

Objectives

- Testing
- Unit testing
- JUnit Framework
 - In Eclipse

"I don't think anybody tests enough of anything."
From *A Conversation with James Gosling*

Oct 19, 2011

Sprenkle - CSCI209

1

Review

- Start Eclipse for later exercise
- Describe and compare the two software development models we discussed
 - Think of the model shapes
- How can we categorize prototypes?
 - What are their characteristics?

Oct 19, 2011

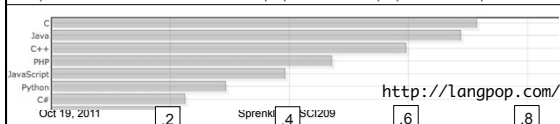
Sprenkle - CSCI209

2

Programming Language Popularity

Position Oct 2011	Position Oct 2010	Delta in Position	Programming Language	Ratings Oct 2011	Delta Oct 2010	Status
1	1	=	Java	17.913%	-0.25%	A
2	2	=	C	17.707%	+0.63%	A
3	3	=	C++	9.072%	-0.73%	A
4	4	=	PHP	6.818%	-1.51%	A
5	6	↑	C#	6.723%	+1.76%	A
6	8	↑↑	Objective-C	6.245%	+2.54%	A
7	5	↓↓	(Visual) Basic	4.549%	-1.10%	A
8	7	↓	Python	3.944%	-0.92%	A

<http://www.tiobe.com/index.php/content/paperinfo/tpci/>



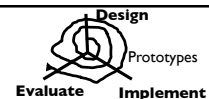
SOFTWARE TESTING PROCESS

Oct 19, 2011

Sprenkle - CSCI209

5

Spiral Model Steps



- Design a {method, class, package}
- Implement the {method, class, package}
- Test the {method, class, package} ←
- Fix the {method, class, package}
- Deploy the {method, class, package}
- Get feedback
 - Probably will require modifications to design
- Repeat

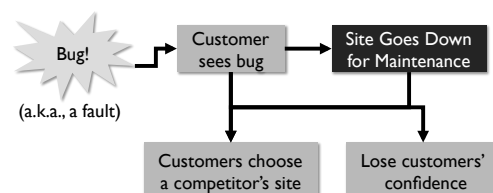
Oct 17, 2011

Sprenkle - CSCI209

4

Why Test Programs?

- Consider an online bookstore



Oct 19, 2011

Sprenkle - CSCI209

6

Microsoft Windows Vista Testing

- Beyond their internal testing ...
 - 5 million people beta tested
 - 60+ years of performance testing
 - 1 Billion+ Office 2007 sessions
- Still, users found correctness, stability, robustness, and security bugs

Oct 19, 2011

Sprenkle - CSCI209

7

Type 1 Bugs: Compile-Time



- Syntax errors
 - Missing semicolon, parentheses
- Compiler notifies of error
- Cheap, easy to fix

Oct 19, 2011

Sprenkle - CSCI209

8

Type 2 Bugs: Run-Time



- Usually logic errors
- Expensive to locate, fix

Oct 19, 2011

Sprenkle - CSCI209

9

Aside: Objections to "Bug" Terminology

- "Bug"
 - Sounds like it's just an annoyance
 - Can simply swat away
 - Minimizes potential problems
 - Hides programmer's responsibility
- Alternative terms
 - **Defect**
 - **Fault**

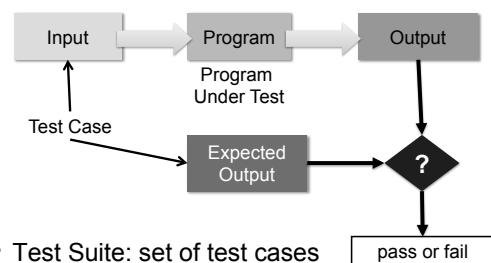


Oct 19, 2011

Sprenkle - CSCI209

10

Software Testing Process



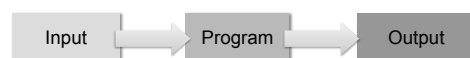
- Test Suite: set of test cases

Oct 19, 2011

Sprenkle - CSCI209

11

Software Testing Process



- Tester plays devil's advocate
 - **Hopes** to reveal problems in the program using "good" test cases
 - Better tester finds than a customer!

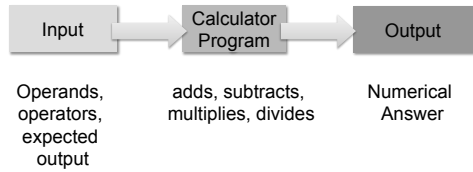
How is **testing** different from **debugging**?

Oct 19, 2011

Sprenkle - CSCI209

12

How Would You Test a Calculator Program?



- What test cases: input and expected output?

Oct 19, 2011

Sprenkle - CSCI209

13

Example Test Cases for Calculator Program

- Basic Functionality
 - Addition
 - Subtraction
 - Multiplication
 - Division
 - Order of operations
- Invalid Input
 - Letters, not-operation characters (&,\$, ...)
- “Tricky” Cases
 - Divide by 0
 - Negative Numbers
 - Long sequences of operands, operators
 - VERY large, VERY small numbers

Oct 19, 2011

Sprenkle - CSCI209

14

Types of Testing

- Black-box testing
- White-box testing
- Non-functional testing
- Acceptance testing

Ideas or definitions of any of these?

