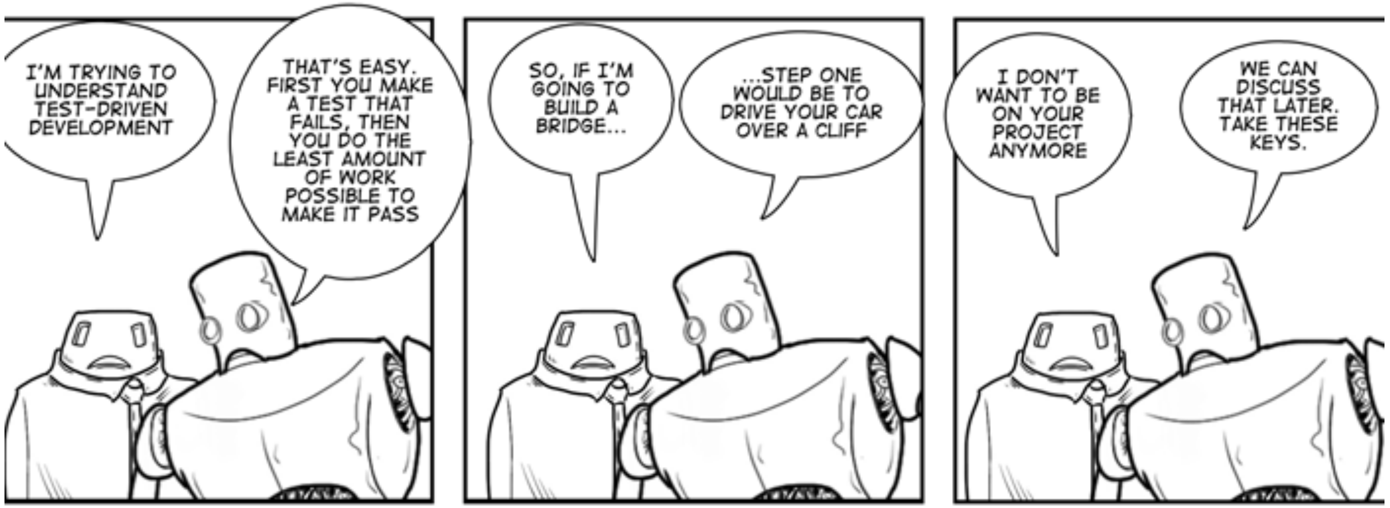

I'M TRYING TO UNDERSTAND TEST-DRIVEN DEVELOPMENT
THAT'S EASY. FIRST YOU MAKE A TEST THAT FAILS, THEN YOU DO THE LEAST AMOUNT OF WORK POSSIBLE TO MAKE IT PASS
SO, IF I'M GOING TO BUILD A BRIDGE...
...STEP ONE WOULD BE TO DRIVE YOUR CAR OVER A CLIFF
I DON'T WANT TO BE ON YOUR PROJECT ANYMORE
WE CAN DISCUSS THAT LATER. TAKE THESE KEYS.

Software Testing for Continuous Delivery

Seminar 4: Code-level Testability & Test-Driven Development

Dr. Byron J. Williams
August 28, 2019

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

FCS Research

Content adapted from “Developer Testing” by Alexander Tarlinder 1st Edition (2016)

