Fog Creek

Articles and interviews on
Business, Design, Programming, Support and other Resources

Taming a Wild, Testless Code Beast – 4 Steps To Improving Test Coverage

By Andre Couturier on Programming

Whether you're working on an existing or new application, all too often you'll find yourself playing catch up when it comes to tests. Soon deploying code changes feels like poking at some ugly, sleeping code monster – you aren't sure what's going to happen, but you know it won't be good. So you pussyfoot around the old code, hoping it won't stir.

Now, this beast may have been of your own creation – you started out with the best of intentions, but development life got in the way and writing tests just never made it to the top of the To-Do list. Or it may have been unleashed on you. You got dumped on to a legacy project with barely any documentation let alone tests. Regardless of how you got there, when faced with a bunch of code and few, if any, tests, at some point you're going to have to try and tame the wild, testless code beast.

Here are the 4 things you should do first to tame the beast and improve test coverage:

1. Add the Right Tests

Start by adding tests in the areas where it is easiest. It's important to consider the order in which you do this, to make sure you get the most out of your scarce resources. You want to start adding tests in the following order:

- **i. Create tests as you fix bugs**

  Add tests to prove that your specific fix is working and keep them running to show that this does not break again. This benefits from being somewhat targeted – you are creating tests in your weakest areas first. The weaker it is (i.e. more bugs) the faster you will build up tests.

- **ii. Create tests with all new features**

  All new features will need to have tests created to prove that the feature works as expected. If you're covering the new aspects of your application, then at least things aren't getting worse.

- **iii. Create tests as you add to old features**

  When updating old features, add tests as you go to show that the older functionality is not breaking in unexpected ways.

- **iv. Create tests in identified risk areas**

  Talk to the Devs and Testers on your team and ask them to point out any weak areas or issues they have experienced. Talk to your support team too – they are an excellent resource with a direct line to the customer. They'll know the areas where your product frequently has issues.

2. Turn on Code Coverage

Code coverage is a tool included in most Continuous Integration systems (or one that can be added with a plugin). These tools will instrument and monitor your code as your tests run to determine how much of your code the tests use. For this to be useful, follow these steps:

- **Start running code coverage against all your code**
- **Get a baseline**

  Find out what the tool can see, where you are currently at etc.

- **Determine areas that you want to exclude.**

  There are likely areas of your code that you don't want to cover – third-party tools, ancient code untouched in years etc.

- **Determine coverage goals**

  Sit down with your team and discuss what your current coverage is and what your ideal can realistically be (usually 90% or above).

- **Work-out steps to improve your coverage**

  You aren't going to fix this problem overnight. Put in place some specific tasks which are going to help you achieve your goals over time.

- **Determine your pass/fail criteria**

  Is staying the same OK, or should it always go up? Is any drop a fail?

- **Run Code Coverage constantly**

Use automation to run your coverage tests, use the criteria you defined as a team to report a pass/fail and do this constantly. It is a lot easier to add tests when the code is still front and center in your mind than later on.

3. Run your Tests on a Scheduled Basis

You should run your tests regularly, on several schedules:

- **Run them on every check-in**

  Use CI tools like Jenkins to run (at least) your unit tests on every check-in. Look at making them run in parallel if they are taking too long to run at this frequency.

- **Run them on every build (package)**

  Depending on how your systems work, your CI infrastructure can help you with this. This could be on every check-in if you are on Continuous Deployment, or every day/week/month that you use. This should be a clean install on a test environment and a full run of all your automated tests.

- **Run them on every deploy**

  You should run all your automated tests against your environments immediately after a deploy.

- **Run them every X days/hours/minutes**

  Run your automation suite as often as you can against your constant environments (Production, Staging etc). This should be daily at least and during ‘off-peak’ times when it won’t interrupt others too much. You can increase the frequency further if your tests are short, don't add too much load to the system etc.

4. Provide a Button to Run the Tests

Again, use a tool like Jenkins to make test runs a self-serve operation. A developer should not have to be delayed by asking a QA person to run a test for them. Get a system in place where your tests will run and give them a button to press. Remove as many barriers for everyone to run the tests as possible. The easier you make it to run them, the more likely they are to be used and run when they should be.

If you follow these steps, then you'll find that over time you'll be able to turn an unwieldy application into something more manageable. By adding tests to the key areas first, then making things easier when you can, you can build confidence around your code changes and deploys.

More on Programming:

YOLO, So Don't Just Code Solo*

Announcing HyperDev - The Developer Playground for Building Full-stack Web Apps, Fast

FogBugz API now supports JSON

dev.life - Interview with Steve Klabnik

The Problems with Open Source (and How to Fix Them) - Interview with Justin Searls

Read the best from 17 years of Fog Creek

Business / Design / Programming / Support / Resources / Everything

Keep up with the latest

you@email.com Subscribe

Search the Blog