Software Testing

V. Paúl Pauca

Department of Computer Science
Wake Forest University

CSC 331-631
Fall 2013

Software Testing I

Executing a program with the intent of finding errors

Two Key Goals

Validation: *Are we building the right product?*

Verification: *Are we building the product right?*

Boehm, 1979

Software Testing II

Validation testing

Does the software meet the technical requirements? (typical usage)

Defect testing

Is the system behaving incorrectly? (atypical situations)

Testing Issues

1. What is the primary purpose of testing?

Testing Issues

1. What is the primary purpose of testing?
2. Exactly what is tested for defects?

Testing Issues

1. What is the primary purpose of testing?
2. Exactly what is tested for defects?
3. What are the sources of these defects?

Testing Issues

1. What is the primary purpose of testing?
2. Exactly what is tested for defects?
3. What are the sources of these defects?
4. How does it matter whether a defect is found during the planning, development or post-release?

Testing Issues

1. What is the primary purpose of testing?
2. Exactly what is tested for defects?
3. What are the sources of these defects?
4. How does it matter whether a defect is found during the planning, development or post-release?
5. How should code be tested?

1. What is the primary purpose of testing?

- To detect failures so that defects may be uncovered and corrected

1. What is the primary purpose of testing?

- To detect failures so that defects may be uncovered and corrected
- Which is it:
  - Establishes a product functions properly under all conditions, or
  - Establishes a product does not function properly under specific conditions

1. What is the primary purpose of testing?

- To detect failures so that defects may be uncovered and corrected
- Which is it:
  - Establishes a product functions properly under all conditions, or
  - Establishes a product does not function properly under specific conditions

Testing can only show the presence of error, not their absence – Dijkstra, 1972

2. Exactly what is tested for defects?

- Scope: examine the code as well as the execution of the code in various environments and conditions

2. Exactly what is tested for defects?

- Scope: examine the code as well as the execution of the code in various environments and conditions
- Functional testing: tests that verify a specific function of the code
  Does this particular feature work?

2. Exactly what is tested for defects?

- Scope: examine the code as well as the execution of the code in various environments and conditions
- Functional testing: tests that verify a specific function of the code
  Does this particular feature work?
- Non-functional testing: tests that verify aspects not directly related to function
  Is the system scalable? Is it secure?

3. What are the sources of these defects?

- Software fault:
    - programmer makes an **error** (mistake)
    - results in a **defect** (fault, bug) in the code

3. What are the sources of these defects?

- Software fault:
  - programmer makes an **error** (mistake)
  - results in a **defect** (fault, bug) in the code
- Failure: If defect is executed, producing wrong results

3. What are the sources of these defects?

- Software fault:
  - programmer makes an **error** (mistake)
  - results in a **defect** (fault, bug) in the code
- Failure: If defect is executed, producing wrong results

Not all defects will necessarily result in failures!

3. What are the sources of these defects?

- Software fault:
  - programmer makes an **error** (mistake)
  - results in a **defect** (fault, bug) in the code
- Failure: If defect is executed, producing wrong results

  Not all defects will necessarily result in failures!

- Can all defects be traced to coding errors?

3. What are the sources of these defects?

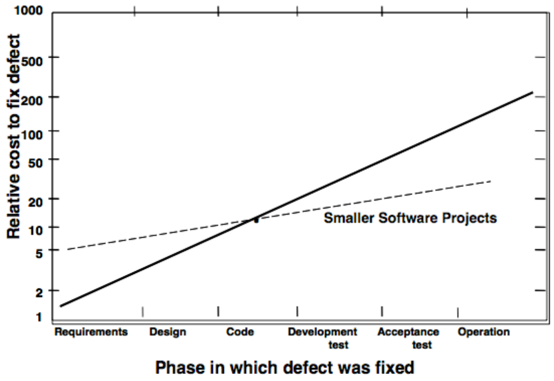
- Software fault:
  - programmer makes an **error** (mistake)
  - results in a **defect** (fault, bug) in the code
- Failure: If defect is executed, producing wrong results

  Not all defects will necessarily result in failures!

- Can all defects be traced to coding errors?

  Other culprits:
  - Missing requirements $\rightarrow$ errors of omission in design
    Common case: Non-functional req. (scalability, maintainability, security, etc.) were not specified
  - Ambiguity of requirements
  - Incorrect design

4. How does it matter whether a defect is found during the planning, development or post-release?



Barry Boehm - A View of 20th and 21st Century Software Engineering

- *The earlier a defect is found the cheaper it is to fix it*
- E.g. if problem in requirements is found only in post-release, then the cost to fix it is 10-100 times more

5. How should code be tested?

- Static approaches: when the software isn’t actually used
  - Reviews
  - Walkthroughs
  - Inspections

5. How should code be tested?

- Static approaches: when the software isn't actually used
  - Reviews
  - Walkthroughs
  - Inspections
- Dynamic approaches: when the software is actually executed
  - Break test domain into regions and test the boundaries
    Test domain for a code?
  - Can take place *before, during, and after* program is complete
    Stubs (piece of code standing-in for some functionality) often used while program is being completed.

