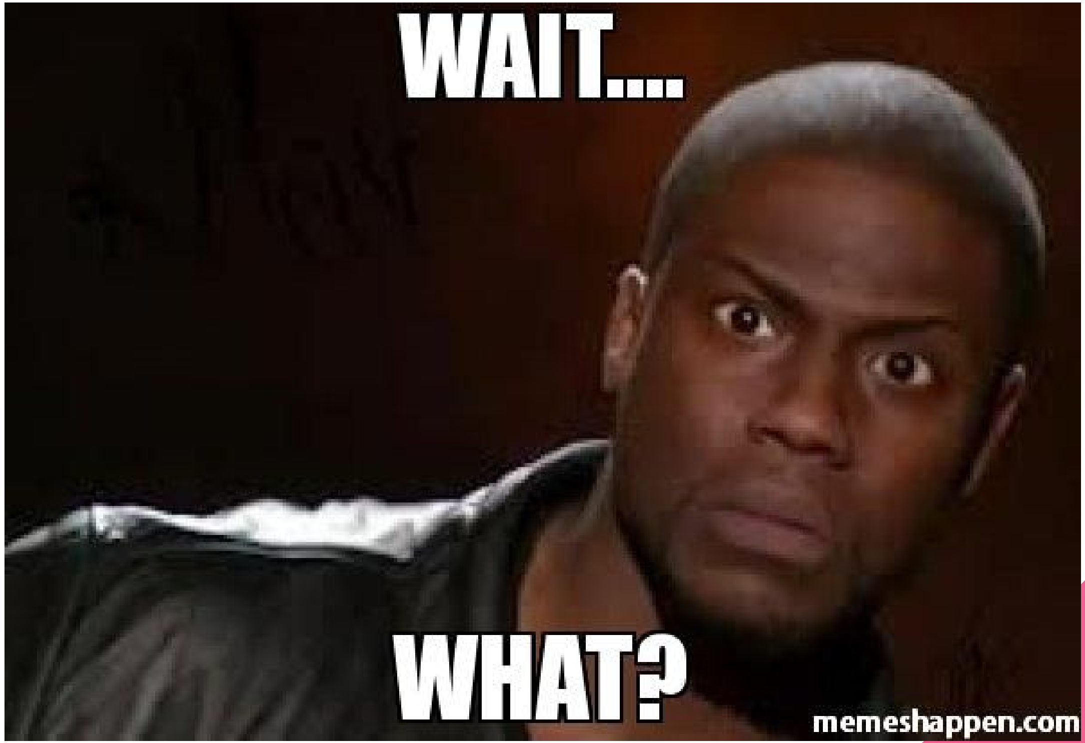
CS 130: Software Engineering

Lab 1C: Week 5 Discussion

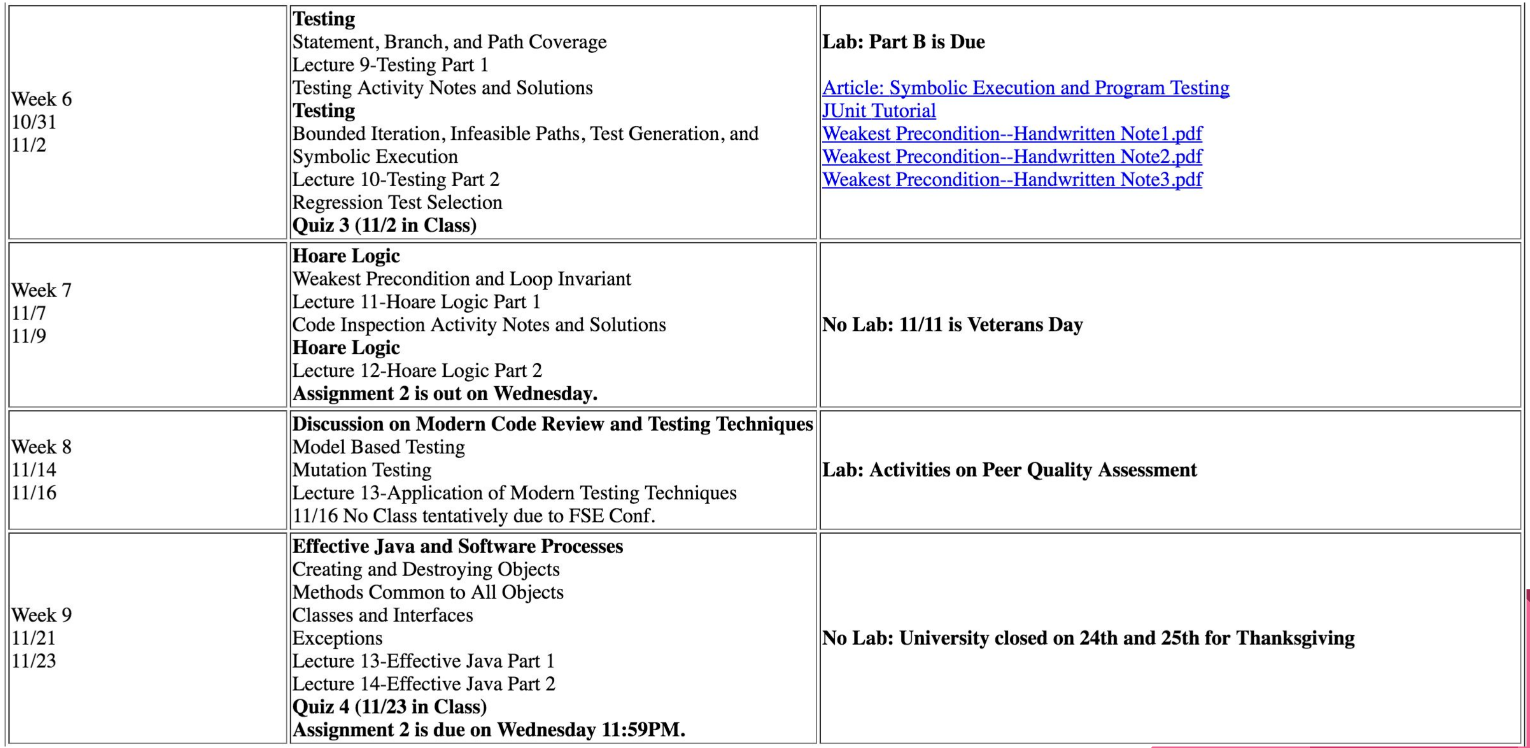
Agenda

- Group Project: Part C
- Midterm
- Testing

Group Project: Part C



Whoa

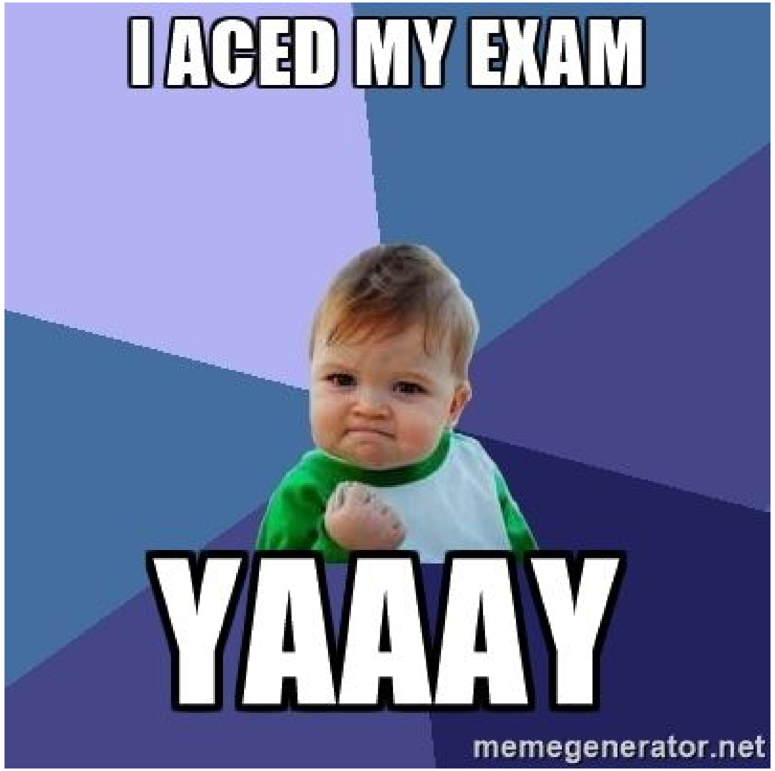


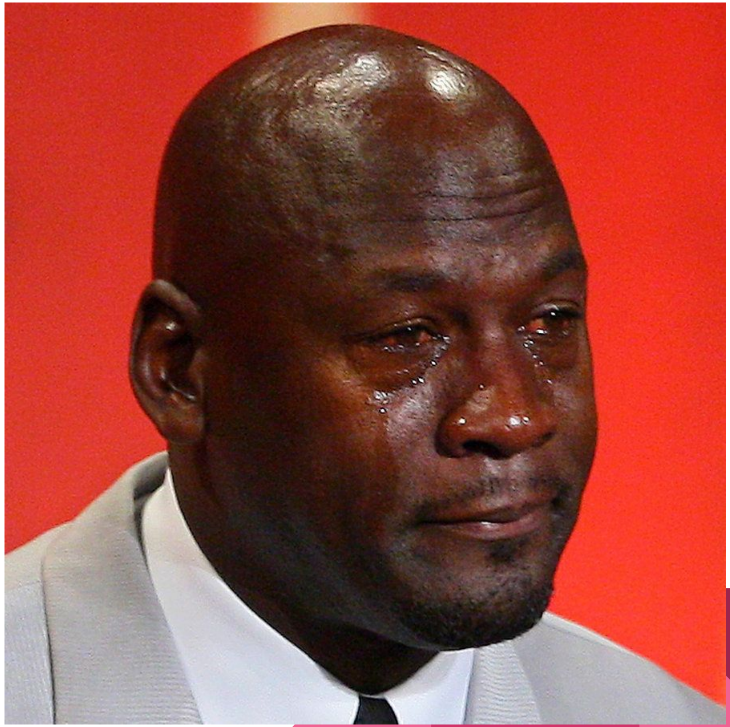
Week 6 10/31 11/2	**Testing** Statement, Branch, and Path Coverage Lecture 9-Testing Part 1 Testing Activity Notes and Solutions **Testing** Bounded Iteration, Infeasible Paths, Test Generation, and Symbolic Execution Lecture 10-Testing Part 2 Regression Test