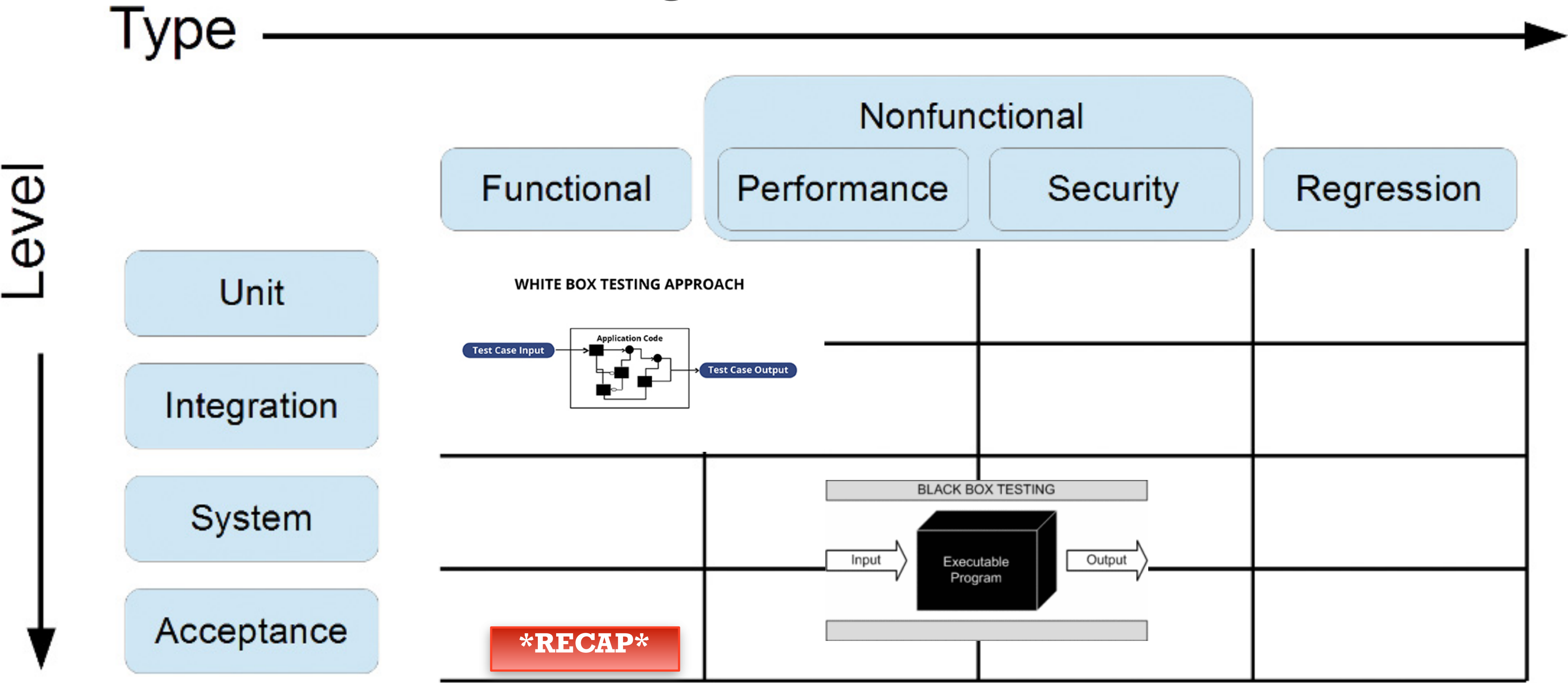
Defect

Error, ***Fault***, **or** ***Failure*** **in the system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways**

(think specifications / stories / expectations)

