Lecture 8: Testing

CS190: Software System Design

February 20, 2002
Steven P. Reiss

I. Today's Class

A. Testing

1. What it means
2. How to do it

B. Testing in the context of XP

II. Start with an example

A. Suppose you have a function that takes three integer values as parameters. The three values are interpreted as representing the lengths of the sides of a triangle. Routine should return a string indicating whether the triangle is "scalene", "isosceles", or "equilateral".

B. Write down all the test cases you think you would need to adequately test this function.

C. Lets score: Give yourself 1 point for each:

1. A test case that is a valid, non-degenerate, non-isosceles, non-equilateral scalene triangle.
2. A test case that represents a valid non-degenerate equilateral triangle
3. A test case that represents a valid non-degenerate isosceles (but not equilateral) triangle
4. At lease three test cases that represent valid isoceles triangles showing the different permutations
5. A test case in which one side has a zero value
6. A test case in which one side has a negative value
7. A test case with three integers greater than zero such that the sum of two numbers is equal to the third
8. All permutations of previous

9. A test case with three integers greater than zero such that the sum of two numbers is less than the third
10. All permutations of previous
11. A test case with all sides 0
12. If you specified the output of each of your test cases

III. What is Testing

A. Testing is the process of executing a program with the intent of finding errors

1. Destructive, not constructive process
2. Successful tests find bugs
3. Errors can be of omission or commission

B. Testing is not

1. Verification
2. Debugging

C. Testing serves a purpose in XP

1. To support rapid program construction
2. To support continuous integration
3. To provide confidence in other’s code

IV. Testing Principles

A. A necessary part of the test case is the expected output

B. A programmer should avoid testing his or her own programs

C. Thoroughly inspect the results of each test

D. Test cases must be written for invalid and unexpected as well as valid and expected conditions

E. Check that the program does not do what it is not supposed to do

F. Avoid throw-away test cases unless the program is a throw-away program

G. Plan testing on the assumption that errors will be found

H. The probability of the existence of one or more errors in a section of code is proportional to the number of errors already found in that section

I. Testing is an extremely creating an intellectually challenging task

V. Testing Methodology (XP)

A. Non-execution testing

1. Simulate your code by hand

a) As you write it out

b) As you type it in

c) As part of pair programming

2. Walkthroughs are formal attempts at this

B. Write the test cases as you write the code

1. In C++ put a testing main program in each package directory.

2. In Java, add a mainline to each class to test just that class and a test program in the package to test the package as a whole
3. As you add classes and methods, add the corresponding test cases for them.
4. Using exceptions to report errors makes this easier

C. Maintain the test cases as the code evolves

1. Add new test cases rather than replacing old ones
2. This is regression testing

D. Write a test harness

1. Script that runs all the test cases for a package
2. Checks and flags any problems (differences from what is expected)
3. Run as a shell script or from make

VI. Test Coverage

A. Tests should be written to find errors

B. Tests should cover all possible inputs/outputs, program features, etc.

1. Boundary cases
2. Special effects
3. Errors
4. Working cases

C. Tests should cover all possible executions

1. Statement coverage
   - a) Minimum expected
   - b) tcov for C++
2. Decision coverage
   - a) Each branch should be taken at least once
3. Condition coverage
   - a) Each logical condition takes on each output value once
4. Multiple condition coverage
   - a) All possible combinations of conditions are checked

D. Test cases should cover multiple uses

VII. Integration Testing

A. Bottom-up Testing

1. **Test modules first**
2. **Then test larger and larger units of the program**
3. **Advantages**
    - a) Good if major flaws occur at bottom of program
    - b) Test conditions are easier to generate
    - c) Observation of test results is easier
4. **Disadvantages**
    - a) Driver modules must be produced
    - b) Driver modules have bugs
    - c) Program not tested in full early on

B. Top-down Testing

1. **Write stubs for unwritten routines**
2. **Test from the main program down**
3. **Advantages**
    - a) Good if major flasw occur at top of program
    - b) Once I/O has been added, test cases easy to generate
    - c) Whole program runs early on
4. **Disadvantages**
    - a) Stub moudles must be produces -- complex & buggy
    - b) Test cases difficult w/o I/O
    - c) Test cases difficult with GUI
    - d) Test conditions may be impossible or difficult to create
    - e) Observation of results can be difficult

C. Compromise

1. **Do both -- bottom up testing at the package**
2. **Top down testing for each of your stories**
    - a) Have a suite of tests that should be run and designate someone in the group as in charge of running them
    - b) This can change daily or weekly