

## TCSS 360: SOFTWARE DEVELOPMENT AND QUALITY ASSURANCE

# Software testing II

Wes J. Lloyd  
 Institute of Technology  
 University of Washington - Tacoma



## COURSE PLANNING

- Project deliverable #1 – Today Wednesday February 15
- Project collaborative meeting – Today February 15
  - One group representative
  - Define common data sharing APIs
  - Share service endpoints (Heroku)
- Move to private project repositories – phase II
  - Free student accounts include unlimited private repos
  - <https://education.github.com/>

February 15, 2017
TCSS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma
L11.2

## OBJECTIVES

- Continue chapter 8:  
 Software Testing – Test Driven Development
- Test driven development (TDD)
- Junit testing
  - → Mock objects
  - → Test coverage

February 15, 2017
TCSS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma
L11.3

## MOCK TESTING

- How do we perform unit tests on classes which can't run without instances of other classes?
- For example: a database connection
- Use mock objects
  - Simulated objects
  - Mimic behavior or real objects
  - Support unit testing of middleware classes
  - Used to test behavior of other objects

February 15, 2017
TCSS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma
L11.4

## SPOCK

- Testing and specification framework for Java and Groovy
- Inspired by Junit, Rspec (shown in ch. 8)
- Write specifications
  - Describe expected features exhibited by system of interest (SOI)
  - SOI can be single class, or entire application
  - SOI is also called the system under specification (SUS)
- In SPOCK you write specifications which describe the features to test.
- A specification is to SPOCK, what a unit test is to Junit

February 15, 2017
TCSS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma
L11.5

## SPOCK VS. JUNIT COMPARISON

Spock	JUnit
Specification	Test class
setup()	@Before
cleanup()	@After
setupSpec()	@BeforeClass
cleanupSpec()	@AfterClass
Feature	Test
Feature method	Test method
Data-driven feature	Theory
Condition	Assertion
Exception condition	@Test(expected=...)
Interaction	Mock expectation (e.g. in Mockito)

February 15, 2017
TCSS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma
L11.6

### MOCK OBJECTS - EXAMPLE

- Consider unit testing for a class hierarchy:
  - cook ← waiter ← customer
- Testing cook is easy since it has no dependencies
- cook ← test\_driver
- We can write simple tests to exercise the methods of cook
- Simpler to data objects (e.g. user class)

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.7

### MOCK OBJECTS EXAMPLE - 2

- cook ← waiter ← test\_driver
  - Testing waiter this way depends on proper behavior of the cook
  - Testing this way is not really unit testing
  - More like integration testing
- How can we test the waiter class independent of the cook class?
- Use Mock Objects
- Isolate the component under test: "waiter"

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.8

### MOCK OBJECTS

- Like puppets
- Behavior completely controlled by programmer in test script
- Allows isolation of unit tests from collaborator classes
- Supports keeping tests **Independent**

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.9

### MOCK TESTING EXAMPLE - 3

- Three types of cooks to test waiter:
- **A fake cook:**
  - Test the waiter using a fake cook, someone who pretends to be a cook by using frozen dinners and a microwave
  - (substitute class)
- **A stub cook**
  - Test the waiter using a hot dog vendor, that always cooks hot dogs no matter what food is ordered
  - (fixed implementation class - limits testing scope)
- **A mock cook**
  - An undercover cop follows a script and pretends to be a cook in a sting operation

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.10

### MOCK TESTING EXAMPLE - 4

- With mock objects we can write tests to cover more possibilities than with a stub or fake object
- Time is not spent writing stubs or fake test objects
- Mock object testing can allow you to write tests before code
- You can mock the classes you plan to write, and later remove the mocks once the classes are available
- This feature is a key to supporting Test Driven Development (TDD)


February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.11

### JAVA TESTING: SPOCK

- Examples
- Maven Pom.xml requirements
  - Required plugins
  - Dependencies
- For more information:
  - <http://spockframework.org/>
  - <https://martinfowler.com/articles/mocksArentStubs.html>

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.12

## SOFTWARE TESTING: PLAN AND DOCUMENT PERSPECTIVE



L11.13

January 25, 2017

## TDD AND F.I.R.S.T.

- **Implicit requirements:** explicit requirements lead to additional implicit (unspecified) requirements
  - Implicit requirement implementation required to fulfill explicit requirements
  - Often discovered through written tests
- **Fast and Repeatable**
  - Use stub classes to mimic external dependencies to improve test performance
  - Stub: databases, webservices, etc.

