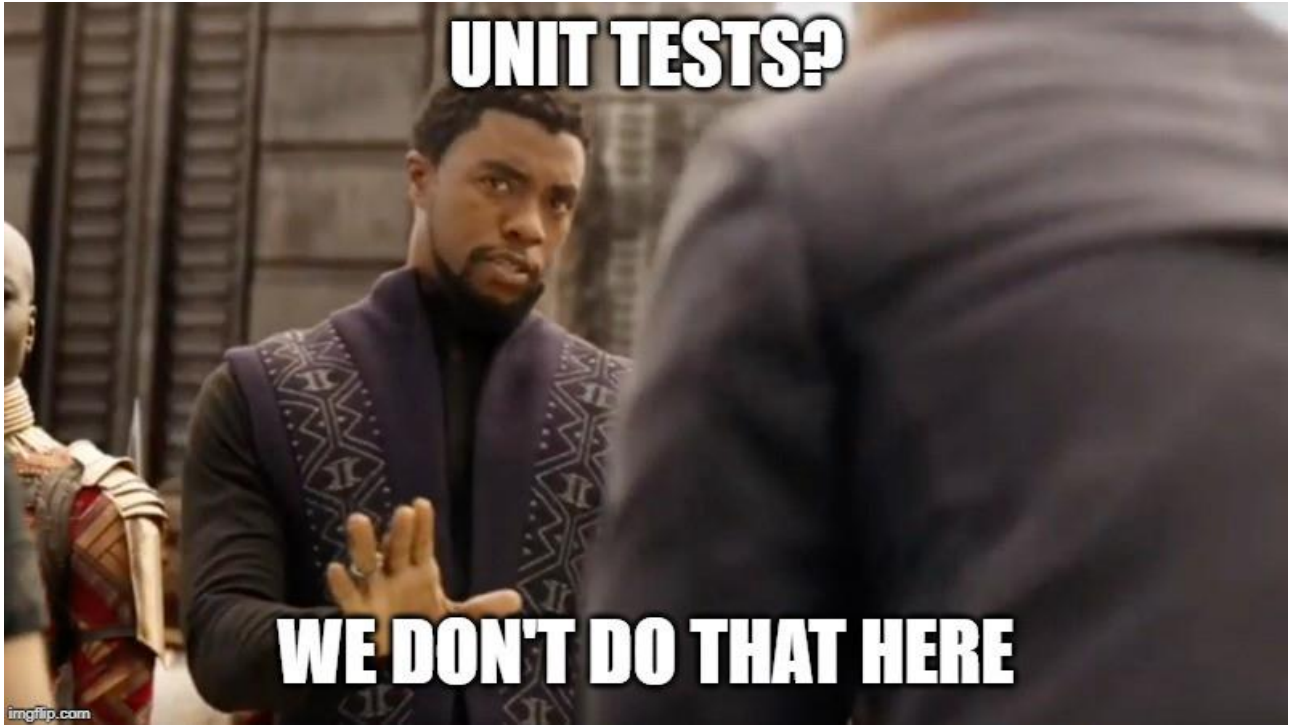

UNIT TESTS?
WE DON'T DO THAT HERE
imgflip.com

Module 1-14

Unit Testing

Objectives

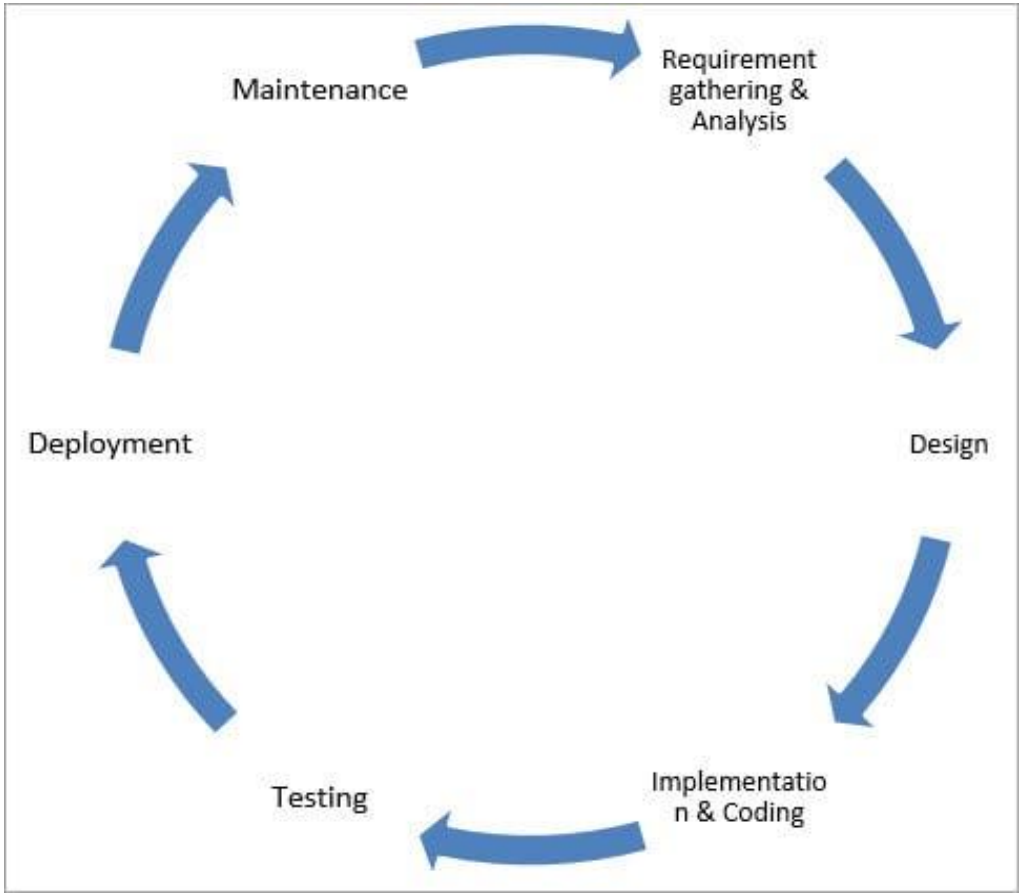
- What is testing?
- Manual vs. Automated testing
- Exploratory vs. Regression testing
- Unit, Integration and Acceptance testing
- How to write unit tests
- How to choose the correct asserts from a testing framework
- Be able to describe boundary cases and how to determine what the boundary cases are in a piece of code

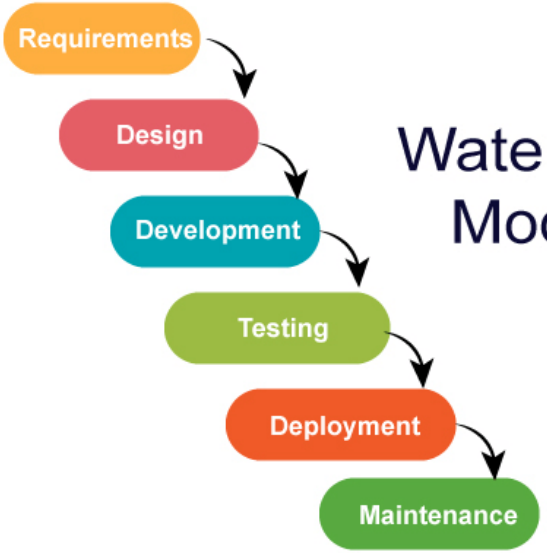
Testing

Goes without saying… we need a way to test the code we've written.