Selection **Quiz 3 (11/2 in Class)**	**Lab: Part B is Due** Article: Symbolic Execution and Program Testing JUnit Tutorial Weakest Precondition--Handwritten Note1.pdf Weakest Precondition--Handwritten Note2.pdf Weakest Precondition--Handwritten Note3.pdf
Week 7 11/7 11/9	**Hoare Logic** Weakest Precondition and Loop Invariant Lecture 11-Hoare Logic Part 1 Code Inspection Activity Notes and Solutions **Hoare Logic** Lecture 12-Hoare Logic Part 2 **Assignment 2 is out on Wednesday.**	**No Lab: 11/11 is Veterans Day**
Week 8 11/14 11/16	**Discussion on Modern Code Review and Testing Techniques** Model Based Testing Mutation Testing Lecture 13-Application of Modern Testing Techniques 11/16 No Class tentatively due to FSE Conf.	**Lab: Activities on Peer Quality Assessment**
Week 9 11/21 11/23	**Effective Java and Software Processes** Creating and Destroying Objects Methods Common to All Objects Classes and Interfaces Exceptions Lecture 13-Effective Java Part 1 Lecture 14-Effective Java Part 2 **Quiz 4 (11/23 in Class)** **Assignment 2 is due on Wednesday 11:59PM.**	**No Lab: University closed on 24th and 25th for Thanksgiving**

