The Importance of Independent Testing

Amy Gander
University of Northern Iowa
Department of Computer Science
Cedar Falls, IA 50614
Gandera3081@uni.edu

Computer science students without industrial experience do not understand that software development and testing are two separate and opposite sides of the software development process. In the academic world, one does not often have a team of people that can divide the development and testing tasks. Most often there is only one person to do both tasks. The person, who writes the code, tests the code. Now, you might say, "What is wrong with that?" The problem is that testers and developers have different views of the problem, different approached to testing the product, and different goals. If only the developers test their own programs, the program and end users of the program suffer from inadequate testing. It is necessary to have separate development and testing groups. Once the groups are established, the interaction between them is not as straight forward as one might think. It is very important for the success of the product or project that these two groups are able to communicate effectively. Although the groups are working for the release of a hopefully well-received program, their goals in the immediate scheme of things are quite different. The developers main goal is to write executable code and the testers main goal is to identify faults in the code. This paper covers the differences between the testing done by programmers or owners of a product and testers and the communication problems that these two groups face.

What "Professionals" like to tell us

Most books that the student of computer science reads on software development don't even touch the fact that who does the testing makes a difference. While books devoted to software testing seem to all agree that having an independent tester is important to the success of a product, but only a few cover it in any detail. As Boris Beizer states, half a programmer's professional life is testing, yet "less than 5% of all the programmer's education will be devoted to testing."(1) In this case the software engineers must turn to the classics. Glenford Myers' The Art of Software Testing, though written in 1979, is a good book for the basic testing principles in today's software development world. On the topic of independent testers Myers states that a developer's state of mind is that of a person building and creating and a tester's state of mind is that for finding flaws, tearing it down so to speak:

> Testing is a destructive process. In other words it is extremely difficult, after a programmer has been constructive while designing and coding a program, to suddenly, overnight, change his or her perspective and attempt to form a completely deconstructive frame of mind toward the program. Most programmers cannot test their own programs, because they cannot bring themselves to form the necessary mental attitude of wanting to expose errors. (4)

Beizer might add to this Myers statement with, "the tester in you must be suspicious, uncompromising, hostile, and compulsively obsessed with destroying, utterly destroying, the programmer's software." Clearly programmers have the wrong attitude for finding errors in their own programs and not finding these errors may be due to a misunderstanding that the programmer has of the problem statement or specification. (1)

Even a book called The Complete Guide to Software Testing by Bill Hetzel has only a few words on the importance of independent testers. Hetzel points out that programmers are too close to their programs to test them:

> Having participated in developing an approach, we gain so much 'ownership' of it that it is impossible to view it fairly against competing approaches. This is just one example of many that could be cited in support of the principle that testing must be done independently if it is to be fully effective in measuring software quality. (2)

The independent tester initially views the program as the beginner user would see it, Beizer refers to such a tester as a "naïve tester." Such a tester has no prior ideas about what is and is not possible in a program, and therefore will "design tests that the program's designer would never think of." The designer may be more efficient at testing, "but also (has) blindness to missing functions and strange cases." When a designer sets about making and executing tests for their programs, these tests "are by nature biased toward structural considerations and therefore suffer the limitations of structural testing."(1)

According to William Perry, testing is something that should be performed by many people, however none of these people are the developer:

> Who should perform computer software testing? The answer to this question is the stakeholders (i.e. those having an interest in the success of the program) in the function being automated. If it is a payroll application, then all the employees responsible for payroll should be involved. This can include the individuals responsible for preparing payroll, collecting payroll information, and perhaps even sampling of those who would be paid. It is the quality of the testing group that is significantly more important than the quality of time expended. Someone knowing the business function can quickly provide the verification that might take a clerical person hours or even days to do less effectively. (5)

Boris Beizer and David Gelperin wrote of the growth of testing as a career. They say that testing will be more than just a part of quality assurance. (3) This shows that developer and tester are different jobs with different skill sets. For example, the developer's skills revolve around the creation of a product, while a tester's skills revolve around finding the products faults so that they may be fixed.

