The UNIVERSITY of VERMONT

Benjamin Fenster
CS-100 (Damon)
8 February 2003

Notes – Testing

I. Concept
   a. We know the specification and we use it to write the code.
   b. The code *should* behave as the specification describes, but we know it may not always work perfectly on the first try.
   c. How do we determine whether it's working or not?

II. Procedure
   a. Run the program with a known set of inputs and observe the output.
   b. Any unexpected result is a bug.
   c. The *goal* is to find bugs!
      i. Finding no bugs is a failure!
      ii. The test should be designed, then, to uncover bugs, not to demonstrate what's already working.
      iii. A test *cannot* prove that no bugs exist.
      iv. It might build confidence if no bugs can be uncovered, but the goal is always to find them.
   d. Write the tests first!
      i. Before writing the code
      ii. If you know what the test will uncover, you can write the code better the first time.
      iii. You may even have to refine the specification as you realize what the test involves.
      iv. It forces you to consider concrete cases of runs, which makes it easier to understand the general case.

III. Approaches
   a. Black Box
      i. Can't see the inside.
      ii. Tests are based entirely on the specification.
      iii. Written first, and can apply to subclasses.
      iv. Expected Behavior
         1. Test the mainstream behavior.
         2. What will *normally* happen when the code is run?
         3. What inputs will normally be given?
      v. Fringe cases?
         1. What happens with valid, but fringe, inputs?
         2. Eg: Inserting at the beginning of a list, sending a minimum or maximum.
         3. Test the cases that are still specified as valid, but which might potentially be handled differently (perhaps incorrectly?) in the code.
         4. Remember that you still only get to see the specification, not the code, so you're guessing what might be handled differently.
      vi. Error Cases
         1. Deliberately send bad input and see what happens.
         2. It should handle it *reasonably,* but not necessarily normally.
         3. No infinite loops, some indication of failure returned, etc.
         4. Potentially MANY convoluted invalid sets of data.
         5. Opportunity exists to do some really devious things, so do! Remember the goal is to find bugs.
   b. White Box ("Glass Box")
      i. You've seen the code, so you can base the test on what you know it will do.
      ii. You can identify sections that might potentially cause problems.
      iii. Look at the flow of control, make sure one thing flows properly to the next.
      iv. Makes ure each piece executes correctly and that all branches are taken as appropriate.

        v. Does a pointer ever enter a branch as null unexpectedly, for example?
c. For Assignments
    i. Test the mainstream cases and a few error cases.
    ii. There won't always be time to do as thorough testing as other circumstances might demand.