The UNIVERSITY of VERMONT

Benjamin Fenster
CS-100 (Damon)
6 April 2003

Notes – Testing (Continued)

I. Review
   a. Write tests before you write the code.
      i. That helps make the code easier to write.
      ii. It also encourages doing the testing.
   b. Black Box
      i. You don't know what the code says, just what the specs say.
      ii. Check normal cases. What will happen most of the time?
      iii. Fringe cases?
      iv. Error cases?
   c. White Box
      i. Written while looking at the code.
      ii. Design tests to make the code execute in different configurations.
      iii. If there's an if..else split, make sure both halves get run.
   d. Example
      i. Steering on RobotControl
      ii. Try +6, -5 (normal), +10, -10 (fringe), 0, +1, -1 (fringe)

II. Testing Classes
   a. Test every public method.
      i. If private methods are called from those public methods, they get tested too.
      ii. Any private method that's not called isn't worth concern.
   b. Test every fundamentally different state
      i. Can't check every single possibility.
      ii. Can look for key differences from one to another and try those.
   c. Write a test() function for each class
      i. It sets up all appropriate scenarios and calls each public method.
      ii. It then interprets the result.
      iii. It always returns true or false for success or failure.
      iv. It may or may not print some results.
   d. Want to guarantee the rep invariant.
      i. Private methods are allowed to break it, but public methods must maintain it.
      ii. repOk() helps here.
   e. Would like to say that if each individual call holds, any combination of calls will too.
      i. That's not always true.
      ii. To be 100% thorough you'd need to test every single combination of results
      iii. Since it's not possible to do that, group similar sets of input together.
      iv. If any one case in a group works, they can all be assumed to work.
      v. It's not always easy to develop appropriate groups.

III. Testing Type Hierarchies
   a. Start with a minimum test. Always call super.test() since that needs to work anyway.
   b. Also need to know that the subclass won't replace some methods that result in a bug
   c. Design constraints that make any legal subclass work in some reasonable way.
   d. Most testing looks for bugs in the code. These look for bugs in the specifications.
   e. You need to define what's legal and illegal accurately.
   f. The goal is to give well-meaning developers the tools to keep from breaking it.
   g. Malicious developers will always be able to break it.

IV. Unit Testing vs. Integration Testing
   a. Unit testing looks at a specific class.
   b. Integration testing deals with system.
   c. Full system testing worries about how / what the user will do.
      i. A whole new way to do testing.
      ii. Doesn't include any white box testing – the user doesn't know or care about the code.
      iii. The point is to make sure that the user gets what s/he expects all the time.