3 ways to maintain quality in your unit tests

By Gil Zilberfeld

In this article we'll review three different techniques you and your team can learn to make sure that unit tests don't become a burden to your team and your organization.

1. Do test reviews
2. Separate Unit tests from other types of tests
3. Use Good naming Conventions

Do Test Reviews

We've all heard of code reviews, but many of us don't actually do them. There are many reasons for this, but there's no denying that doing code reviews helps prevent bugs and maintains a higher quality codebase.

Test reviews offer the same benefits, but at a much lower cost. To perform a test review, simply ask one of your colleagues to review the unit tests that you've written, before checking them into the source control system. Have them look at the test names, at the overall readability of the test, whether it actually asserts that something is true or false and that it makes sense in the context of your application.

Why the lower cost? Tests are very small chunks of code with a very specific purpose– to recreate some scenario in the system, using pure code. That makes them much more easily digestible for the reader. Also, because unit tests inherently show the intent of the developer solving a specific problem or requirement, it's much easier to discover problems in the developer's understanding of the requirements or problem domain.

From my experience, test reviews can take a tenth of the amount of time of a code review, maybe even less.

Test reviews can help discover several situations which could lead to failed agile adoption in your team. For example, test reviews can detect if you do indeed do the other practices mentioned in this article – all of which are immensely important to the success of unit testing in your team.

We've also released a free tool for helping out with test reviews if you're working with Visual Studio 2010 –Test Lint. As far as I know, it is the only one of its kind.

'We've all heard of code reviews, but many of us don't actually do them. There are many reasons for this, but there's no denying that doing code reviews helps prevent bugs and maintains a higher quality codebase.'

Separate unit tests from other types of tests

At Typemock, we try to make sure that unit tests are written in their own projects, and not mixed in with other types of tests. The main reason is that we want developers to have an easy single click way to run tests that will always pass. Since unit tests are supposed to only run in memory, and not require any outside configuration, developers who see their unit test project failing will likely pay more attention to that failure and it's less likely to fail for configuration or other false reasons. In other words, having projects made up of just unit tests will give developers less false positives (tests that fail without there actually being a bug in the code).

If we mix unit tests and non-unit tests with each other, we will have more false positives and developers will stop caring about failed tests (it is human nature to say "ah it's OK, it's just a configuration problem. I'll fix it later").

Use good naming conventions

There are two important words here: the naming of the tests has to be a convention used throughout the company or project, and the naming convention has to be good. What's a good naming convention?

One that captures three important key data: The thing(s) under test, the scenario they will be going through, and the expected behavior of the system.

By leaving even one of these out of the equation, you risk that people who read the test will either not understand why it's failing, or even not understanding what it's trying to test in the first place. Tests that are not readable will contribute to the lack of collaboration and willpower from people who actually try to use your unit tests. If people don't want to use your unit tests, or maintain them, you're on a quick road to unit tests being a hassle and an inconvenience to your team, instead of a help.

A good naming convention we like is mentioned in Roy Osherove's book "The Art Of Unit Testing" (Manning, 2009). It follows the following pattern:

```
Public void ThingUnderTest_Scenario_Expected-
Behavior()
{
//…
}
```

The underscores make it more readable and more noticeable if the writer has left out one of the three core parts. You'll also find that in many Behaviour Driven Development (BDD) frameworks, those three key parts are always there, just sometimes in a different arrangement.

Summary

I hope this has shed some light on the things I find most important about real world unit testing. □

AUTHOR PROFILE - GIL ZILBERFELD

With over 15 years of experience in software development, Gil has worked with a range of aspects of software development, from coding to team management, and implementation of processes. Gil presents, blogs and talks about unit testing, and encourages developers from beginners to experienced, to implement unit testing as a core practice in their projects. Gil writes a personal blog, and contributes to the Typemock blog.