Testing Basics & Unit Test

PSU CS 300 Lecture 8-2

Bart Massey
Assoc Prof Computer Science
Portland State University
<bart@cs.pdx.edu>
Unit test is a laboratory

- “Unit” = procedure, function, method, etc
- Most real units aren't well specified and designed
- Most new unit testing driven by new methodologies
- Unit test informs system test
Testing basics

- **Input**: sequence of inputs presented to program after startup
- **Output**: sequence of behaviors of program after startup
- **Test case**: input $\rightarrow$ output
More testing basics

- **Test set**: set of test cases
- **Domain**: set of all possible inputs; **large or infinite**
- **Subdomain**: subset of a domain with interesting properties
Stubs and drivers

- **Stub**: for testing a system or unit that depends on code that has not yet been written
- **Driver (harness)**: for testing code outside of its designed environment (may also not have been written)
Black-box and other-box

• **Black box testing:** as though interface is opaque; can see only interface requirements

• **White / clear / broken box testing:** can look at design and/or code in order to try to improve testing
Black box tests

- random
- profile / user
- domain coverage
- subdomain coverage
- range coverage
- error tests
White box tests

- **Boundary conditions**
  - control boundaries
  - data boundaries
  - mutation
  - fault seeding

- **Test coverage**
  - control: statement, branch, path
  - data: range analysis
What's a test plan?

- In either order
  - Generate test set
  - Get code to test
- Write needed stubs / drivers
- Run tests
- Measure and analyze output
Regression testing

- Save test set for maintenance changes
- Add tests during maintenance activities
- Automate test runs
- **Fight regressions:** things become “unfixed”
Unit testing

- **Usually white-box**
  - in fact, *may not have any spec other than the code itself*

- **Naturally bottom-up**
  - integration test plan is a natural style of test

- **Sparse and targeted**
  - make system test easier
Test-driven development

- Write the unit test
- *(XP: run the unit test and make sure it fails)*
- **Then** write the code
- Then verify that it passes
- Not my favorite style
Code coverage

- Good code coverage is a common testing goal
- **Coverage tools** help measure
- Easier to cover code with unit tests than with system tests
- But tests may not reflect “normal” input domain
Unit stubs and drivers

- **White box:** requires accessing module internals
- **Many tools and libraries** for auto-generating drivers
- Stubs are a problem; bottom-up helps
- **OO folks are leaders here**
What unit tests give

- Do pseudocode + inspection, supercede unit tests? No
  - Implementation mistakes can be quite subtle
  - Unit testing can be slightly cheaper than inspection
- But like all V&V, need to limit use to needed portion