RECAP

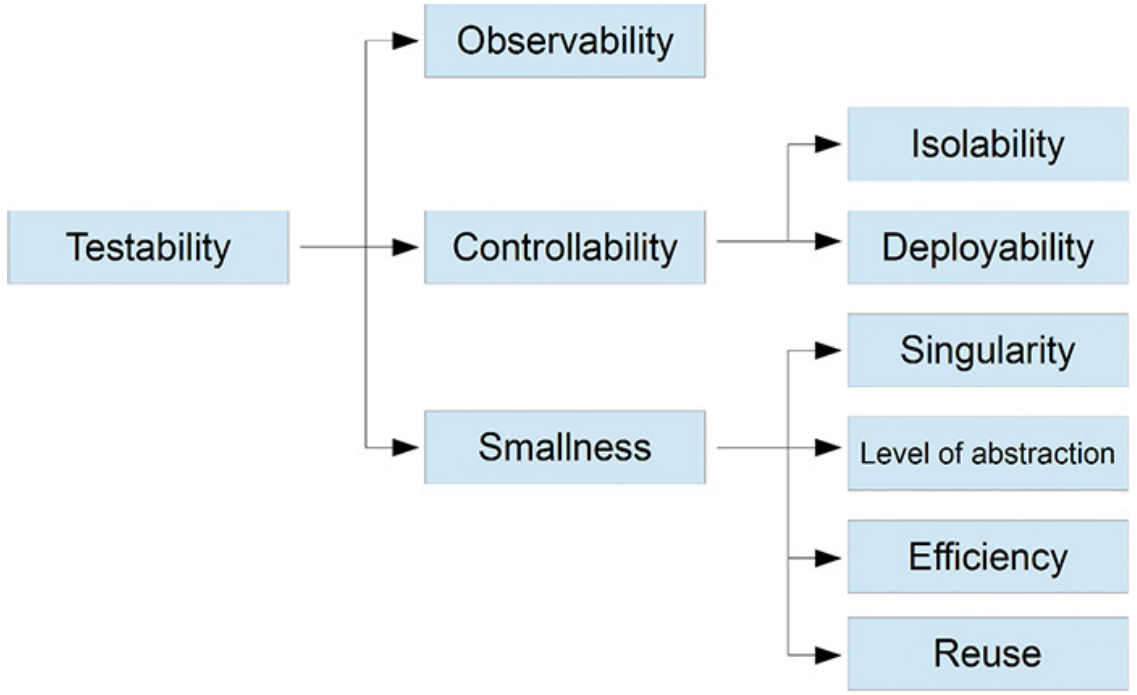
Testing Levels



Testability

- The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met
  - i.e., "program elements" - can be put into a known state, acted on, then observed
- Plainly speaking – **how hard it is to find faults in the software**
- Testability is dominated by two practical problems
  - How to provide the test values to the software
  - How to observe the results of test execution

Testability Explained
"Bertie, do you ever get the feeling we're an easy target?"
AG
Andy Glover cartoontester.blogspot.com © 2012



Testability Quality Decomposed

NOTE: When a ***program element*** is testable, it means that it can be put in a **known state**, **acted on,** and then **observed.** Further, it means that this can be done **without affecting any other program elements** and **without them interfering**

RECAP

Herbert Wertheim College of Engineering
UNIVERSITY *of* FLORIDA

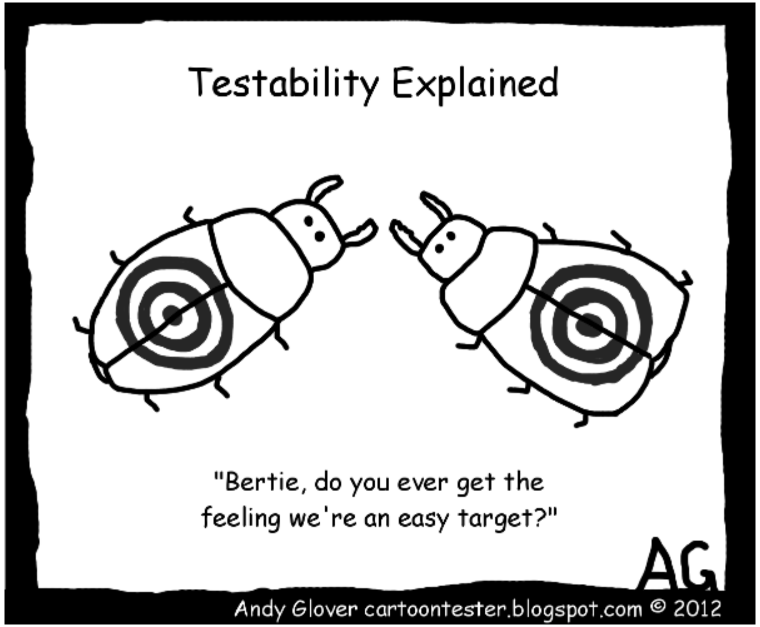
POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

