

Why we should Test First

SLUG/Linux Australia roundup
April 2006

© 2006 Creative Commons share-alike licence,
Robert Collins <robertc@robertcollins.net>

Who am I?

- Contributor to a few open source projects.
 - Project lead for a few others.
 - Project lead for the Bazaar project @ Canonical for the last two years
 - Now Quality Czar at Canonical.
-
-

Why Test?

- Regressions
 - Does my project work on platform X?
 - “Are we there yet?”
 - Writing tests encodes decisions.
 - Testing encourages clean interfaces.
 - Testing encourages clean implementation.
 - Testing is awesome!
-
-

Regressions

- A test for the exact regression can prevent it recurring. For instance a test of the routine that was behaving badly which fails if the routine starts to display the erroneous behaviour. [As long as you run the tests!].
 - A test for the *style* of regression can catch variations on a theme. For example, a *user story* test where you drive the top level UI through the scenario that created the regression.
-
-

Does my project work on platform X?

- If you have comprehensive tests that work on platform X, its a fairly good bet that your project will work too. [Buildbot]
 - When you have platform specific limitations/changing defaults/bugs, platform specific tests can allows testing that those limitations are obeyed. e.g. choosing aufs vs diskd.
 - Simulation can allow tests to fail when code would trigger an issue on a different platform. I.e. simulated hardware devices for linux development. [lca kernel devel tutorial.]
-
-

“Are we there yet ?”

- Write a test that performs the actions you want a user to be able to perform. When the test starts passing, you have delivered that specific *user story*.
 - Developers can read failing user story tests to see what remains to be done.
 - Finer grained tests flag milestones reached on the way to larger goals.
 - Eventually these tests are sufficiently small that they are easy to make pass by writing the code for them. [Forgetting NP-C problems!]
-
-

Writing tests encodes decisions

- When you write a test, you typically make an *assertion* that a specific call or action will have a specific result and specific side effects. This encodes a decision about how your software should behave.
 - Not ALL decisions can be recorded in this fashion, but every one that *is* recorded is enshrined – code changes that alter this *assertion* will result in a test failure.
 - This documents your program too.
-
-

Cleaner interfaces

- Testing exercises code paths.
- To exercise a code path, you need to be able to exercise it. (Yes, that's circular!).
- So the act of writing a test to exercise a code path forces you make it accessible.

Cleaner implementations

- Testing an implementation ensures it works in a specific environment.
 - Different tests vary that environment.
 - Hidden dependencies and assumptions start to bubble up and become visible.
 - Because you have tests for the functionality you care about, you can rearrange your code more easily.
-
-

The Red-Green-Refactor cycle.

- See 'Test Driven Design' by Kent Beck.
 - Red: Test written/modified and fails.
 - Green: Code changed so all tests now pass.
 - Refactor: Aggressively remove duplication.
 - If you refactor your code to reduce/remove duplication after adding each test, your code is leaner and more malleable, and so are your tests. (Each test adds a triangulation point for providing clear objects/functions).
-
-

Add this all up: Test First gives you

- Clean implementations.
 - Clean interfaces.
 - You dont need to figure out what you intended the code to some arbitrary time later!
 - Goal posts for when to stop.
 - Better cross-platform confidence.
 - Regression testing facilities for free.
 - An easier development process.
-
-

How to test?

- Automated tests
 - Make check – an interface for users.
 - xUnit – pyunit, nunit, sunit, shunit, check, cppunit.
These can perform essentially anything within the test.
 - Heuristic tests – for instance, run 1000 operations from a defined set in random order with random parameters, and any error-code or exception is a 'fail'. See for instance the 'crash-windows' testing tools.
 - Human driven tests
 - Test scripts that a human can execute.
 - Beta programs, where people just get out and use the software. - suck it and see.
-
-

But but but I'm stuck without tests

- Start the Red-Green-Refactor cycle!
- Ignore the fact you are lacking tests.
- Write a test for as many changes you make from here on as you can reasonably do.
- KISS!

Crashes while writing this using openoffice 2.0 impress:

- Six crashes.
- Last edits lost in one case.