L11.14

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma

## TEST COVERAGE

- How much testing is enough?
  - As much as you can do before the shipping deadline
- Code-to-test ratio (non-comment lines of code)
  - Production < 1 : more test code than app code
- Types of tests
  - Integration/system tests
  - Functional tests

L11.15

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma

## TEST COVERAGE: SAMPLE CODE

```

1 public class MyClass
2 {
3     int w;
4     public void food(int x, int y, int z) {
5         if (x)
6             if (y && z)
7                 bar(0);
8         else
9             bar(1);
10    }
11    public void bar(int x) {
12        w = x;
13    }
14 }
    
```

L11.16

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma

## LEVELS OF TEST COVERAGE

- **Code coverage**
  - Percentage of code exercised by testing
  - Not always straightforward to measure
  - Level are increasingly difficult to achieve
- **Method coverage (S0)** – Is every method executed at least once?
  - > Must call foo and bar at least once
- **Call coverage (S1)** – Has every method been called from every place it could be called?
  - > Requires calling bar from line 7 & 9

L11.17

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma

## LEVELS OF TEST COVERAGE - 2

- **Statement coverage (C0)** – Is every statement of the source code executed at least once by the test suite?
  - > Count each branch of a conditional as a single statement: must call with x=true, y=false
- **Branch coverage (C1)** – Has every branch been taken at least once?
  - > Must call foo with x=true&false and also y&z so both branches execute
- **Subset - Decision coverage:** Each subexpression that affects a conditional expression must be tested with both a true and false value. y&z where y=false, y&z where z=false
- **Path coverage (C2)** – Has every possible route through the code been executed?
  - > If x,y,z are booleans, then there are 8 possible paths

L11.18

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017]  
 Institute of Technology, University of Washington - Tacoma

### TEST COVERAGE - 3

- Achieving statement coverage (C0) is straightforward
- Achieving branch coverage (C1) is more difficult:  
Test cases must ensure each branch is taken at least once in each direction
- Path coverage (C2) is very difficult:  
Not all experts agree on the value of path coverage
- Does high test coverage imply a well-tested application?
- Coverage says nothing about the quality of the tests.
- Low coverage implies a poorly tested application

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.19

### TEST COVERAGE - 4

- Code coverage statistics
  - Highlight under-tested, untested parts of application code
  - Show the overall comprehensiveness of test suite
- A number of tools exist to evaluate code coverage of tests
  - For Java see Jcov, Clover, EMMA, Serenity

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.20

### CLASSIC TESTING

- Section 8.8
- Control flow coverage
  - Represent program as control flow graph
  - Test cases visit nodes of the graph
  - Fraction of all nodes visited = test coverage (%)
- Define-use coverage
  - What percentage of variable assignment instructions are visited?
  - What percentage of variable read instructions are visited?

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.21

### CLASSIC TESTING - 2

- Black-box testing
  - Tests based solely on APIs and external interfaces
  - Black box test of a hash function implementation is limited to provided key/value pairs
  - We don't know what edge/boundary conditions to tests...
  - What is the bucket size?
- White-box (glass box) testing
  - Tests reflect knowledge of software implementation
  - Tests can exploit knowledge of the implementation
  - Example: hash function test - since we know the hash function, construct worst case tests that produce many hash collisions
  - Boundary conditions: parameter values likely to exercise different parts of the code

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.22

### CLASSIC TESTING - 3

- Mutation testing
  - Test automation technique which generates small but syntactically legal changes to program code which should produce test failures
  - If we randomly mutate program code and no test fails, we don't have enough test coverage - (or its an odd program)
- Fuzz testing
  - Throw random data at a program to try to cause failure
  - ~25% of Unix utilities will crash by fuzz testing
  - Microsoft finds 20-25% of bugs using fuzz testing
  - Dumb fuzz: totally random data
  - Smart fuzz: data includes knowledge about the application