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

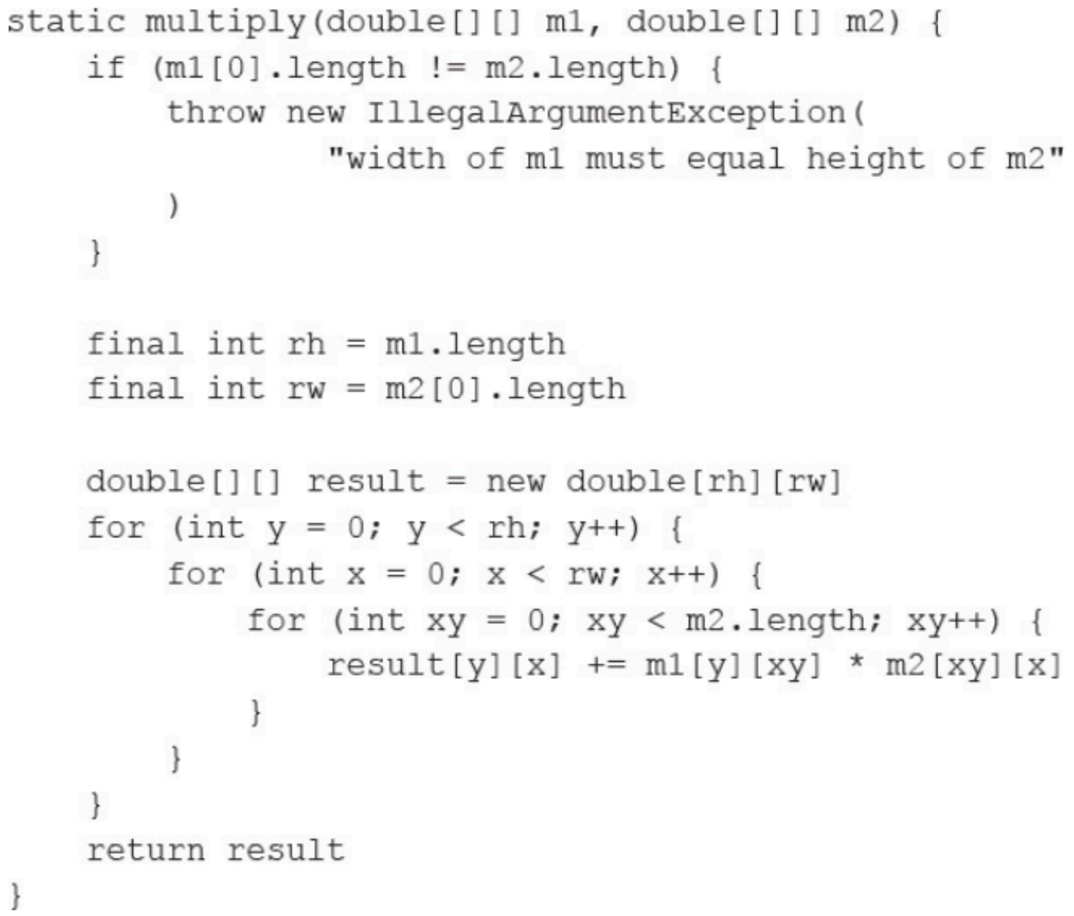
Code-level Testability

Code-level Testability

- Some constructs and behaviors in code have great impact on its testability


Testability Explained
"Bertie, do you ever get the feeling we're an easy target?"
AG
Andy Glover cartoontester.blogspot.com © 2012

Code Drivers — Testability

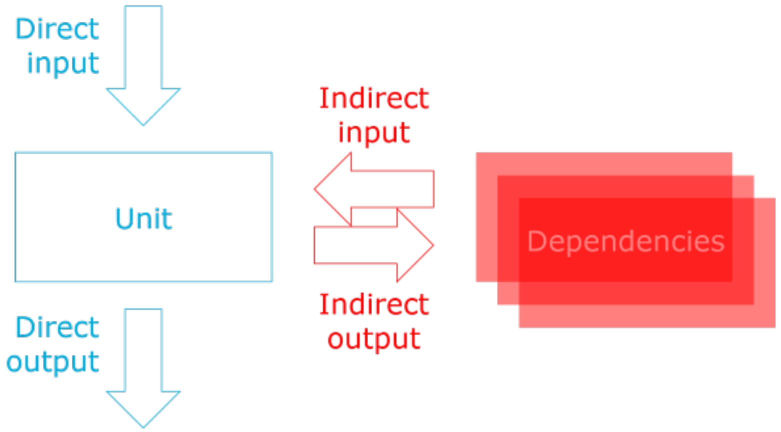


```
static multiply(double[][] m1, double[][] m2) {
    if (m1[0].length != m2.length) {
        throw new IllegalArgumentException(
                "width of m1 must equal height of m2"
        )
    }

    final int rh = m1.length
    final int rw = m2[0].length

    double[][] result = new double[rh][rw]
    for (int y = 0; y < rh; y++) {
        for (int x = 0; x < rw; x++) {
            for (int xy = 0; xy < m2.length; xy++) {
                result[y][x] += m1[y][xy] * m2[xy][x]
            }
        }
    }
    return result
}
```

