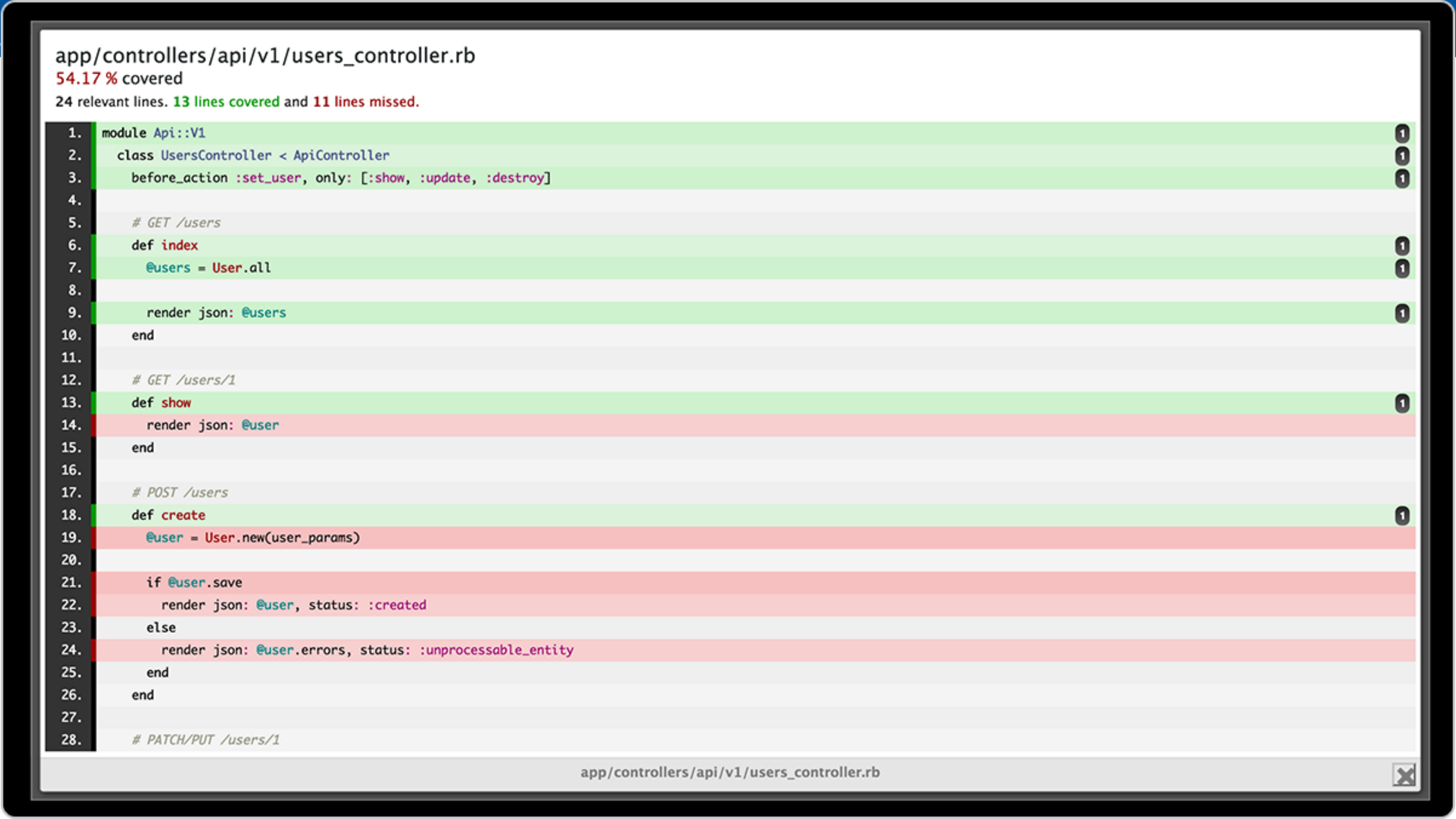
code.google.com/p/go.blog/content/cover/size.go   not tracked   not covered   covered

```
package size

func Size(a int) string {
        switch {
        case a < 0:
                return "negative"
        case a == 0:
                return "zero"
        case a < 10:
                return "small"
        case a < 100:
                return "big"
        case a < 1000:
                return "huge"
        }
        return "enormous"
}
```

Go Statement Coverage Example

https://blog.golang.org/cover



app/controllers/api/v1/users_controller.rb
54.17 % covered
24 relevant lines. 13 lines covered and 11 lines missed.

```ruby
module Api::V1
  class UsersController < ApiController
    before_action :set_user, only: [:show, :update, :destroy]

    # GET /users
    def index
      @users = User.all

      render json: @users
    end

    # GET /users/1
    def show
      render json: @user
    end

    # POST /users
    def create
      @user = User.new(user_params)

      if @user.save
        render json: @user, status: :created
      else
        render json: @user.errors, status: :unprocessable_entity
      end
    end

    # PATCH/PUT /users/1
```

app/controllers/api/v1/users_controller.rb

Herbert Wertheim
College of Engineering
UNIVERSITY *of* FLORIDA

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

DEPARTMENT OF COMPUTER & INFORMATION SCIENCE & ENGINEERING
FLORIDA INSTITUTE FOR CYBERSECURITY RESEARCH (FICS)
Dr. Byron J. Williams

More on Unit Testing

What is JUnit?