Oct 19, 2011

Sprenkle - CSCI209

15

Types of Testing

- Black-box testing
 - Test *functionality* (e.g., the calculator)
 - No knowledge of the code
 - Examples of testing: boundary values
- White-box testing
 - Have access to code
 - **Goal:** execute *all* code
- Non-functional testing
 - Performance testing
 - Usability testing (HCI)
 - Security testing
 - Internationalization, localization
- Acceptance testing
 - Customer tests to decide if accepts product

Oct 19, 2011

Sprenkle - CSCI209

16

Levels of Testing

- Unit
 - Tests minimal software component, in isolation
 - For us, Class-level testing
 - Web: Web pages (Http Request)
- Integration
 - Tests interfaces & interaction of classes
- System
 - Tests that completely integrated system meets requirements
- System Integration
 - Test system works with other systems, e.g., third-party systems

Cost increases

Oct 19, 2011

Sprenkle - CSCI209

17

UNIT TESTING

Oct 19, 2011

Sprenkle - CSCI209

18

Why Unit Test?

- Verify code works as intended in isolation
- Find defects **early** in development
 - Easier to test small pieces
 - Less cost than at later stages

Oct 19, 2011

Sprenkle - CSCI209

19

Why Unit Test?

- Verify code works as intended in isolation
- Find defects **early** in development
 - Easier to test small pieces
 - Less cost than at later stages
- As application evolves, new code is more likely to break existing code
 - Suite of (small) test cases to run after code changes
 - Also called **regression** testing

Oct 19, 2011

Sprenkle - CSCI209

20

Some Approaches to Testing Methods

- Typical case
 - Test typical values of input/parameters
- Boundary conditions
 - Test at boundaries of input/parameters
 - Many bugs live "in corners"
- Parameter validation
 - Verify that parameter and object bounds are documented and checked
 - Example: pre-condition that parameter isn't null

➡ All black-box testing approaches

Oct 19, 2011

Sprenkle - CSCI209

21

Another Use of Unit Testing: Test-Driven Development

- A development style, evolved from Extreme Programming
- Idea: write tests first *without code bias*
- The Process:
 1. Write the tests that code/new functionality should pass
 - Like a specification for the code (pre/post conditions)
 - All tests will initially *fail*
 2. Write the code and verify that it passes test cases
 - Know you're done coding when you pass **all** tests

How do you know you're "done" in traditional development?

What assumption does this make?

Oct 19, 2011

Sprenkle - CSCI209

22

Software Testing Issues

- How should you test? How often?
 - Code may change frequently
 - Code may depend on others' code
 - A lot of code to validate
- How do you know that an output is correct?
 - Complex output
 - Human judgment?
- What caused a code failure?

➡ Need a *systematic, automated, repeatable* approach

Oct 19, 2011

Sprenkle - CSCI209

23

Characteristics of Good Unit Testing

- **Automatic**
- **Thorough**
- **Repeatable**
- **Independent**

Why are these characteristics of good (unit) testing?

Oct 19, 2011

Sprenkle - CSCI209

24

Characteristics of Good Unit Testing

- **Automatic**
 - Since unit testing is done frequently, don't want humans slowing the process down
 - Automate executing test cases and evaluating results
 - Input: in test itself or from a file
- **Thorough**
 - Covers all code/functionality/cases
- **Repeatable**
 - Reproduce results (correct, failures)
- **Independent**
 - Test cases are independent from each other
 - Easier to trace fault to code

Oct 19, 2011

Sprenkle - CSCI209

25

JUNIT

Oct 19, 2011

Sprenkle - CSCI209

26

JUnit Framework

- A framework for unit testing Java programs
 - Supported by Eclipse and other IDEs
 - Developed by Erich Gamma and Kent Beck
- Functionality
 - Write tests
 - Validate output, automatically
 - Automate execution of test suites
 - Display pass/fail results of test execution
 - Stack trace where fails
 - Organize tests, separate from code
- But, you still need to come up with the tests!



Oct 19, 2011

Sprenkle - CSCI209

27

Aside: Framework

A **framework** is a basic conceptual structure used to solve or address complex issues.

This very broad definition has allowed the term to be used as a buzzword, especially in a software context.