Testing Methods

Black box testing

- Software product is a **black box**
- No knowledge of the implementation is assumed
- Many methods proposed, e.g. equivalence partitioning, boundary value analysis, etc
- Specification-based testing: Verify output for well-defined test cases

Testing Methods

Black box testing

- Software product is a **black box**
- No knowledge of the implementation is assumed
- Many methods proposed, e.g. equivalence partitioning, boundary value analysis, etc
- Specification-based testing: Verify output for well-defined test cases

White box testing

- Have access to internal data structures and algorithms
- testing API
- testing code coverage, etc.
- Includes all static testing

Testing Levels

- Unit Testing
  - Verifying functionalitty of a specific section of the code
  - Usually at class level in OO environments
  - Written by the developers
  - Ensures building blocks can work independently

Testing Levels

- Unit Testing
  - Verifying functionalitty of a specific section of the code
  - Usually at class level in OO environments
  - Written by the developers
  - Ensures building blocks can work independently
- Integration Testing
  - Verifying the interfaces between components against a software design
  - Integration can be iterative, or
  - all together (big bang)
  - Aim is to expose defects in the interfaces and interaction between modules

Testing Levels

- Unit Testing
  - Verifying functionalitty of a specific section of the code
  - Usually at class level in OO environments
  - Written by the developers
  - Ensures building blocks can work independently
- Integration Testing
  - Verifying the interfaces between components against a software design
  - Integration can be iterative, or
  - all together (big bang)
  - Aim is to expose defects in the interfaces and interaction between modules
- System Testing
  - Testing of the completely integrated system

Unit Testing with JUnit

- A tool for testing OO Java code
- A JAR linked at compiled time
  - `junit.framework` for v. 3.8 and earlier
  - `org.junit` for v. 4.0 and later
- Has been ported to many other languages

Framework for JUnit v. 4.0

- Classes: `Assert`, `Assume`, `Test.None`
- Annotations: `@After`, `@AfterClass`, `@Before`, `@BeforeClass`, `@ClassRule`, `@Ignore`, `@Rule`, `@Test`

Simple example:

```
public class Example {
   File output;
   @Before
   public void createOutputFile() {
         output= new File(...);
   }
   @Test
   public void something() {
         ...
   }
   @After
   public void deleteOutputFile() {
         output.delete();
   }
}
```

Sample Code[a]

[a] Hong Qing Yu

```
class Money {
   private int fAmount;
   private String fCurrency;

   public Money(int amount, String currency) {
        fAmount= amount;
        fCurrency= currency;
   }
   public int amount() { return fAmount; }

   public String currency() { return fCurrency; }
   public Money add(Money m) {
        return new Money(amount()+m.amount(), currency());
   }
}
```

A JUnit v.4 Test Class

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

public class MoneyTest {

   @Test
   public void testSimpleAdd() {
      Money m12USD= new Money(12, "USD");
      Money m14USD= new Money(14, "USD");
      Money expected= new Money(26, "USD");
      Money result=  m12USD.add(m14USD);

      assertTrue(expected.equals(result));
   }
}
```

Will this test pass or fail?

Another JUnit v.4 Test Class

```
public class MoneyTest {
   private Money f12USD;
   private Money f14USD;

   @Before public void setUp() {
      f12USD= new Money(12, "USD");
      f14USD= new Money(14, "USD"); }

   @Test public void testSimpleAdd() {
      Money expected= new Money(26, "USD");
      Money result= f12USD.add(f14USD);
      assertTrue(expected.equals(result)); }

   @Test public void zeroTest() {
      Money zero= new Money(0, "USD");
      Money result= zero.add(zero);
      assertTrue(result.equals(zero)); }

   @Test(expected=ArithmeticException.class)
   public void negativeTest() {
      Money odd= new Money(-16, "USD");
      Money result= f12USD.add(odd);
      Money expected = new Money(-4, "USD");
      assertTrue(result.equals(expected)); }
}
```

Useful Resources

`junit.sourceforge.net/javadoc`

`open.ncsu.edu/se/tutorials/junit/`

`developer.android.com/training/activity-testing/activity-unit-testing.html`

`mobile.tutsplus.com/tutorials/android/android-sdk-junit-testing/`