Code Drivers — Testability

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

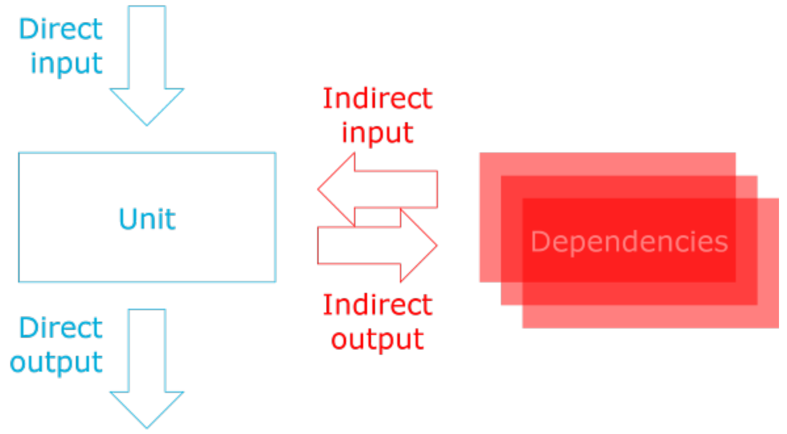
Direct Input / Output



- Program element's behavior affected solely by values that have been passed in via its public interface — ***direct input***
- Reliance on only direct input is quite a desirable property
  - largest concern is to find relevant inputs to pass as arguments to the tested method
  - not caring about other actors or circumstances that may affect its behavior
- ***direct output*** - return value - observable through public interface

https://blog.ploeh.dk/2017/02/02/dependency-rejection/

Indirect Input / Output

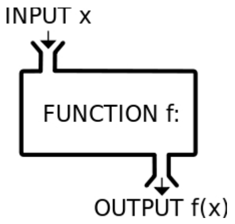


- Input is considered ***indirect*** if it isn't supplied using the program element's public interface
  - "Would I be able to test this without having access to the source code?" … "no," —> indirect input
  - e.g., static variables/methods, system properties, files, databases, queues, etc.
- ***indirect output*** - no return value or return plus other output - not observable through public interface
  - reliance on system for output verification

https://blog.ploeh.dk/2017/02/02/dependency-rejection/

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

State

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

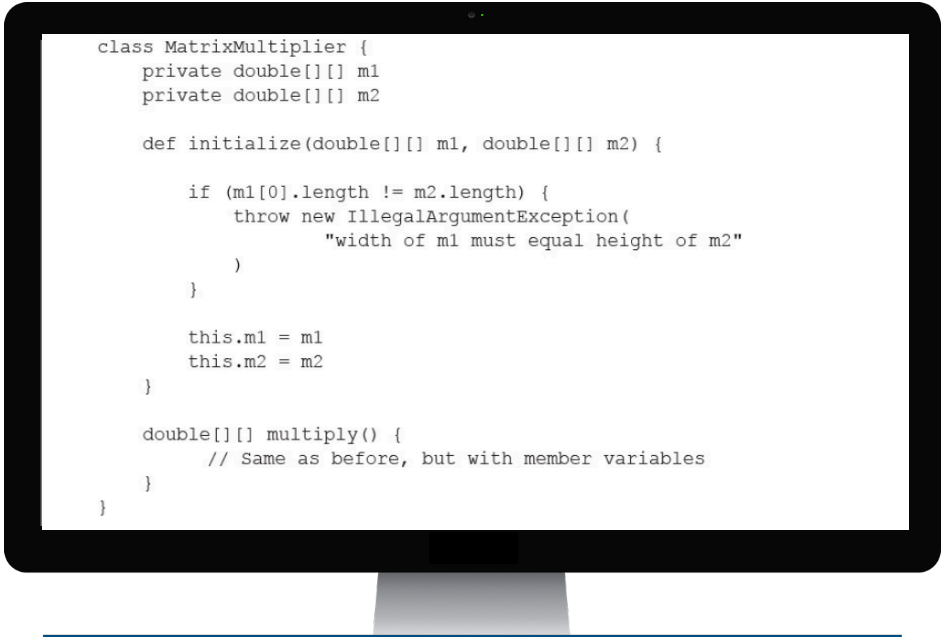
```
if (++processedInvoices == BATCH_SIZE) {
    invoiceRepository.archiveOldInvoices();
    invoiceQueue.ensureEmptied();
}
```

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?”

