How to Add and Improve Testing in your CSE Software Project

The IDEAS Scientific Software Productivity Project
ideas-productivity.org/resources/howtos/

Overview: Adding tests of sufficient coverage and quality improves confidence in software and makes it easier to change and extend. Tests should be added to existing code before the code is changed. Tests should be added to new code before (or while) it is being written. These tests then become the foundation of a regression test suite that helps effectively drive future development and improves long-term sustainability.

Target Audience: CSE software project leaders and developers who are facing significant refactoring efforts because of hardware architecture changes or increased demands for multiphysics and multiscale coupling, and who want to increase the quality and speed of development and reduce development and maintenance costs.

Purpose: Show how to add quality testing to a project in order to support efficient modification of existing code or addition of new code. Show how to add tests to support (1) **adding a new feature**, (2) **fixing a bug**, (3) **improving the design and implementation**, or (4) **optimizing resource usage**.

Prerequisites: First read the document *What Are Software Testing Practices?* and browse through *Definition and Categorization of Tests for CSE Software*.

Steps:

1. Set up **automated builds of the code** with high warning levels and eliminate all warnings.
2. **Select test harness frameworks**
   a. **Select a system-level test harness** for system-executable tests that report results appropriately (e.g., CTest/CDash, Jenkins).
   b. **Select a unit test harness** to effectively define and run finer-grained integration and unit tests (e.g., Google Test, pFUnit).
   c. **Customize or streamline** system-level and/or unit test frameworks for use in your particular project.
3. **Add system-level tests** to protect major user functionality.
   a. Select inputs for several important problem classes and run code to produce outputs.
   b. Set up no-change or verification tests with a system-level test harness in order to pin down important behavior.
4. **Add integration and unit tests** (as needed for adding/changing code)
   a. **Incorporate tests** [1, 2] **for code to be changed**
      - **Identify change points** for target change or new code.
      - **Find test points** where code behavior can be sensed.
      - **Break dependencies** in order to get the targeted code into the unit test harness.
      - **Cover targeted code** to be changed with sufficient (characterization) tests.
   b. **Add new features or fix bugs with tests** [1, 2, 3, 4]

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.

Version 0.2, April 25, 2016

- **Add new tests** that define desired behavior (feature or bug).
- Run new tests and **verify they fail.**
- Add the minimal code to **get new tests to pass.**
- **Refactor** the covered code to clean up and remove duplication.
- **Review** all changes to existing code, new code and new tests.

5. Select **code coverage** (e.g., gcov/lcov) and **memory usage error detection** (e.g., valgrind) analysis tools.
6. Define a set of **regression test suites**
   a. Define a faster-running **pre-push regression test suite** (e.g., single build with faster running tests) and **run it before every push**.
   b. Define a more comprehensive **nightly regression test suite** (e.g., builds and all tests on several platforms and compilers, code coverage, and memory usage error detection) and **run every night**.
7. Have a policy of **100% passing pre-push regression tests** and work hard to maintain that.
8. Work to **fix all failing nighty regression tests** on a reasonable schedule.

FAQs:

Q: *Why do you need both a system-level and a unit test harness?*

A: A unit test harness aggregates hundreds of unit and integration tests into single executables. A system-level test harness runs these aggregate integration and unit test executables along with the other system-level acceptance and verification tests and alerts developers of any failures.

Q: *Why not just add all of the tests for an existing code and get it over with?*

A: Taking weeks or months (or years) to add sufficient tests for an entire existing code (that lacks sufficient testing) is not usually economical or necessary. Tests need to be added to code only when it is changed (or when adding new code). In that way tests can be added while regular development work is being done.

Q: *Why demand 100% passing pre-push regression tests?*

A: This avoids expensive debugging and other investigations needed to determine whether your changes are breaking failing tests or not (hard). If all tests pass, then your changes could be breaking them (easy).

References:

[1] Feathers, Michael. *Working Effectively with Legacy Code.* Prentice Hall, 2005

[2] *Legacy Software Change Algorithm*: *http://trilinos.org/trac/trilinos/wiki/TribitsLegacySoftwareChangeAlgorithm*

[3] Beck, Kent. *Test Driven Development.* Addison Wesley, 2003

[4] McConnell, Steve. Code Complete (Second Edition). Microsoft Press, 2004

This document was prepared by Ulrike Yang, Roscoe A. Bartlett, Glenn Hammond, Xiaoye Li, Barry Smith, and James M. Willenbring.

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.