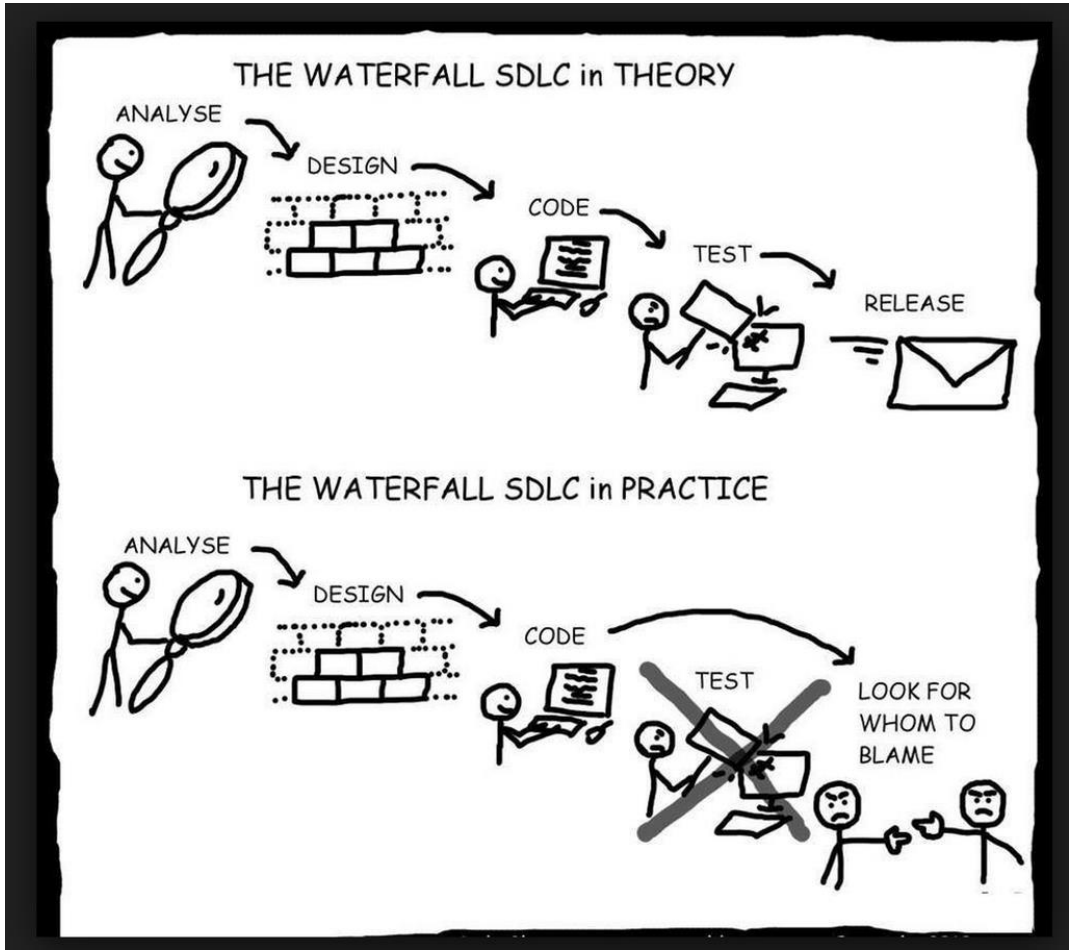
But first, let's talk about the SDLC (Software Development Life Cycle)

SDLC

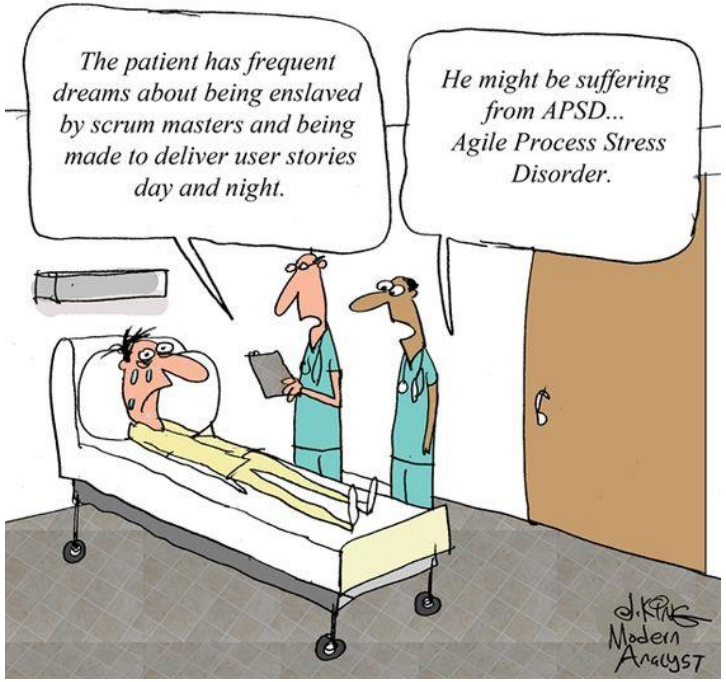

Requirement gathering & Analysis
Design
Implementatio n & Coding
Testing
Deployment
Maintenance