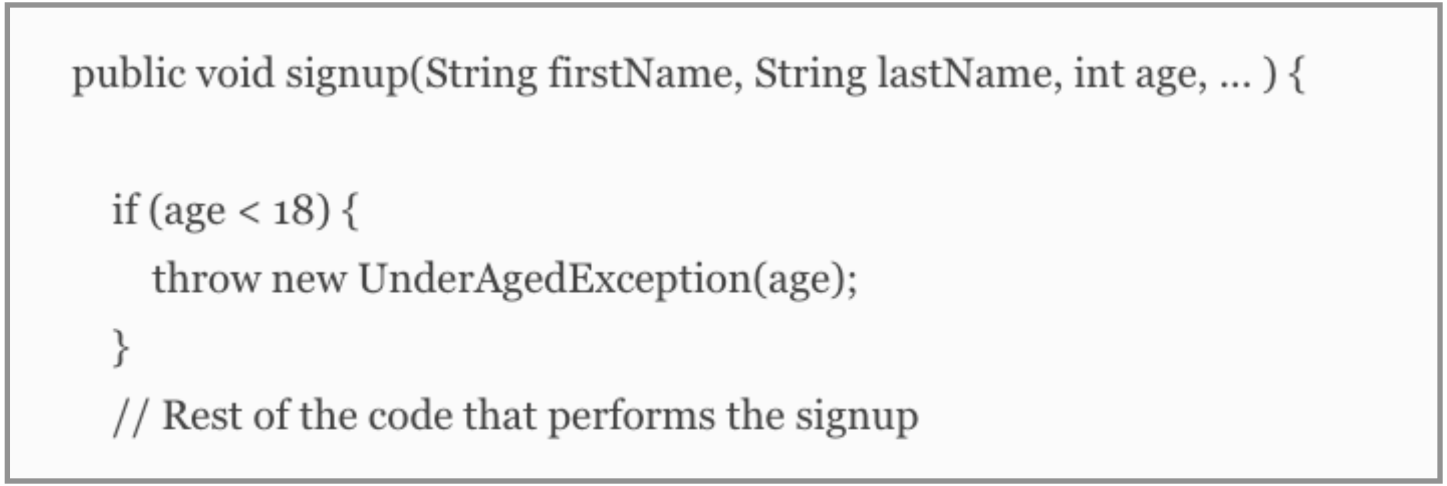
Temporal Coupling

- Temporal coupling is a close cousin of state
- the order of invocation
- Given a program element with functions f1 and f2, there exists a temporal coupling between them if, when f2 is called, it expects that f1 has been called first—that is, it relies on state set up by f1



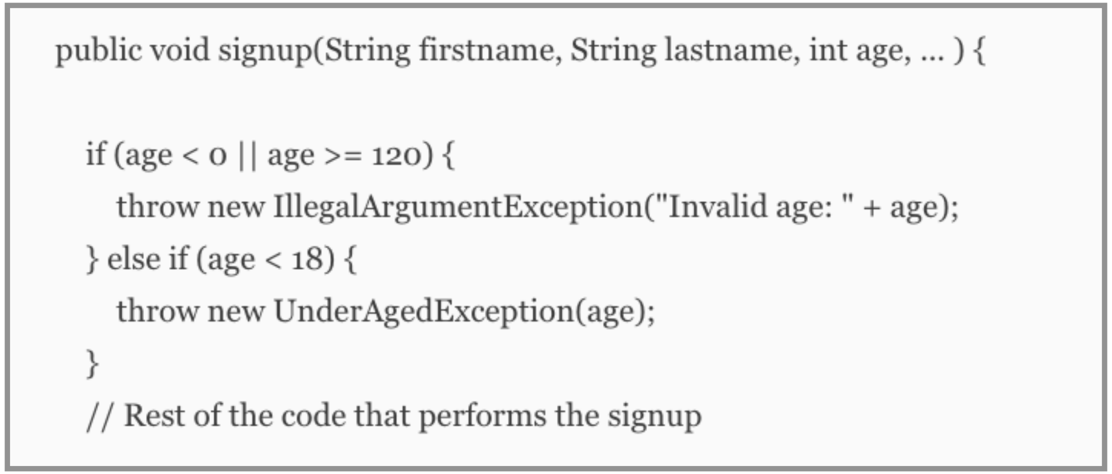
```
class MatrixMultiplier {
    private double[][] m1
    private double[][] m2

    def initialize(double[][] m1, double[][] m2) {

        if (m1[0].length != m2.length) {
            throw new IllegalArgumentException(
                    "width of m1 must equal height of m2"
            )
        }

        this.m1 = m1
        this.m2 = m2
    }

    double[][] multiply() {
         // Same as before, but with member variables
    }
}
```