Group Project: Part C

- **18%** of your final grade!
- Final Presentation
  - Project should be completed and fully functioning
- Report
- Presentation
- YouTube Video

Midterm

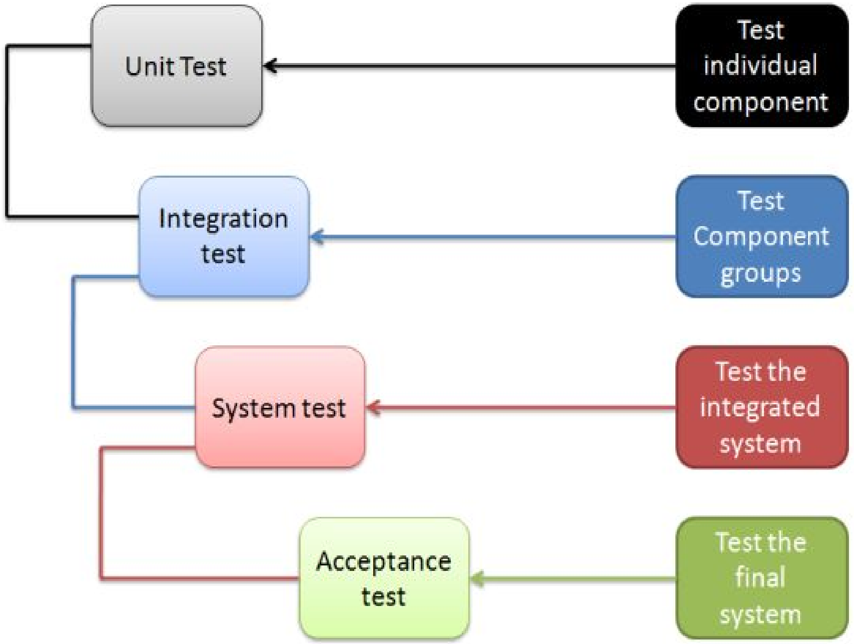




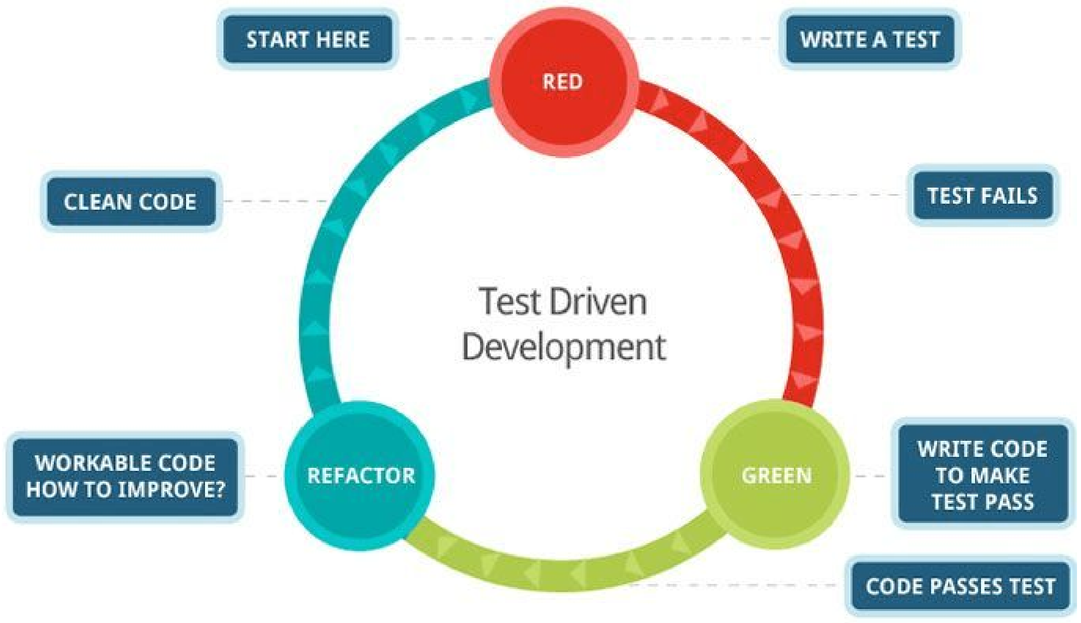
Testing

Testing

- Unit Testing
  - Function, method, class, module
  - Test each unit separately
- Integration Testing
  - Test interaction between different components
- System Testing
  - Test the entire system
- Acceptance Testing
  - test with respect to user needs before release



Test-Driven Development (TDD)



A Perspective on TDD

- ❏ Write tests first
- ❏ Keep the unit small
- ❏ Each test fails initially
- ❏ Write code necessary to pass test
- ❏ Iterative process

- ❏ Testability
- ❏ Simpler Code
- ❏ Gives confidence
- ❏ Requires more discipline
- ❏ Validates your design
- ❏ Provides rapid feedback

The Three Laws of TDD

- You can’t write production code until you have written a failing test
- You can’t write more of a unit test than is sufficient to fail
- You can’t write more production code than is sufficient to pass the current failing tests

Fundamental Principles

- Think about what you want to do first
- Follow the TDD cycle and the three laws
- Never write a new functionality without a failing test
- Continuously make small, incremental changes
- Run tests immediately after each change commit

JUnit

- JUnit is an open-source Java testing framework
  - write and run repeatable automated tests
- Features
  - Assertions for testing expected results, e.g., *assertEquals, assertNull, assertTrue, etc.*
  - Expression annotations in JUnit 4, @Test, @TimeOut, @Teardown
  - Easy to share test data
  - Test suites for easily organizing and running tests
  - Graphical and textual test runners, e.g., stand-alone java program, IDE plugins

Writing tests for JUnit

