CMSC 132: Object-Oriented Programming II

Program Testing

Department of Computer Science
University of Maryland, College Park

Overview

- Testing
  - Types of testing
  - Levels of testing
**Program Testing**

- **Goal**
  - Detect and eliminate errors in program
  - Feedback to improve software
    - Specification changes
    - Add new functionality
  - Extremely important for success!

**Program Testing**

- **Empirical testing**
  - Test software with selected test cases
  - More scalable than verification
  - Not guaranteed to detect all errors
Testing – Terminology

- Test case
  - Individual test

- Test suite
  - Collection of test cases

- Test harness
  - Program that executes a series of test cases

- Test framework
  - Software that facilitates writing & running tests
  - Example – JUnit

Testing – Terminology

- Test driver
  - Program to create environment for running tests
  - Declares variables, creates objects, assigns values
  - Executes code and displays results of tests

- Stub
  - Skeleton code in place of unfinished method / class
  - Simply return if called
  - Possibly print message indicating stub called
  - Allows software testing to begin
Testing – Terminology

- Tester (Quality Assurance)
  - Person devising and / or performing tests
  - More effective if 2nd person writes tests

- Walkthrough
  - Programmer explains code to 2nd person

Types of Testing

- Clear box testing
  - Allowed to examine code
  - Attempt to improve thoroughness of tests

- Black box testing
  - No knowledge of code
  - Treat program as “black box”
  - Test behavior in response to inputs
Levels (Stages) of Testing

1. Unit test
2. Integration test
3. System test
4. Acceptance test

Unit Test

- Test individual units extensively
  - Classes
  - Methods
- Central part of Extreme Programming (XP)
  - Extensive unit testing during development
    - Pair programming (1 coder, 1 tester)
    - Design unit tests along with specification
- Approach
  - Test each method of class
  - Test every possible flow path through method
Flow Path

- Unique execution sequence through program

Example

```
S1
while (B1) {
  if (B2)
    S2
  else
    S3
}
```

Flows

- `S1`
- `S1, S2`
- `S1, S3`
- `S1, S2, S2`
- `S1, S2, S3`
- `S1, S3, S2`
- `S1, S3, S3`
- `...`

Unit Test – Flow Path

- Not possible to test all flow paths
  - Many paths by combining conditionals, switches
  - Infinite number of paths for loops
  - New paths caused by exceptions

- Test coverage
  - Whether code is executed by some test case
  - Alternative to flow path
  - Ensure high % (if not all) of lines of code tested
  - Does not capture all possible flow paths
    - Even if all lines of code tested by some test case
Unit Test – Test Coverage

Automated test coverage

- Performed by submit server
- For code, reports # tests executing code
  - For set of tests selected (from link)
    - E.g., all public tests, individual test
- For conditionals, reports X/Y where
  - X = # tests executing True
  - Y = # tests executing False
- Color
  - Green = executed by some test case
  - Pink = not executed

Unit Test – Test Coverage Example

Coverage information for public-student test #all:

<table>
<thead>
<tr>
<th>Source file</th>
<th>statements</th>
<th>conditionals</th>
<th>methods</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilities.ja</td>
<td>10/10</td>
<td>5/5</td>
<td>2/2</td>
<td></td>
</tr>
</tbody>
</table>

```java
package utilities;

public class Utilities {
  public static String letterGrade(double numericGrade) {
    if (numericGrade >= 90.0)
      return "A";
    else if (numericGrade >= 80.0)
      return "B";
    else if (numericGrade >= 70.0)
      return "C";
    else if (numericGrade >= 60.0)
      return "D";
    else
      return "F";
  }

  public static boolean passingGrade(double numericGrade) {
    return numericGrade >= 70.0 ? true : false;
  }
}
```
Integration Test

- Test interaction between units
  - Possible units fail when combined
  - May find problems in specifications
- Approach
  - Test units together
  - Proceed bottom up, in increasing size
- Example test sequence
  1. AB, AC, AD, CD, CE
  2. ACD
  3. ABCDE

System Test

- Test entire software
  - Include all components of software
  - In context in which software will be used
- Ensure all pieces of software interact correctly
Acceptance Test

- Test full functionality of software
  - Ensure program meets all requirements

Approach

- Place software in user environment
- Test software with
  - Real-world data
  - Real users
  - Typical operating conditions
  - Test cases selected by users

Acceptance Test – Stages

- Alpha test
  - Test components during development
  - Usually clear box test

- Beta test
  - Test in real user environment
  - Always black box test
Regression Test

- Ensure functionality is not lost / changed
  - As software is modified / extended

Approach

- Save suite of tests and expected results
- Rerun test suite periodically after software changes
- Report any loss of functionality

Typically run overnight

- Software is more stable when developers leave work

Developing Test Cases

- Quality of testing depends on test cases

Tips on developing test cases

- Develop test data during analysis & design phases
- Attempt to exercise alternate program paths
- Check boundary conditions
  - 1st and last iterations of loop
  - 1st and last values added to data structure
- Pay close attention to problem specification
- UML use cases → test cases