

Where should you start?

Basic unit/functional tests and associated fixtures.

Buildbot.

Every kind of test (tool) can be useful and informative; no one answer.

(You can always hire us! ;)

Retrofitting projects.

Build out fixtures as you need them. (This can be a *huge* time investment, with little immediate return seen, if done up front.)

Test minimally to start with.

Add tests as you find bugs & add code.

Use code coverage analysis to target untested areas.

Manual test death spiral

- Programmers and customers like new features
- Fixed resources for testing.
- Programmers write unit/functional tests based on canned data set
- Testers focus their manual effort on:
 1. Customer stories for this iteration
 2. Exploratory testing of new code
 3. “Smoke tests” of old code.

So, progressively more code is only tested *once*, if at all.

Lessons learned:

- Technology is relatively easy (compared to people).
- Change the technology, the people will come.
- (corollary) Upper management support is necessary.
- Testers *should not be separate from developers*: shared pain is good.
- Customers want it to “just work”; collaboration seen as expensive.

So, why do software projects fail?

- 1) Lack of solid code testing leads to inability to maintain, extend, and fix existing code (no safety net)
- 2) Failure to communicate clearly with "customer" leads to lack of clear goals: ACCEPTANCE TESTS.

Unit tests ensure that you write the code right.

Acceptance tests ensure that you write the right code.

Remember...

- 1) You're not testing if you don't have automated tests.
- 2) You don't have automated tests if they don't run regularly in a continuous build system.
- 3) Corollary: You don't have automated tests if you're not automatically notified when they fail.