In the Real World

Experience is the best of all teachers. One can read literature, like this paper, about the importance of independent testers, but until one experiences it, the words lack meaning. I hope that the tale of my experiences while doing an internship with a major microprocessor manufacturer helps to make this principle one to remember in your future career.

For my internship, I tested software that was "ready" to be distributed to the company's internal environment. Some of the software was purchased from other companies and some was developed internally. My group was developed as one last test point before the products were distributed company wide. By the time the software had reached my group, the developers have tested their program and assumed it is ready foe deployment.

The problem with a developer's machine.

The first thing that I learned was that a developer's computer has the latest and greatest software, fixes and upgrades, which the average target computer's environment user does not. Having tested the product thoroughly on his or her own machine, the developer is ready to deploy the software to the entire environment. The problem with this is that the developer is usually running

the software on only one operating system build (version) and is using a server that is only supporting a handful of developers. This is not a true test of the program's ability to perform on machines of different processor speeds and operating system builds. The developer, who does not have the resources for such testing, should request testing from an outside source to ensure that the program is going to run in the environment with minimal errors. Environments are much more complex than the developer's personal environment that he has created for himself; Networks have more than just a handful of people on them. Running things off of the network slows the response times of all programs, but if the program is not efficient enough, it should not be deployed. It had happened more than once at the company that I was working for that a developer had been working with a .dll file that was not in the environment yet. This and the fact that the programs would not be tested for the stress that a well traveled network would produce are reasons for independent testing before deployment. For, the cost of the deployment, removal, rewriting, and TESTING is greater than the cost of testing in the first place. In this case the developer had become too far removed from the target environment.

Trusting the Professionals Or Not

In many cases programs come from outside programmers and updates are expected. A good example of this is anti-virus software. As a new virus comes out, so do the programs to protect your data from it. Many companies gladly update their anti-virus software each month without doing some of their own testing. During my internship I was asked to do routine testing of the company's virus detection software. I recall the day that the latest and greatest came into our lab and we saved the company some money. What had happed was that the anti-virus people had sent out a product that was less than complete. It had failed to detect two viruses that it had cleaned in previous versions. This could have been disastrous for the company if it had blindly updated the software. In the end the problem was escalated to the anti-virus company and recognition awards and thanks were passed down. That anti-virus company would have avoided that embarrassing failure if they had employed independent testers. It was good to know that the people that I was working for were not taken chances with their data.

Gonna Getcha One Way or Another

Many programmers test there programs locally on there own machine and they seem to forget that the program has to get to the user somehow. Some user will have the program installed from a CD, which is the most like how the developer tests his or her program, however in many cases the program is going to be pushed down from a network, offered on the web, or passed around on a networked drive. When the developer is not prepared for this, files can be lost in the distribution method. Independent testing resources are the best way to go in this case, because the have solve the problem of the "developer's machine" and the will have the network for testing the push from the network without involving the production server. The developer forgets that there is more than one way to receive a program and if not all methods are tested, one of them is bound to fail.

Mistakes, what Mistakes

The programmer knows his or her code inside out, however they also know how to get around flaws in the program. The best example that I can give is for installation of a new program. Programmers are a creative lot when it comes to passing their own product. There was a programmer that was confident that his product was ready for deployment, however during testing we were not able to install the program using the setup.exe file. His explanation of this was that there was a registry key that needed to be modified before the setup could be run successfully. The testing facility was not about to pass a product that required the user to edit the registry. The programmer honestly thought that this was an okay work around for the product's failure. Outside testing is best in this situation, because the developer doesn't realize that the mistake in the installation was something that needed to be fixed within his own program.

You mean to tell me that not all data is ideal?!

