Automated Testing Playbook

Automated testing roadmap

This is a roadmap towards improving a team's automated testing practices, loosely based on Google's Test Certified program, as encapsulated in the "Test Certified: Lousy Name, Great Results" Testing on the Toilet episode. Though these steps are not split into "levels", they should be followed in roughly the order listed below.

WORK IN PROGRESS: This material is currently in draft form and under active development. See the GitHub issues page to examine progress.

Set up a continuous build

A continuous build system is the first important step in establishing an automated testing culture. Even if the system does not currently have tests, a continuous integration and build system will encourage the team to keep the build free of non-test-related breakages.

Register a coverage bundle

Establishing baseline code coverage metrics gives visibility into where tests are currently lacking, and encourages progress in those areas.

Classify tests by size

Classifying tests by Small (unit), Medium (integration), and Large (system) test sizes identifies the current balance of automated tests.

Create a smoke test suite

If a test suite takes too long to run, identify a set of "smoke tests" that must be run before each submit.

Identify nondeterministic tests

Tests that arbitrarily fail without changes to the code that is being tested or the inputs are called "nondeterministic" or "flaky." Flakiness indicates that not all of the inputs to a test are properly controlled, and rely on "outside resources" such as the system clock, remote systems, etc. As a first step to tackling flaky tests, triage them using a special label or directory, so that team members know that such tests need careful attention and should be fixed eventually.

Establish Test classification ratio

For systems beyond a certain size and complexity, there should be a reasonable balance between Small (unit), Medium (integration), and Large (system) tests that makes sense. This enables informed decisions about the kind of tests the system should have and the kind of refactoring work necessary to support this target balance.

No releases with broken tests

Once the team has established its continuous build, identified tests by size and flakiness, and possibly defined a smoke test suite, team policy should mandate that all tests must pass before each new release.

Require tests for all nontrivial changes

To begin building up coverage, team policy should mandate that all new nontrivial changes should be accompanied by automated tests. If a change is covered by an existing test, it's a judgment call on behalf of the author and reviewer whether or not a new test should be written to test the new change at a different scope from the existing test.

Require tests or smoke tests to pass before submit

To ensure the build always remains in a passing state, team policy should dictate that either all tests, or the smoke test suite, should pass before each submitted code change.

No nondeterministic tests

Eventually, all flaky tests should be fixed so that they no longer fail barring a change in the test inputs or the code being tested.

Total coverage of at least X%

The team should establish a baseline coverage goal and ensure that coverage remains above that baseline level.

All tests and smoke tests run in X minutes or less

The team should monitor the running time of its entire test suite, or at least the smoke test suite, and ensure that it doesn't cross an agreed-upon upper bound.