temporal coupling arises as soon as one program element needs something to have happened in another program element in order to function correctly



```
public void signup(String firstName, String lastName, int age, ... ) {

  if (age < 18) {
    throw new UnderAgedException(age);
  }
  // Rest of the code that performs the signup
```

Data Types



```
public void signup(String firstname, String lastname, int age, ... ) {

  if (age < 0 || age >= 120) {
    throw new IllegalArgumentException("Invalid age: " + age);
  } else if (age < 18) {
    throw new UnderAgedException(age);
  }
  // Rest of the code that performs the signup
```

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

Test-Driven Development

What is TDD?

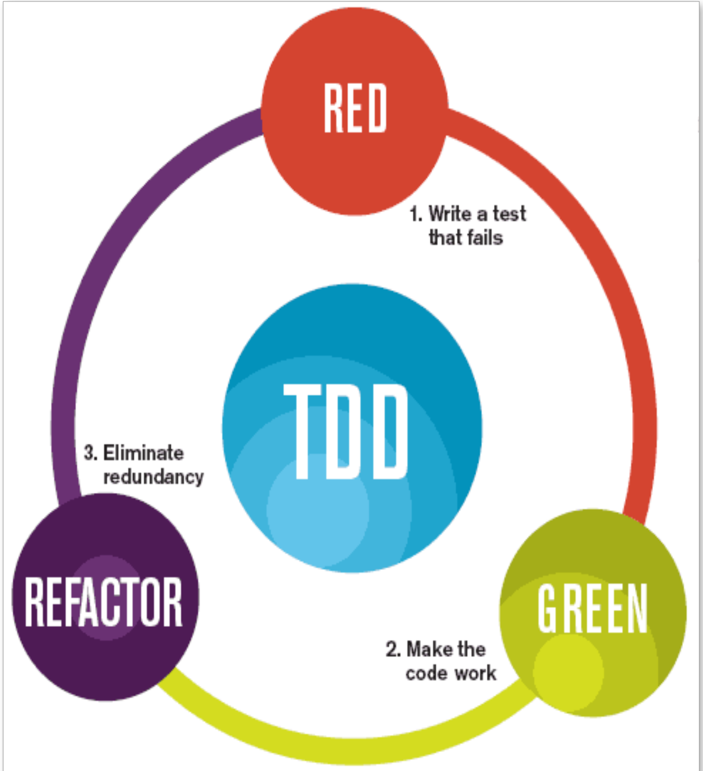
Test-driven development (TDD) is a software development **technique** that uses **short development iterations** based on pre-written test cases that define desired improvements or new functions. Each iteration **produces code necessary to pass that iteration's tests**. Finally, the programmer or team **refactors** the code to accommodate changes. A key TDD concept is that preparing tests before coding **facilitates rapid feedback** changes. Note that test-driven development is **a software design method**, not merely a method of testing.

https://en.wikipedia.org/wiki/Test-driven_development

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



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 1/22

Test-Driven Development (TDD)

1. **Red**
   - Write a new test for a section of code (the "unit")
   - Verify failure of the new test and the success of existing tests. If the new test passes, verify that it is not redundant, then start composing the next test (step 1a).
2. **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.
3. **Refactor**
   - Refactor the code to 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 the tests all pass, the developer can be confident that the refactor did not effect any tested functionality.
4. **[Repeat]**