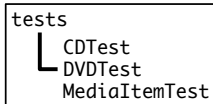
Oct 19, 2011

Sprenkle - CSCI209

28

Testing with JUnit

- Typical organization:
 - Set of testing classes
 - Testing classes packaged together in a **tests** package
 - Separate package from code testing
- A test class typically
 - Focuses on a specific class
 - Contains methods, each of which represents another test of the class



Oct 19, 2011

Sprenkle - CSCI209

29

Structure of a JUnit Test

1. Set up the test case (optional)
 - Example: Creating objects
2. Exercise the code under test
3. Verify the correctness of the results
4. Teardown (optional)
 - Example: reclaim created objects

Oct 19, 2011

Sprenkle - CSCI209

30

Annotations

- Testing in JUnit 4: uses **annotations**
- Provide data about a program that is not part of program itself
- Have no direct effect on operation of the code
- Example uses:
 - **@Override**: method declaration is intended to override a method declaration in parent class
 - If method does not override parent class method, compiler generates error message
 - Information for the compiler to suppress warnings (**@SuppressWarnings**)

Oct 19, 2011

Sprenkle - CSCI209

31

Tests are Methods

- Mark your testing method with **@Test**
 - From `org.junit.Test`

```
public class CalculatorTest {
    @Test
    public void addTest() {
        ...
    }
}
```

Class for testing the Calculator class

A method to test the "add" functionality

- Convention: Method name describes what you're testing

Oct 19, 2011

Sprenkle - CSCI209

32

Assert Methods

- Variety of assert methods available
- If fail, throw an exception
- All **static void**
- Example:


```
assertEquals(Object expected, Object actual)
```

```
@Test
public void addTest() {
    ...
    assertEquals(4, calculator.add(3, 1));
}
```

Oct 19, 2011

Sprenkle - CSCI209

33

Assert Methods

- To use asserts, need *static* import:


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

 - `static` allows us to not have to use classname
- More examples
 - `assertTrue(boolean condition)`
 - `assertSame(Object expected, Object actual)`
 - Refer to same object
 - `assertEquals(double expected, double actual, double delta)`

Oct 19, 2011

Sprenkle - CSCI209

34

Example Uses of Assert Methods

```
@Test
public void testEmptyCollection() {
    Collection collection = new ArrayList();
    assertTrue(collection.isEmpty());
}

@Test
public void testPI() {
    final double ERROR_TOLERANCE = .01;
    assertEquals(Math.PI, 3.14, ERROR_TOLERANCE);
}
```

Will fail if `ERROR_TOLERANCE = .001`

Oct 19, 2011

Sprenkle - CSCI209

35

Set Up/Tear Down

- May want methods to set up objects for every test in the class
 - Called **fixtures**
 - If have multiple, no guarantees for order executed

```
@Before
public void prepareTestData() { ... }

@Before
public void setupMocks() { ... }

@After
public void cleanupTestData() { ... }
```

Executed before each test method

Oct 19, 2011

Sprenkle - CSCI209

36

Example Set Up Method

```
private CD testCD;

@Before
public void setUp() {
    testCD = new CD("CD title", 100, 1997,
                    "CD Artist", 11);
}
```

@Before Executed before **each** test method
Can use **testCD** in test methods

Oct 19, 2011

Sprenkle - CSCI209

37

Expecting an Exception

- Handling Error Cases
 - Sometimes an exception *is* the expected result

Add an “expected” attribute:

```
@Test(expected=IndexOutOfBoundsException.class)
public void testIndexOutOfBoundsException() {
    ArrayList emptyList = new ArrayList();
    Object o = emptyList.get(0);
}
```

Test case passes iff exception thrown

Oct 19, 2011

Sprenkle - CSCI209

38

Set Up/Tear Down For Class

- May want methods to set up objects for set of tests
 - Executed once before any test in class executes

```
@BeforeClass
public static void
setupDatabaseConnection() { ... }

@AfterClass
public static void
teardownDatabaseConnection() { ... }
```

Oct 19, 2011

Sprenkle - CSCI209

39

JUNIT IN ECLIPSE

Oct 19, 2011

Sprenkle - CSCI209

40

Using JUnit in Eclipse

- Eclipse can help make our job easier
 - Automatically execute tests (i.e., methods)
 - We can focus on coming up with tests

Oct 19, 2011

Sprenkle - CSCI209

41

Using JUnit in Eclipse

- In Eclipse, go to your MediaItem project
- Create a new JUnit Test Case (under Java)
 - Use JUnit 4
 - Add junit to build path
 - Put in package media.tests
 - Name: DVDTest
 - Choose to test DVD class
 - Select setUp and tearDown
 - Select methods to test
- Run the class as a JUnit Test Case

Oct 19, 2011

Sprenkle - CSCI209

42

Example

- Test method that gets the length of the DVD
 - Revise: Add code to setUp method that creates a DVD
- Notes
 - Replaying all the test cases: right click on package
 - FastView vs Detached
 - Hint: CTL-Spacebar to get auto-complete options

Oct 19, 2011

Sprenkle - CSCI209

43

Unit Testing & JUnit Summary

- Unit Testing: testing smallest component of your code
 - For us: class and its methods
- JUnit provides framework to write test cases and run test cases automatically
 - Easy to run again after code changes
- JUnit Resources available from Course Page's "Resource" Link, under Java
 - API
 - Tutorials

Oct 19, 2011

Sprenkle - CSCI209

44

Project 1: Testing Practice

- Due next Wednesday
- Given: a Car class that only has enough code to compile
- Your job: Create a **good** set of test cases that **thoroughly/effectively** test Car class
 - Find faults in my faulty version of Car class
 - Start: look at code, think about how to test, set up JUnit tests
 - Written analysis of process

Oct 19, 2011

Sprenkle - CSCI209

45