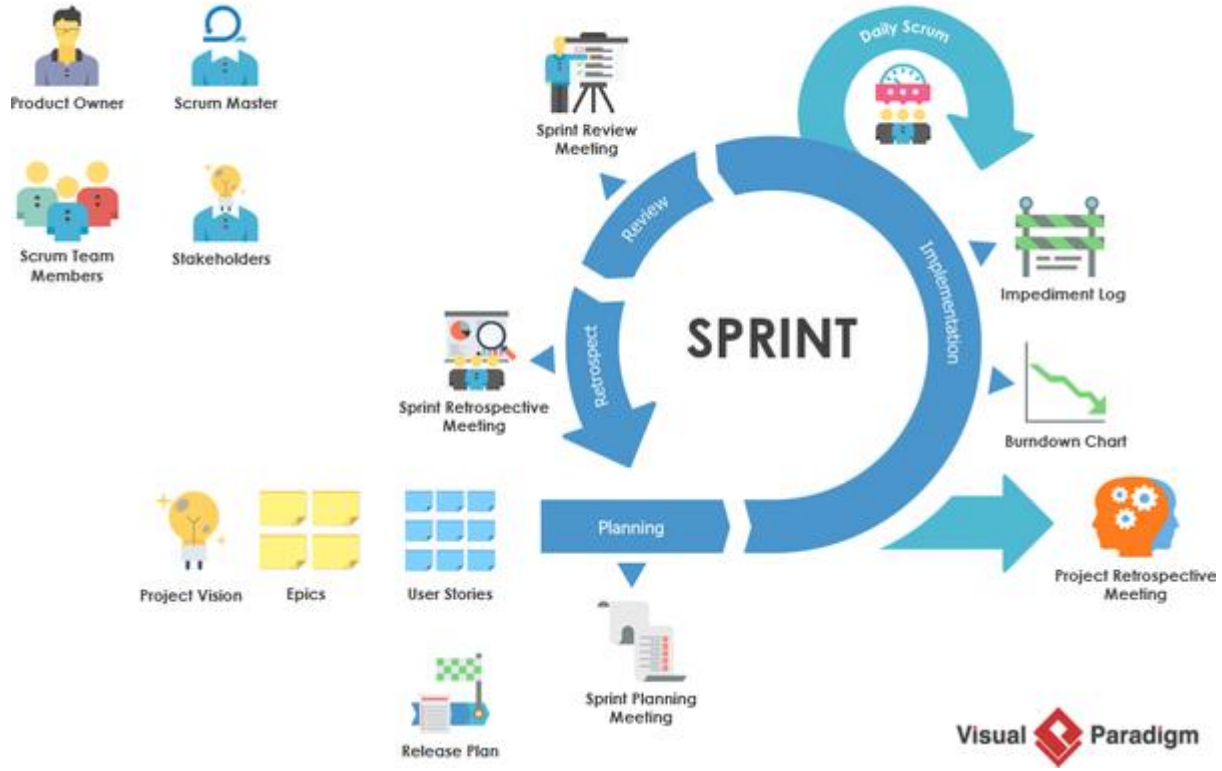
Waterfall Model




Agile Methodology
Review
Test
Build
Design
Plan
Launch
Sprint 1
Sprint 2
Sprint 3


The patient has frequent dreams about being enslaved by scrum masters and being made to deliver user stories day and night.
He might be suffering from APSD... Agile Process Stress Disorder.
Modern Analyst

The Agile – Scrum Framework



SOFTWARE TESTER

What my friends think I do

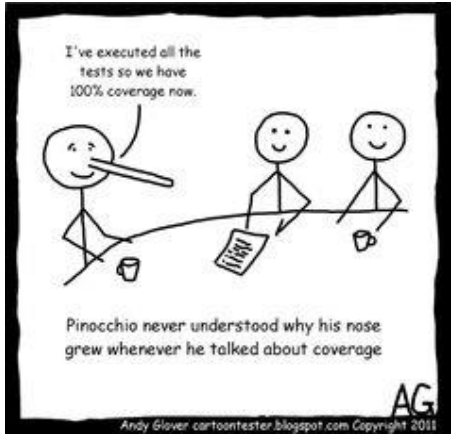
What my mom thinks I do

What society thinks I do

What programmers think I do

What I think I do

What I actually do



Manual Testing vs Automated Testing

- Historically, tests were written on a third party tool (i.e. Excel) with a script a tester should follow. The results are recorded.
  - This is a very error prone manual process.
  - Exploratory and Usability testing are best done manually.