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.23

### PLAN AND DOCUMENT PERSPECTIVE

- Composing unit tests for integration testing:
  - Top-down integration
  - Bottom-up integration
  - Sandwich integration
- Formal testing methods - use formal specification to prove the behavior of the code using mathematical proofs
  - Automatic theorem proving
  - Model checking
    - Verify selected properties via exhaustive search of all possible states a system could enter during execution

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.24

### AGILE VS PLAN AND DOCUMENT TESTING

Tasks	Plan and Document	Agile
Test Plan and Documentation	Software Test Document such as IEEE Standard 829-2008	User stories
Order of coding and testing	<ol style="list-style-type: none"> <li>Code units</li> <li>Unit tests</li> <li>Module test</li> <li>Integration test</li> <li>System test</li> <li>Acceptance test</li> </ol>	<ol style="list-style-type: none"> <li>Acceptance test</li> <li>Integration test</li> <li>Module test</li> <li>Unit test</li> <li>Code units</li> </ol>
Testers	Developers (unit tests), QA testers (module, integration, system, and acceptance)	Developers (all) Customer (acceptance)
When Testing Stops	Company policy (e.g. statement coverage, happy or sad user inputs)	All tests pass (green)

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.25

### FALLACIES / PITFALLS

- **Fallacy:** 100% test coverage with all tests passing means no bugs
- **Pitfall:** Dogmatically insisting on 100% test coverage all passing (green) before you ship
- **Fallacy:** You don't need much test code to be confident in the application
- **Pitfall:** Relying too heavily on just one kind of test (unit, functional, integration)
- **Pitfall:** Under tested integration points due to over-stubbing

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.26

### FALLACIES / PITFALLS - 2

- **Pitfall:** Mock Trainwrecks: too many mock objects required for unit testing due to class coupling (even circular dependencies)
- **Pitfall:** Inadvertently creating dependencies regarding the order in which specs (tests) are run
- **Pitfall:** Forgetting to re-prepare the test database when the schema changes


February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.27

### TUTORIAL 4

- **Java data persistence w/ postgresSQL and heroku...**
- [http://faculty.washington.edu/wlloyd/courses/tcss360/tutorials/TCCS360\\_w2017\\_Tutorial\\_4.pdf](http://faculty.washington.edu/wlloyd/courses/tcss360/tutorials/TCCS360_w2017_Tutorial_4.pdf)

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.28

# QUESTIONS



February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L23.29

### THE CONTENTS OF THIS SLIDE SET ARE BASED ON THE FOLLOWING REFERENCES:

- Armando Fox, David Patterson, *Engineering Software As A Service: An Agile Approach Using Cloud Computing*, 1<sup>st</sup> edition (v1.2.1), Strawberry Canyon LLC., 2016. ISBN-13: 978-0984881246. [Chapter 8]

February 15, 2017 TCCS360: Software Development and Quality Assurance [Winter 2017] Institute of Technology, University of Washington - Tacoma L11.30

### SPOCK SPEC SAMPLE

```
def "HashMap accepts null key"() {
  setup:
  def map = new HashMap()

  when:
  map.put(null, "elem")

  then:
  notThrown(NullPointerException)
}
```

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.31

### SPOCK SPEC SAMPLE - 2

```
def "events are published to all subscribers"() {
  def subscriber1 = Mock(Subscriber)
  def subscriber2 = Mock(Subscriber)
  def publisher = new Publisher()
  publisher.add(subscriber1)
  publisher.add(subscriber2)

  when:
  publisher.fire("event")

  then:
  1 * subscriber1.receive("event")
  1 * subscriber2.receive("event")
}
```

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.32

### SPOCK SAMPLE SPEC - 3

```
def "offered PC matches preferred configuration"() {
  when:
  def pc = shop.buyPc()

  then:
  with(pc) {
    vendor == "Sunny"
    clockRate >= 2333
    ram >= 406
    os == "Linux"
  }
}
```

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.33

### SPOCK SAMPLE SPEC - 4

```
class MathSpec extends Specification {
  def "maximum of two numbers"(int a, int b, int c)
  {
    expect:
    Math.max(a, b) == c

    where:
    a | b | c
    1 | 3 | 3
    7 | 4 | 7
    0 | 0 | 0
  }
}
```

February 15, 2017

TCCS360: Software Development and Quality Assurance [Winter 2017]  
Institute of Technology, University of Washington - Tacoma

L11.34