TDD — Design Methodology

- 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

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**

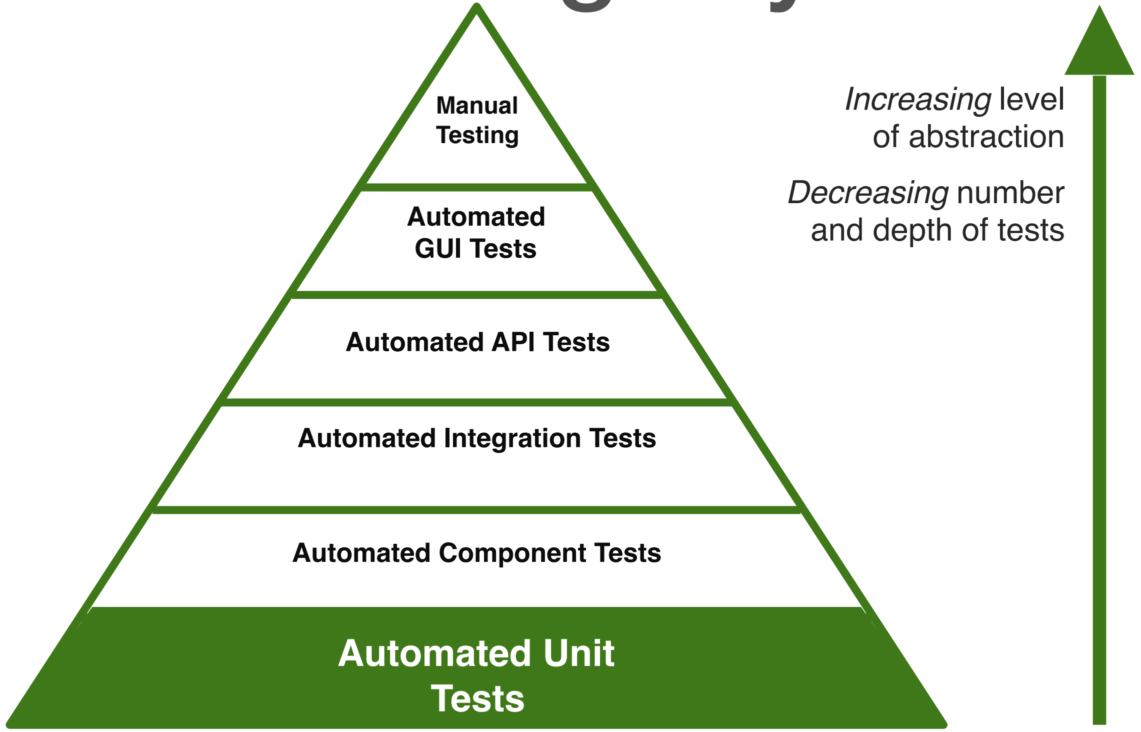
References: Some concepts from martinfowler.com/bliki/UnitTest.html and artofunittesting.com/definition-of-a-unit-test/

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/

Software Testing "Pyramid"


Manual Testing
Automated GUI Tests
Automated API Tests
Automated Integration Tests
Automated Component Tests
Automated Unit Tests
Increasing level of abstraction
Decreasing number and depth of tests

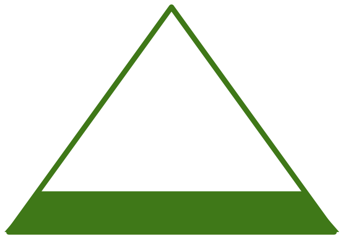
References: Based on watirmelon.com/tag/testing-pyramid/

Unit Testing vs TDD

- Unit testing and TDD are distinct concepts
- While closely related and often used together, they could be used separately
- The following slides and demos with present the two concepts combined, as they are frequently used together

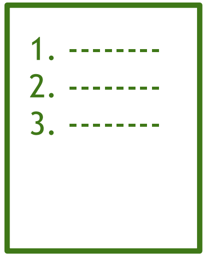
Unit Testing

Level of testing



Test-Driven Development

Development *process*



Special Thanks: Charles Boyd

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

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

Unit Tests

```
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…)

Special Thanks: Charles Boyd

Example: Unit Tests (JavaScript + Jasmine)

```
function isPositive(x) {
    return x >= 1;
}
```

The unit tests revealed one or more defects in this code, hopefully before other functions relied on it

Source Code

Results	Unit Tests
✓	expect(isPositive(5)).toEqual(true);
✓	expect(isPositive(-5)).toEqual(false);
✓	expect(isPositive(1)).toEqual(true);
✓	expect(isPositive(-1)).toEqual(false);
✗	expect(isPositive(0.5)).toEqual(true);
✓	expect(isPositive(-0.5)).toEqual(false);
✗	expect(isPositive(0)).toEqual(true);

Special Thanks: Charles Boyd