Testing is Not Just for Quality Assurance People

Every developer needs to test their code, or have it tested by someone. I don't know about you, but I am horrible at testing my own code. Does this mean that I do not need to test my code? Heck, no! It is always best if you do not rely on your end-user to test your code. This can end up with a very frustrated user, and your user can lose faith in your ability to get their project done. There are several ways you can get your code tested. This article explores a few of these methods for testing and talk about the advantages and disadvantages of each.

Why you should Test your Code

We all know that we need to test our code prior to putting it into production. Testing is your way of ensuring that the code you have written actually works as expected. But it is not enough to check that it works as expected, but also under varying circumstances, and with different inputs. For example, what if someone takes the database offline? Will your code recover gracefully from this exception? Does your code inform the user that there is a problem with the database connection in a friendly, easily-understood manner? All of these questions should be answered in the testing of your code. You need to simulate these exceptions so you can test your exception code.

Performing the act of testing often helps you improve the quality of your code. As you think about the various scenarios that can go wrong, you add additional code to handle these scenarios. This leads to your code being more robust and ultimately more maintainable and user-friendly. You will find that taking time to test your code makes you a better programmer in the long run. Your boss, and your end-users, will appreciate the extra effort.

Problems with the Develop/Test Cycle

There are a lot of disconnects between the development of the code and testing of code. In many cases there is too much code embedded in the user interface (UI). Too much code in the UI makes testing hard because the tools for testing user interfaces aren't very robust. It is also hard to determine if all of the code in the UI has actually been tested. A tester has to know all of the inputs to give the UI to have it run all of the code.

This leads to yet another disconnect; there is too much communication required between a developer and the QA person. Sometimes a developer will forget to tell the QA person about a "special" case. In other cases, a QA person forgets to tell the developer about something that failed, so it does not get caught and ends up as another bug that, hopefully, will be found later.

Another problem is when a developer thinks they have fixed a bug, only to find out that the fix caused a bug in another part of the program. The QA department may not know to test that other part of the program which could mean a bug gets shipped to a customer. Or, the QA department thinks it affects other areas, so they end up doing a lot more regression testing than maybe they need to.

How to Test Your Code

There are many different methods used to test code. Here are some different methods you can use to ensure you are writing quality code.

1. Use a Quality Assurance (QA) person or company
2. Have your end-user test
3. Have a co-worker test
4. Write code to test your code

Of the above listed methods of testing, the first three involve humans, while the last one is a more automated approach. Let's look at these different methods of testing code.

The Human Approach

Having a human test your code does have a lot of advantages. First off a human is going to be able to click on different buttons and test the UI very well. Having different people test your code can be helpful because each person may think of different things to test. In addition it will take less time up front to develop your software.

But just as there are advantages to using humans for testing, there are also disadvantages. Testers find bugs, document them and push them back to you to re-code. After this, the tester has to go back and re-test. You will find this takes a lot more time in the long run. Many times prior to testing someone has to create records in a database, and afterwards get rid of tables, records, or files on disk. This is a very time-consuming and error-prone operation. You also find that the burn-out rate for QA people is very high. You frequently have a high turnover rate in this job function. So there are many down-sides to using humans to perform testing.

The Automated Approach

Instead of having a human do all the testing, you should start to use the great testing tools that are available in Visual Studio to create automated tests. Unit testing has become very prevalent in today's professional software shops. Unit testing simply means you write a program, or most likely a series of programs to test each line of code you write to ensure that it operates properly under all circumstances.

While this sounds like a lot of work, and it can be more work up-front, you will more than make up that time by avoiding multiple regression tests by a human. You can automate the setup and tear down of databases, files, etc. With the correct architecture you can automate almost all the various inputs a human tester would have to enter by hand.

You will also save the time you normally eat up when you do your own testing by pressing F5 to start Visual Studio, wait for the compile process, click through a half dozen menus to finally get to the point where you want to start testing. As you know, this can sometimes take 15 seconds to a minute depending on how large your application is. Instead, you can run one or two tests in just a couple of seconds. This will add up in saving you many hours over the complete development cycle.

As you can see, there are many advantages to an automated approach over a human approach to testing. Automated tests are repeatable as opposed to having a human who might forget to test something. You will end up with more of your code tested because things won't be forgotten. Because you will be forced to think of how to test your code while you are writing it, you will write better code. You will also save time on the setup and the tear down of the tests since these tasks can also be automated.

Of course, there are disadvantages to the automated approach as well. First of all, it does take more time up-front to develop these tests. Automated tests are only as good as the person that develops the tests. The tools to test user interfaces are not too great at this time without spending a fortune on third party testing tools. Of course, if you are using good N-Tier techniques, MVC, MVP, MVVM or similar design patterns, there should be very little UI for you to test anyway.

Which Approach Should You Use?

The big question is this: do you use automated unit tests or do use a QA department? I still think you need a little of both. You want to get as many automated tests as you possibly can as this will save a lot of regression testing. Then the QA department can focus more on system testing, workflow and ensuring the correct data is flowing all the way through the application. The QA department can also focus on the UI features that are difficult to automate. A good mix of the two will go a long way toward making your applications much more robust and error free.

Architecting Your Code for Testing

As you are coding your application there are things you can do to prepare to take advantage of unit testing. Correctly architecting an application will do wonders for re-usability, readability, upgradeability, maintainability and testability. Take advantage of the design patterns that exist today such as MVC, MVP, MVVM or derivatives thereof. All of these patterns help you remove code from the user interface layer of your application. Removing code from the UI layer is this is the single best thing you can do to have a well architected, easily testable application.

Your UI code should strive to just call or bind to classes. Each class you write is where all the logic is contained for your application. Moving all application logic into classes, you take advantage of the various unit testing tools in Visual Studio. All of these classes should be combined into assemblies to aid not only in testing, but also in re-usability.

Other techniques you should employ are to make your methods as small as possible. A method should do one thing and one thing only. This makes testing that method easy as you most likely will only need to write one method to test that method. It is better to have many smaller methods and smaller tests, than one large method with lots of tests against that one method. One last technique is to use code generation tools for generating your CRUD layer. Create, Read, Update and Delete logic is employed in almost all business applications and is very standard code. There is no reason to hand-write this code with all of the tools available today to generate it. And, generated code is generally code you do not need to test. If the code generator is good, it will generate bullet proof code every time.

Visual Studio Testing Tools

Beginning with Visual Studio 2008, unit testing tools were added to almost all SKUs of Visual Studio. This means you do not need to purchase Team Foundation Server to be able to use unit testing. It also means you do not have to use NUnit or some other third-party application for unit testing. It is available to you right out of the box. Of course, the TFS version of Visual Studio has many more features, but for basic unit testing, you will find everything you need in the normal versions.

Summary

Unit testing is something every developer should employ as a part of their software development life cycle. Not only will help you improve the quality of your code, it will save you time and money in the long run. As you focus on architecting your applications for unit testing, you will find that your code will become re-usable, maintainable and more robust. Take some time to learn about the unit testing tools built-in to the Visual Studio suite of products and you will be on your way to better productivity.