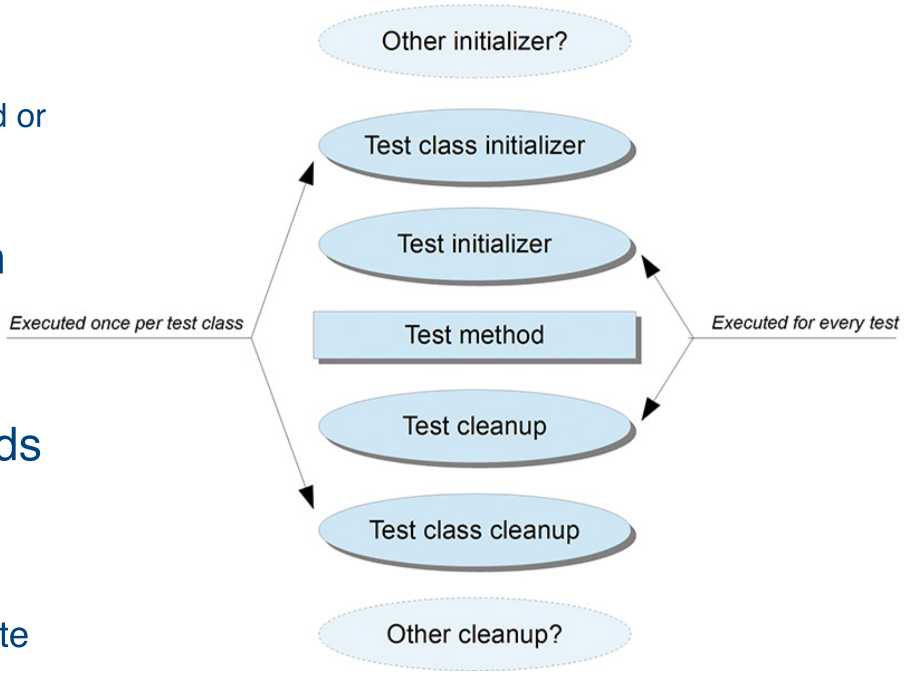
JUnit 5

- Open source Java testing framework used to write and run repeatable automated tests - open source (junit.org)
- A structure for writing test drivers adopted by many ***"xUnit"*** testing frameworks
- JUnit features include:
  - **Assertions** for testing expected results
  - Test features for sharing common test data
  - Test suites for **easily organizing** and running tests
  - Graphical and textual test runners
- JUnit can be used as stand alone Java programs (from the command line) or within an IDE

xUnit Tests

- xUnit can be used to test …

  … an entire object or part of an object i.e., a method or some interacting methods
  … interaction between several objects
- It is primarily intended for unit and integration testing, not system testing
- Each test is embedded into one test method
- A test class contains one or more test methods
- Test classes include :
  - A collection of test methods
  - Methods to set up the state before and update the state after each test and before and after all tests



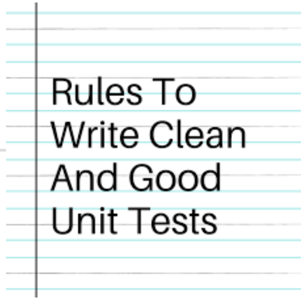
life cycle of a unit testing framework

JUnit Assertions

- `void` **`assertEquals`**`(boolean expected, boolean actual)` - Checks that two primitives/objects are equal
- `void` **`assertTrue`**`(boolean expected, boolean actual)` - Checks that a condition is true
- `void` **`assertFalse`**`(boolean condition)` - Checks that a condition is false
- `void` **`assertNotNull`**`(Object object)` - Checks that an object isn't null
- `void` **`assertNull`**`(Object object)` - Checks that an object is null
- `void` **`assertSame`**`(boolean condition)` - The assertSame() method tests if two object references point to the same object
- `void` **`assertNotSame`**`(boolean condition)` - The assertNotSame() method tests if two object references do not point to the same object
- `void` **`assertArrayEquals`**`(expectedArray, resultArray)` - The assertArrayEquals() method will test whether two arrays are equal to each other

https://www.tutorialspoint.com/junit/junit_using_assertion.htm

Good Unit Tests


Rules To Write Clean And Good Unit Tests

- Test a **single** logical concept in the system
- Have **full control** over all the pieces running and uses mocks to achieve this isolation as needed
- Are able to be fully **automated** and can be **run in any order**
- Return a **consistent result** for the same test (For example, no random numbers; save those for broader tests)
- Cause **no side effects** and **limits external access** (network, database, file system, etc.)
- Provide trustworthy results such that referring to the code is unnecessary to confirm it
- **Run fast** and are written with readable and maintainable test code, descriptive test names, and is organized or **grouped logically**

