

TCSS 360: SOFTWARE DEVELOPMENT AND QUALITY ASSURANCE

Software testing

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



COURSE PLANNING

- Project deliverable #1 – Due Wednesday February 15
- Class-wide project collaborative meeting: Wednesday February 15
 - One group representative
 - Define common data sharing APIs
 - Exchange project wikis
 - Share service endpoints

February 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma L10.2

OBJECTIVES

- From chapter 8:
Software Testing – Test Driven Development
- Test driven development (TDD)
- JUnit testing
- Mock objects
- Test coverage

February 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma L10.3

SOFTWARE TESTING

- Verification: building the right thing
- Quality assurance (QA) team
 - Engineers tasked with writing tests, manual testing, writing bug reports
 - Agile/DEVOPS: expects QA team to be developers
 - Developers = Testers

February 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma L10.4

TEST DRIVEN DEVELOPMENT (TDD)

- Construct software from a test-centric perspective
 - Also called test-first development
 - Test code written before functional code
 - Improves code readability, maintainability
 - Testable code, tends to be good code
 - Code is well tested, more modular, easier to read

February 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma L10.5

AGILE: TEST AUTOMATION

- Agile: DEVOPS team writes tests
- Tests are automatically applied using tools
- Tests continuously run/applied via automated build system which builds code and executes test suites

February 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma L10.6

F.I.R.S.T. TESTING

- **Fast:** should be easy and quick to run a subset of test cases for various coding tasks
- **Independent:** No test should rely on preconditions created by other tests. Support prioritization of running subsets of tests to cover only recent code changes
- **Repeatable:** tests should not depend on external factors: today's date, magic constants
- **Self-checking:** test results can be determined by the system. Computer can infer pass/fail status without human intervention
- **Timely:** Tests are created/updated at the same time as the code being testing. Test creation ideally immediately before code creation.

February 13, 2017

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

L10.7

JAVA – JUNIT TESTING

- Can write test methods for every method of a class
- A Junit test class will collect the test methods for a given class in the system
- Set up and tear down methods
 - Help configure working environment so preconditions are in place for test runs
 - Example: parameterize a database connection
- Java annotations are used to mark up Junit classes

February 13, 2017

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

L10.8

JUNIT ANNOTATIONS

- **@BeforeClass, @AfterClass**
 - Runs once before and after entire all tests
 - Good for preparing a database connection, initializing dependencies
- **@Before, @After**
 - Execute before and after each individual junit test is run
- **@Test**
 - Annotates each test method
 - Identifies individual test cases

February 13, 2017

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

L10.9

JUNIT TESTING - 3

- Assert methods:
 - `assertArrayEquals()`
 - `assertEquals()`
 - `assertFalse()`
 - `assertNotEquals()`
 - `assertNotNull()`
 - `assertNotNull()`
 - `assertSame()`
 - `assertThat()`
 - `assertTrue()`
- Parameter order: expected value, actual value
- Optional: First parameter can be a string message that is output on failure

February 13, 2017

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

L10.10

JUNIT TEST SUITE

- Test suites support writing multi-class tests
- Integration tests
- `@RunWith(Suite.class)`
- `@Suite.SuiteClasses({})`

February 13, 2017

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

L10.11

PARAMETERIZED TESTS

- Run same test over and over using different values
- Annotate test case with `@RunWith(Parameterized.class)`
- Create a public static method annotated with `@Parameters` that returns a Collection of Objects (as an Array) as test data
- Create a public constructor that takes in what is equivalent to one "row" of test data.
- Create an instance variable for each "column" of test data
- Create your test case(s) using the instance variables as the source of the test data

February 13, 2017

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

L10.12

EXCEPTION TESTING

```
@Test(expected = IndexOutOfBoundsException.class)
public void empty() {
    new ArrayList<Object>().get(0);
}

@Test
public void testExceptionMessage() {
    try
    {
        new ArrayList<Object>().get(0);
        fail("Expected an IndexOutOfBoundsException to be thrown");
    }
    catch (IndexOutOfBoundsException anIndexOutOfBoundsException)
    {
        assertThat(anIndexOutOfBoundsException.getMessage(),
            is("Index: 0, Size: 0"));
    }
}
```

