# CSC 405 Lecture Notes Week 3

## Some Project Specific Testing Details
## Relevant Testing Theory

# I. Mapping Testing Terminology onto 405 and 406

A. Consider last week's testing activities.

B. What follows are notes on how they'll be accomplished in 405 & 406.

C. These are ***notes***, not ***the*** answers.

# II. Unit Testing

A. For the OCU, CppUnit looks like a good choice.

1. Originally a port of JUnit to C++.

2. Start at sourceforge site.

# Unit Testing, cont'd

B. For the kernel-level code,
unit testing may not be done independently.

1. See the "Linux Test Project" for a start.

2. It may provide some unit testing.

3. It also is at sourceforge.

# III. Module Testing

A. CppUnit provides test fixtures.

B. It provides other class-level testing features.

C. See the CppUnit "Cookbook".

# Module, cont'd

D. LTP provides support for kernel module testing.

E. There is also the *scalable testing platform*.

# IV. Integration Testing

A. OCU and kernel-level independently tested.

B. OCU teams write stub for kernel model.

C. Not clear what kernel stubs may be possible.

# V. System Testing

A. The issue of kernel builds needs to be worked out.

B. Also, the issue of how far to go with competing 802-11 and Batman needs to be determined.

# VI. Acceptance Testing

A. Once OCU user-level spec is done, write acceptance spec from it.

B. If possible, have actual human OCU users conduct acceptance tests (or Julie as customer rep).

C. STP claims to provide kernel acceptance testing.

# VII. Black Box Testing

A. The OCU may be black-box testable in its entirety.

B. For kernel-level, black-box testing is driven by specs, as possible.

# VIII. White box

A. OCU may not need explicit white-box tests.

B. For kernel, it depends on what LTP has to offer, or comparable testing frameworks.

# IX. Testing Design

A. To be determined based on chosen frameworks.

B. For the physical layout, I suggest standard UNIX directories, a la the CSC 309 testing repository.

# X. Test Plan

A. I *strongly* recommend a common, uniform structure for these

B. C/C++ Function comment for unit test plans.

C. C++ .h file comment for module test plans.

D. Source-resident .html for integration plan.

E. Use `doxygen` (or equiv) for generating browsable test plans.

# XI. Top-down Testing

A. Test OCU top-down, with MESH stub.

B. OCU model unit tests may be done bottom up, as necessary.

# XII. Bottom-up Testing

A. Test kernel-level bottom up / inside out.

# XIII. Test Case

A. Again, I recommend having a standard test design convention for how this is represented.

# XIV. Testing Oracle

A. For OCU, given absence of formal(ized) spec, the oracle will be a human domain expert.

B. For kernel-level, the extant specs are used, with human interpretation as necessary.

# XV. Testing Stub

A. As noted above, major stub is for OCU testing.

# XVI. Test Driver

A. Provided by the chosen frameworks.

B. Also, if dynamic testing feature is implemented, the OCU is a live run-time test driver.

# XVII. Regression Testing

A. CppUnit is both programmatic and file based.

B. Kernel-level is most likely log file based.

C. STP may provide regression support.

# XVIII. Test Coverage

A. `gcov` is a widely-used C coverage tool.

B. LTP has `lcov`, a graphical front-end to `gcov`.

# XIX. Test Subsumption

A. OCU is simple enough so as not to need this.

B. LTP may have some support, but not likely.

# XX. Test Automation

A. Done with `make`, per the requirements of the chosen frameworks.

# XXI. Mutation Testing

A. Most likely not performed in 405.

B. We may do some in 406

# XXII. Testing Harness

A. *Covered above, particularly under Test Drivers.*

# XXIII. Testing Framework

A. To summarize ...

B. CppUnit for OCU testing.

C. LTP for kernel-level testing.

D. Other possibilities are CxxTest, DejaGnu.

# Resources

- opensourcetesting.org
- sourceforge.net/apps/mediawiki/cppunit
- ltp.sourceforge.net
- cxxtest.tigris.org
- www.gnu.org/software/dejagnu
- sourceforge.net/projects/stp

# XXIV. **Design and testing best practices.**

A. Concrete structure in Fisher's 309 SOPs:

1. **Vol 1** -- Repository Structure
2. **Vol 2** -- Design & Imple'n Conventions
3. **Vol 3** -- Testing Conventions

# Design and testing best practices, cont'd

B. Widely-used tools:

1. CppUnit
2. Gcov
3. Linux Test Project, Lcov

# Design and testing best practices, cont'd

C. An exemplary "best testing practice site" --

SQLite Testing