- Annotate your test cases with @Test
- Each test case checks a condition using assert methods provided by JUnit
  - for ease of automation and generating reports
- All of the test cases return void

```
public class Client{
        public int add(int a, int b){
                return a + b;
        }
}
```

```
import org.junit.Test;
import static org.junit.Assert.*;

public class ClientTest{
        @Test
        public void testAdd(){
                Client c = new Client();
                assertEquals(5, c.add(2,3));
        }
}
```

Basic Annotations

- @BeforeClass
  - Run once before any of the test methods in the class
  - e.g., Database connection
- @AfterClass
  - Run once after all the tests in the class have been run
  - e.g., close connection
- @Before
  - Run before each test method
  - e.g., create one object and share for all test cases in the same class
- @After – Run after each test method
  - Run before each test method
  - e.g., create one object and share for all test cases in the same class

More Annotations

- Test expected exceptions
  - @Test with optional ‘expected’ attribute
  - Try-catch and always fail()
  - @Rule ExpectedException
- @Rule intercepts test methods to do stuff before and after test execution
  - Similar to @Before and @After
  - Predefined rules, e.g., ExpectedException, TemporaryFolder, etc.
  - Create your own rule by implementing the TestRule interface
- @Timeout checks the program performance

Android UI Test

- Espresso
  - Find a View
  - Perform an action
  - Inspect the result
- https://www.youtube.com/watch?v=kL3MCQV2M2s

Selenium

- Selenium is a testing framework for web applications
    - Simply, Selenium automates browsers!
    - Provides an IDE to record and replay user actions in a browser, e.g., click links (demo)
    - Also provides a WebDriver and APIs to write tests from popular programming languages