February 13, 2017

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

L10.13

EXCEPTION TESTING - 2

```
@Rule
public ExpectedException thrown = ExpectedException.none();

@Test
public void shouldTestExceptionMessage() throws
    IndexOutOfBoundsException
{
    List<Object> list = new ArrayList<Object>();
    thrown.expect(IndexOutOfBoundsException.class);
    thrown.expectMessage("Index: 0, Size: 0");
    list.get(0); // execution will never get past this line
}
```

February 13, 2017

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

L10.14

OTHER JUNIT ANNOTATIONS

- Ignore a test
- @Ignore("This is ignored test is ignored")
- Specify a timeout value
- @Test(timeout=1000)
- Specify the execution sequence of tests
- @FixOrderMethod(MethodSorters.NAME_ASCENDING)
- @RunWith(Theories.class)
- @DataPoint (marks variables that are data points)
- @Theory
 - Test functionality against subset of infinite data
 - Run a theory against datapoints, mimics scientific theorem
 - Try to prove a theory, theory passes if all datapoints fit the theorem

February 13, 2017

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

L10.15

JUNIT EXAMPLES

- JUnit tests, parameterized tests, suite tests
- Netbeans: direct Junit support/integration
- Automatic test generation!
- Integrating into a maven project
- Pom.xml file requirements
 - Maven-surefire-plugin
 - Specify test class naming conventions
 - Dependencies
- JUnit jar file
- Scope = test

February 13, 2017

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

L10.16

TEST DRIVE DEVELOPMENT: RED-GREEN-REFACTOR

- Before writing code, write test for one aspect of the behavior the new code will have
 - Writing the test forces thinking about how new code will behave and interact if it did exist
- RED** step: run the test, verify it fails because code is implemented yet
- GREEN** step: write the simplest code to make the test pass without breaking any existing tests
- REFACTOR** step: Refactor implementation code or test code. Change structure to eliminate redundancy, repetition, or other ugliness
 - Tests ensure refactoring doesn't introduce bugs.

February 13, 2017

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

L10.17

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 13, 2017

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

L10.18



Groovier testing with Spock
 Rob Fletcher
 git clone http://github.com/robletcher/ggug-spock-examples

February 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
 Institute of Technology, University of Washington - Tacoma L10.19

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 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
 Institute of Technology, University of Washington - Tacoma L10.20

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 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
 Institute of Technology, University of Washington - Tacoma L10.21

FIXTURES

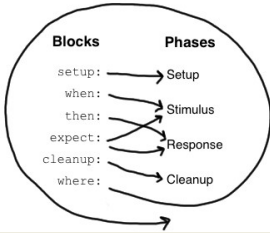


- Fixture: snapshot of a system under test
- Fox and Patterson: "a fixed state used as a baseline for one or more tests"
- Includes a set of static objects used to instrument a test
- Fixture methods: (for setup/teardown)
 - def setup() {} // run before every feature method
 - def cleanup() {} // run after every feature method
 - def setupSpec() {} // run before the first feature method
 - def cleanupSpec() {} // run after the last feature method
- Uses to set up fixtures for the features (tests)
- Fixtures here are used like junit @Before, @After, etc.

February 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
 Institute of Technology, University of Washington - Tacoma L10.22

SPOCK

- Conditions
 - Like assertions in Junit without the use of the "assertion" descriptor
- Blocks
 - Maps flow of the test
 - setup:
 - when:
 - then:
 - expect:
 - cleanup:
 - where:



February 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
 Institute of Technology, University of Washington - Tacoma L10.23

SPOCK SPEC SAMPLE

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

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

    then:
        notThrown(NullPointerException)
}
```

February 13, 2017 TCSS360: Software Development and Quality Assurance [Winter 2017]
 Institute of Technology, University of Washington - Tacoma L10.24

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 13, 2017

TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma

L10.25

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 13, 2017

TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma

L10.26

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 13, 2017

TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma

L10.27

QUESTIONS



February 13, 2017

TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma

L10.28

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*, 1st edition (v1.2.1), Strawberry Canyon LLC., 2016. ISBN-13: 978-0984881246. [Chapter 8]

February 13, 2017

TCSS360: Software Development and Quality Assurance [Winter 2017]
Institute of Technology, University of Washington - Tacoma

L10.29