References: Adapted from http://artofunittesting.com/definition-of-a-unit-test/

Unit Testing & TDD Benefits

- Greatly improved regression testing!
- Allows refactoring without the fear of breaking the code
- Builds the automated unit test suite automatically
- Provides a easy way to test a newly written unit without the burden of the entire program or system
- Demonstrates development progress
- Produces a short feedback loop and rapid iterations
- Forces the developer to plan ahead
- Potentially reduces development time (in the long-term)
- Helps to ensure new code "works" (to the extent of the tests) as it is written
- The tests can serve as a detailed specification for the main code
- Goes hand-in-hand with several other key software development practices
- TDD allows you to always know that a minute ago, everything "worked"

Example: Benefit of Unit Testing

```
function absoluteValue(x){
    if(isPositive(x)==false){
        x = x*-1;
    }
    return x;
}
```

The function isPositive(…) is now covered by its own unit tests. So, here we trust that it works as its tests specify.

Note: To encourage isolation, you may wish to create mocks for calls to external functions

Source Code

Unit Tests

```
✓ expect(absoluteValue(5)).toEqual(5);
✓ expect(absoluteValue(-5)).toEqual(5);
✓ expect(absoluteValue(0)).toEqual(0);
✓ expect(absoluteValue('!')).toThrowError(TypeError);
```

Special Thanks: Charles Boyd

Problems with Unit Testing & TDD pt 1

- **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
  - Long term, however, previously mentioned the benefits of TDD can ultimately save development time
  - This issue is frequently the subject of debate when implementing TDD
- **Difficult to unit test functions with “side effects” or time-delays**
  - Functions that cause irreversible or otherwise un-mockable side effects (such as a imprecise and physical robot arm movement with feedback) may be difficult to unit test
  - To test functions that perform an action but do not return a value, the unit test must check the proper existence/ nonexistence of the side effect, which can become difficult (spies as a solution i.e., did the function get called)
- **No tests for the tests**
  - Unit tests themselves are code and can be written incorrectly
  - They might fail at first (as required by TDD), but never become “green” or successful (or successful too early) during the “write code” step because the test itself has one or more faults

Problems with Unit Testing & TDD pt 2

- **TDD doesn't always assist with changes in requirements, system-wide code structure, or the application programming interface (API)**
  - Especially in the early stages of a project, refactoring or other changes can be so extensive that it needs to "break" some tests
  - Those tests need to be modified/rewritten, but the corresponding main code for them might already exist, so the new tests are written without following TDD. Remember that TDD is focused on code base development, not test development
  - Existing unit tests are beneficial during most changes, particularly if a dependency is modified. The tests for each component that relies on that modified dependency can be run the to ensure the change didn't break the higher-level component.
- **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

Problems with Unit Testing & TDD pt 3

- **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
  - Unless the mock is built in/with the actual implementation of a dependency, the mock must be constantly updated as the API of the dependency
- **Sometimes difficult to test "private" functions/methods**
  - Sometimes, even when treating an object/class as a "unit," private functions (particularly complex ones) need to be unit tested individually. This means, depending on the language and/or testing framework, that the private function(s) must be exposed in some manner.
- **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**

Herbert Wertheim College of Engineering
UNIVERSITY *of* FLORIDA

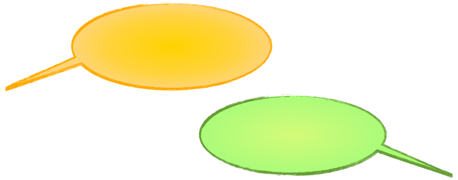
POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

DEPARTMENT OF COMPUTER & INFORMATION SCIENCE & ENGINEERING
FLORIDA INSTITUTE FOR CYBERSECURITY RESEARCH (FICS)
Dr. Byron J. Williams

Behavior-Driven Development

Behavior-Driven Development

- Encourages collaboration between Business Analysts, QA Engineers, Developers & Business Owners / Stakeholders
- Conversation focused & driven by business value
- Extends TDD by using **natural language** that non-technical stakeholders can understand
- User Stories & Acceptance Criteria typically defined with business or user focused language
- Developers implement acceptance criteria



BDD Principles

- BDD focuses on:
    - Where to start in the process
    - What to test and what not to test
    - How much to test in one go
    - What to call the tests
    - How to understand why a test fails

BDD provides software development and management teams with shared tools and a shared process to collaborate on software development

- BDD states that tests of any unit of software should be specified in terms of the desired behavior of the unit
    - **Describe** a test **set/suite** for the unit first ;
    - write what 'it', the test does;
    - make the tests fail;
    - implement the unit;
    - finally verify that the implementation of the unit makes the tests succeed
- Similar to TDD (description of desired behavior up front)