- Over time, testing frameworks were introduced so that we could write code that tests code in your system.
  - This made testing more automated.
  - However, the quality of the tests now partially depends on the developer’s knowledge of the testing framework.

Exploratory Testing vs Regression Testing

- **Exploratory Testing** explores the functionality of the system looking for defects, missing features, or other opportunities for improvement. Almost always manual.

- **Regression Testing** validates that existing functionality continues to operate as expected.

Types of Testing

- **Unit Testing**: Tests the smallest units possible (i.e. methods of a class).
- **Integration Testing:** Tests how various units or parts of the program interact with each other.
    - It can also be used to validate some external dependencies like database systems or API’s.
- **User Acceptance Testing:** Tests the functionality from the end user’s perspective. It can be conducted by a non-technical user.

Other Types of Testing

- **Security Testing:** Is our data safe from unauthorized users?
- **Performance Testing:** it works with 1 user, what about a million?
- **Platform Testing:** Works great on my laptop, what if I pull up the app from my phone?

Unit Testing in Java: Introduction

The most commonly used testing framework in Java is **JUNIT**.

- JUNIT is written in Java and will leverage all the concepts you've learned so far: declaring variables, calling methods, instantiating objects.
- All related tests can be written in a single test class containing several methods, each method could be a test.
- Each method should contain an assertion, which compares the result of your code against an expected value.

The AAA pattern of unit testing

- Arrange – arrange the conditions of the test
- Act – perform action of interest
- Assert – validate that the expected outcome occurred by means on an assertion

```
public class Calculator {
   public double sum(double first, double second) {
     return first + second;
   }
}
```

(1) class container
(2) xUnit's attribute indicating a test (Java uses Junit and @Test)
(3) Name of unit Test
(4) Arrange
(5) Act
(6) Assert

```
public class CalculatorTest {   (1)

    @Test (2)
    public void sum_of_two_numbers() { (3)
       // Arrange
       double first = 10;                              (4)
       double second = 20;                             (4)
       Calculator calculator = new Calculator();   (4)

       // Act
       double result = calculator.sum(first, second); (5)

       // Assert
        assertEquals(30, result);   (6)
```

Unit Testing in Java: Assertions

An assertion is the result of a a comparison between an actual value of an expected value. Supposed we have a Java method that returned the following:

```
public static boolean divBy2(int i) {
        return i%2 == 0;
}
```

Assertion 1: If I run divBy2(4) the result of invoking the method should be true.

If divBy2(4) returns false, then the assertion has failed.

Assertion 2: If I run divBy(5) the result of invoking the method should be false.

If the method is invoked and the result is true, then the assertion has failed.

Unit Testing in Java: Production Code vs Test Code

- Production code refers to the actual code for your project.
- Test code is the code that is designed to test Production Code

- Production code often tends to be DRY (Don’t Repeat Yourself), test code tends to be more WET (Write Everything Twice)

- Production code is for fulfilling the contract. Test code is for verify this contract.

Unit Testing in Java: Example

Production Code

```
package te.examples.testingexamples;

public class MyApp {

   public boolean divBy2(int number) {
      return number%2==0;
   }

   public String concatenator(String [] wordArray) {
      String output = "";

      for (String word : wordArray) {
         output += word;
      }

      return output;
   }
}
```

These two are tests designed to check if divBy2 is working property.

