Becoming a developer, who is less disturbed by a tester! (In 3 pages)

Pradeep Soundararajan.
Consulting Tester, Satisfice Inc &
Test Manager – TriVium Systems, Bangalore
pradeeps@triviumsys.com - +91-98451-76817 – Pradeep.srajan@gmail.com
http://testertested.blogspot.com

-- -- ---- -- ---- -- ---- -- ---- -- ---- -- ---- -- ---- -- ---- -- ---- -- ---- -- ---- -- ---- -- ---- -- --

I was lucky to have interacted with one of the first few computer programmers, to my knowledge on Earth. Jerry Weinberg, today one of the greatest consultants of the world was a developer at IBM during the early days of computer age. I learned from him that they weren't doing much testing or any testing those days. Programs that compiled successfully with no errors were considered to be fit candidates for release. As users increased and so were their demands and ideas to use the products, testing started gaining importance to avoid getting sued from users and to survive the market competition.

Today development and release are so different from those times. Although the code compiles with no errors and exceptions, testers do come out with crashes, hangs and what not?

Analyzing the situations I have been in my career so far, I conjecture that developers being interrupted by testers too frequently can impact the delivery dates and puts pressure first on the developer and then on the tester.

Not many developers, to my knowledge, know the secret of keeping testers away in order to show more output to their managers. *(More output –> More appraisal points –> More money –> More growth –> More you want such things to happen in your life)*

An organization had hired me in past to consult with developers to solve such a problem. By proposing solutions to the developers, indirectly I knew I was adding pressure to testers. Overall the manager who got me there was happy.

All I did is to teach developers; how to test, like a tester!

As a part of getting coached from experts, I was taught a secret from the guru's to observe the pattern of mistakes that developers usually do with their code. When fishing for important problems, I look for patterns in the problems that come out of individual developer's code and this certainly helps me in finding important problems, quickly, when a specific developer with whom I work makes a release. I also try to use a complete set of test heuristics* and oracles** to find bugs that can help in taking a better decision about the release.

* Heuristics – *a fallible method to solve problems.*
** Oracles – *(not the database) are principle or mechanism by which we (humans) identify problems.*

"Dear developers, there are testers around you, who look for pattern of your common programming mistakes and use that as a clue apart from using powerful test heuristics to fish some important problems, quickly and hence delay your further development plans. So what would you want to do about that?"

Knowing the pattern of your mistakes and **unit testing with the powerful test heuristics** is one way, you could achieve in becoming a developer, who is less disturbed by a tester.

I don't want to help you in knowing the pattern of your mistakes *(unless you ask me to do it and I do it, to help you become a better developer and to better my test expertise)* but I certainly can spill the secret of powerful test heuristics that can help you Unit test your code!

Ever wondered, "How can testers think that way?" Keep a watch on their heuristics.

Not many testers, to my knowledge hit the scripted test case when you (developer) make a release to them. They explore! If they spot crashes or hangs during their exploration, they might say, "Smoke/Sanity test failed" which means, "*Developers, spend your late evening and night fixing this while I try to help you continue the good work, next week"*

So, when they can explore, you too can! The more you give time to testers while you are fixing, the more they find ways to crash your code. **How can you as a developer try exploring in a better way than him?**

When you write the code, compile it (and see no errors and exceptions), you start testing the code. Keep this list (preferably a hard copy) with you (or in your PC /cubicle).Before you make a release to the tester, ensure you do **at least three** tests in each of the category:

1. **Functionality:** The tester usually attacks your code based on the logic. A customer usually and unusually attacks your code based on a scenario [his real time usage]
2. **Load** – Any true tester loves to show you a crash in your code and he certainly does when you develop with a mindset "No user would do that".
3. **Stress** – Many testers to my knowledge, use a shoe to keeps the enter button pressed when they are off to a coffee break. Poor code of yours keeps crying and then maybe passes the crying to you when your manager says, "fix it soon".
4. **Scalability** – I too don't know why I love seeing if your code can handle more number of users, interrupts, values, instances… You might want to check, what fun does a tester gets when he plays around with scalability part of your code.
5. **Security** – Good testers in security domain are good hackers. Every software that we build has some level of security and testers in the context of becoming good try to hack your code with uploading virus, spyware, password loggers, input

constraint attacks, SQL injection attacks to practice becoming a better tester. Do you want to allow him log such bugs?

6. **Performance** – This is one element that testers are irritated with. If you develop a code that performs too badly, in taking too much time to respond to a tester's activity, you are motivating him to work more and log great bugs that can keep you awake in the nights and get up at 6 AM on a Saturday morning to fix those bugs. Would you want to motivate testers to do that to you?
7. **Fields and Values –** The quickest bug a tester can find is with your field and value validations. The quicker he finds, the early he interrupts you, do you love early interruptions?
8. **Tools –** A tool does help a tester to generate some critical test data that helps him find some important problems more quickly than you anticipate. I am sure that unless you invite him for a good meal in a great place or a drink in the best pub, he is not going to let you know the secret tools.
9. **Reliability – "**Code is your baby, not ours!" We love putting your babies to long haul work (of course, it's like child labor activity) over the weekends and during the nights where we sleep well thinking that your code has crashed at mid-night. Would you want us to come back on a Monday morning and start our work by sending an e-mail "Your child was crying, didn't you hear?"
10. **Operations and Time –** We operate your code to understand the operations it undergoes and we also keep our chronograph running. When we say "Operation success", we mean… we have found bugs with operations and there might be time as another dimension associated with the failure of the operations of your code.
11. **Structure –** Testers who know little about programming (as programmers know little about testing) model your code in mind as huge concrete building structures. Any twin tower that you build is prone to Al-Tester attack and it comes down.
12. **Usability –** Testers Google a lot and keep information on common usability mistakes done by developers and straight away pin point at the article and your code. I guess Google is available for you too. Hurry up before testers search, "*Is my developer googling?*"
13. **The Last one –** What makes you think that there aren't any more points that can help you do a great unit testing and keep testers to disturb you lesser than now?

There are lots of other points that I know, which I don't want to share to ensure I (as a tester) keep finding bugs and get paid, more!
The better you unit test, the pressure is one me to find more bugs, the better it is for your company – who expects such a thing to happen to make the world a "customer happy place".

I must admit that by reading (or thinking through) this article you (as a developer) can never become a developer who is less disturbed by a tester. It takes passion to develop great quality products and practice using the above heuristics, which over a period of time can really make you "Becoming a developer, who is less disturbed by a tester".

"Pradeep's first language is not English--his first language appears to be testing." -- Michael Bolton