Developers do not allow a lot of time to test their programs. Because of this they often just test with ideal data the fits into the specified parameters. In other words the developer is only testing the program to see if it is doing what it should be doing, and they are not testing to see that the program is doing what it shouldn't be doing. A program needs to be tested outside its parameters to see if it gives the proper response or error message to such data. To get the product out to the environment on time, programmers run test that they already know will work. A test engineer once said to me that "if the program only works on a day when the temperature outside is 80 degrees, the programmer will wait for an 80 degree day on which to test." This may seems extreme, but it is something that I have observed in my internship.

It's All Mine

When working with a project for any amount of time, those involved in getting the project to deployment have developed a biased toward the project. When those accountable for getting the project out to the customer on time and on budget are the ones to test the program, it is certain that certain things will not be checked in order to save time. I remember on one occasion the programmers failed to test the printing option, because "it had always worked before." This program was for calculating certain data and putting it into report form, without the printing option this program became useless. So, because of their past experience with the program and their desire to get the product to deployment, the product came out on time and on budget, but not with the quality that it needed to be beneficial to the company. Testers are not biased to the product and will test the product fully, because it is their job to find the faults, before it goes to deployment.

New Tricks? What do you think that I am?

Writing a test script can be a daunting task. However, once a developer writes a script, he or she does not wish to "reinvent the wheel." Because of this, many test scripts written by developers circulate through the testing process over and over without changing. Recycling scripts could lead to failing to update scripts with the addition of a new feature or the removal of another feature. This is a danger of not having an independent tester. Independent testers can add new strategies to testing, and have a stake in modifying the script so that the product comes out with as few bugs as possible.

Talking to the Enemy

Programmers and testers are not enemies, but miscommunications between these two groups can start a war. It has happened more than once that a tester reports a problem, the programmer can't reproduce the error, and the program is pushed into production. As a tester, patience and details are the keys to success in communication with programmers.

Patience "Spoonful of Sugar"

Testers need to look at the bug (problem) from the programmer's point of view. The programmer is building the program, he or she owns that code. Many times in testing, where finding errors is the goal, testers forget that the program works in many ways and is not completely broken, so don't act that way. A good approach for a tester is to report the errors along with some of the good points that the program possesses, so that the programmer doesn't feel so threaten by what you need to say. Remember, the programmer's job is to get the program to production, so errors are in many ways bad news for them.

Details

Testing is about finding errors, however it is not the tester's job to guess as to where these errors occur. As a tester, one needs to pin point where in the program that the errors are occurring. This is accomplished by writing all the details of the errors that occurred. These include:

1. Steps to getting the error
2. Failed steps to getting the error while trying to reproduce the error
3. Error messages produced by program itself and the operating system… or other programs
4. If the error is not reproducible, the exact conditions under which the error was found
5. Reproduce the error in as many ways as possible

All of these things help in helping the programmer to pin point the error's location in the code.

Conclusion

As one can see, the independent tester is an essential key to the success of any software or hardware product. As a programmer one need to step back from the product and accept criticism from people outside the world of code in order to improve the product's quality. As a tester, one needs to realize that the product is his or her product too, not just the developer's product. When these two groups, testers and developers, work well together a great product can be the only result.

References

1. Beizer, Boris. *Software Testing Techniques.* Van Nostrand Reinhold, New York., 1990. pp. 11-12.
2. Hentzel, Bill. *The Complete Guide to Software Testing.* QED Information Sciences, Wellesley, Mass., 1984. pp. 26-27,190.
3. Hentzel, Bill, Gelperin, David. The Growth of Software Testing. *Communications of the ACM 31,* 6 (1988), 690.
4. Myers, Glenford J. *The Art of Software Testing.* John Wiley & Sons, New York., 1979. pp. 12-13,128.
5. Perry, William. *How to Test Software Packages: A Step by Step Guide to Assuming They Do What You Want.* John Wiley & Sons, New York, 1986.