Test Code

```
// A lot of imports up top, removed for brevity
public class TestContainingClass {

   @Test
   public void threeDivByTwoShouldReturnFalse() {

      MyApp app = new MyApp();
      boolean actualResult = app.divBy2(3);
      boolean expectedResult = false;

      Assert.assertEquals(expectedResult, actualResult);
    //  Assert.assertFalse(actualResult);
   }

   @Test
   public void fourDivByTwoShouldReturnTrue() {

      MyApp app = new MyApp();
      boolean actualResult = app.divBy2(4);
      boolean expectedResult = true;

      Assert.assertEquals(expectedResult, actualResult);
      //  Assert.assertTrue(actualResult);
   }
}
```

Unit Testing in Java: Anatomy of Test Method

Let’s take a closer look at a test method and what happens inside it:

We are using an @Test annotation to indicate this method is a test.

Tests are typically void methods, they follow the same syntax rules as regular methods.

We need to bring in the test collaborators, in this case an instance of the class MyApp

We run any methods in the collaborator that we want to test, obtain the actual result and compare against what we are expecting.

Test Code

```
// A lot of imports up top, removed for brevity
public class TestContainingClass {

    @Test
    public void threeDivByTwoShouldReturnFalse() {

        MyApp app = new MyApp();
        boolean actualResult = app.divBy2(3);
        boolean expectedResult = false;

        Assert.assertEquals(expectedResult,
                actualResult);
    }
}
```

Unit Testing in Java: Multiple Tests

A testing class can contain multiple tests. The same production method can be called and tested as many times as needed.

This class contains two tests.

```
public class TestContainingClass {

    @Test
    public void threeDivByTwoShouldReturnFalse() {
        // test content
    }

    @Test
    public void fourDivByTwoShouldReturnTrue() {
        // test content
    }
}
```

Unit Testing in Java: Before & After

You can specify that certain pieces of code be run before and after every single test.

```
public class TestContainingClass {

    @Before
    public void setUp() throws Exception {

       System.out.println("Test starting.");
    }

    @After
    public void tearDown() throws Exception {

       System.out.println("Test complete.");
    }

    @Test
    public void threeDivByTwoShouldReturnFalse() {
        // Test content.
    }

    @Test
    public void fourDivByTwoShouldReturnTrue() {
        // Test content.
    }
}
```

Anything in the @Before block will run right before a test.

Anything in the @After block will run right after the test.

So the order of operations is:

1. run setup()
2. Run threeDivByTwo… test
3. run tearDown()
4. run setup()
5. Run fourDivByTwo… test
6. Run tearDown();

Unit Testing in Java: Before & BeforeClass

- `@Before`

  `public void method()`

  The `Before` annotation indicates that this method must be executed before each test in the class, so as to execute some preconditions necessary for the test.

- `@BeforeClass`

  `public static void method()`

  The `BeforeClass` annotation indicates that the static method to which is attached must be executed once and before all tests in the class. That happens when the test methods share computationally expensive setup (e.g. connect to database).

- https://www.kualitee.com/software-testing/top-5-software-testing-fails-2018/

- “If you don’t like testing your product, most likely your customers won’t like to test it either.” (*Anonymous*)
- “If debugging is the process of removing bugs, then programming must be the process of putting them in.” (*Edsger Dijkstra*)
- “One man’s crappy software is another man’s full-time job.” (*Jessica Gaston*)
- “Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.” (*Edsger Dijkstra*)

Objectives

- What is testing?

Objectives

- What is testing?
- Manual vs. Automated testing

Objectives

- What is testing?
- Manual vs. Automated testing
- Exploratory vs. Regression testing

Objectives

- What is testing?
- Manual vs. Automated testing
- Exploratory vs. Regression testing
- Unit, Integration and Acceptance testing

Objectives

- What is testing?
- Manual vs. Automated testing
- Exploratory vs. Regression testing
- Unit, Integration and Acceptance testing
- How to write unit tests

Objectives

- What is testing?
- Manual vs. Automated testing
- Exploratory vs. Regression testing
- Unit, Integration and Acceptance testing
- How to write unit tests
- How to choose the correct asserts from a testing framework

Objectives

- What is testing?
- Manual vs. Automated testing
- Exploratory vs. Regression testing
- Unit, Integration and Acceptance testing
- How to write unit tests
- How to choose the correct asserts from a testing framework
- Be able to describe boundary cases and how to determine what the boundary cases are in a piece of code