2012

What If…

50+ tips to improve tester-programmer relationship

Relationships matter…

Ajay Balamurugadas

http://EnjoyTesting.blogspot.com

12/26/2012

What if…

Hello Reader,

Special thanks to you for buying this book. My previous books have concentrated on improving testing skills. This book brings into picture a very important person - the programmer & the programming team. Each one of us might have the experience of working with at least one tough programmer. Some programmers are very friendly and help us with finding bugs. Some of them are very strict with their deliverables and do not respond to any queries outside office hours. Some hardly talk to you unless you ask them a question. There are different types of programmers and bring in variety to our testing challenges.

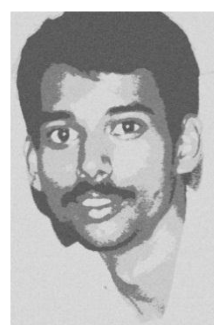
As I write this book, I have completed over six years of software testing and interacting with multiple programmers across different projects within and outside the company. With a rich experience of working with tough programmers, I write this book to help you. I am very thankful to the programmers, especially the tough ones, as they are the ones who have taught me the most important lessons I had to learn to improve in my testing career.

This book is close to my heart as this is my gift to my father. After the first three books, I got a bit busy but got involved in challenging work, especially with challenging programmers. Feel free to share your experience with programmers. Hope you like this book. Details on how to buy my other three books are up on my blog. Thanks once again.

Please feel free to contact me via email, Skype, Twitter or GTalk.

Email: ajay184f@gmail.com Blog: www.EnjoyTesting.blogspot.com

Skype/Twitter/GTalk: **ajay184f**



Foreword

Ajay Balamurugadas is probably the most 'passionate' software tester that I have come across. So it is not a surprise to me that his books about testing are consistently well received by the software community and he has come up with his 4th book '50+ tips to improve tester-programmer relationship' in the 'What If' series in such a short span of time. I express my heartfelt thanks to Ajay for allowing me to be one of the first readers of this book and also to write a foreword.

Let me start with a small story from my own experience. There was this new tester assigned to test the module I had programmed and it just went like how it happens normally. Both of us having our own agenda and goals about our work, the module suffered with plenty of issues, arguments back and forth, blame games and ended up being delayed with low quality. A year later, while we were into other stuff, the tester and me had a personal connect and had become very good friends. Then came another assignment of working together - to program and test – and this time it was a lot different. We worked together, for a common goal, no arguments amongst us, end customer was the focal point of our work and it had worked beautifully. Not a surprise, the module was delivered ahead of time with very well appreciated quality. So much to say that 'Relationship matters..' as Ajay puts it aptly in the title of this book. Even if there isn't an existing relationship between the tester and programmer, this book comes up with how to develop the relationship between the tester and programmer, with beautifully structured 50 tips that ensures the success of what they deliver together.

Opening with describing 'Bug - The weapon' the book talks very intelligently on how 'bugs' can be the focal point around which the relationship of testers/programmers make or break. Some very good tips follow on how to treat bugs and its attributes. Not to miss a very interesting blog post link on 'not to insult programmers'.

Then comes the section of 'Assist the programmer'. The first reaction that came to my mind after reading this was 'what a phrase...!' It is very nicely put by Ajay on how testers can assist the programmer - while it can also be applied vice-versa - and the tips mentioned here in this context goes really a long way in making the relationship of tester developer a great one. The book also talks about the role played by Communication and Appreciation in building and enhancing the quality of the collaboration needed in a software development environment.

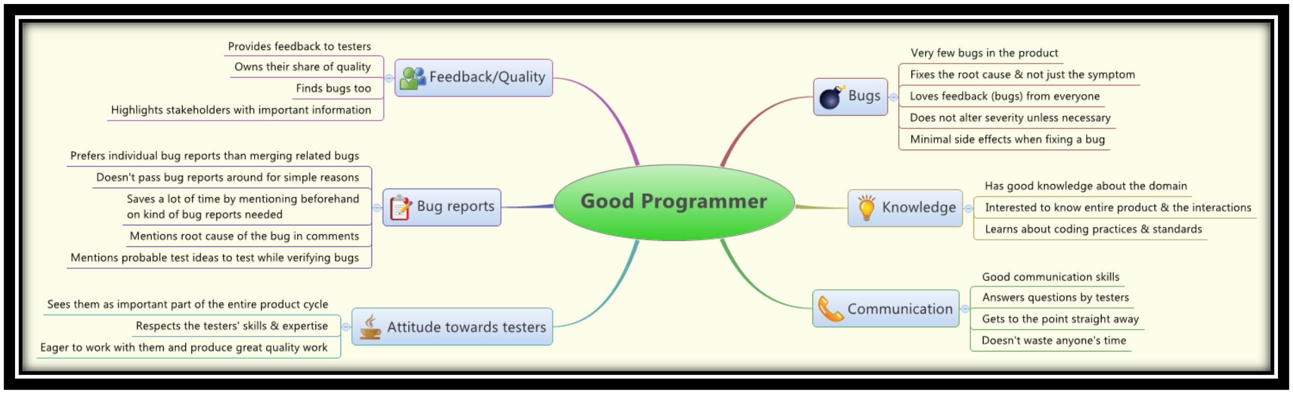
Towards the end of the book, as rightly put, Attitude is one of the most important traits and there are tips on what kind of attitude should the testers possess towards providing a quality effort on testing in a project. Though this book is a must for every new software tester, I would recommend it to all the programmers out there as well, since Ajay brings out several key aspects that go into building the relationship and bonding between the two and also helps understand the psyche of each other on what goes into delivering their respective work. This book is an excellent continuation to the 'What If' series by Ajay and would look forward to more such artifacts by Ajay on various facets of software testing.

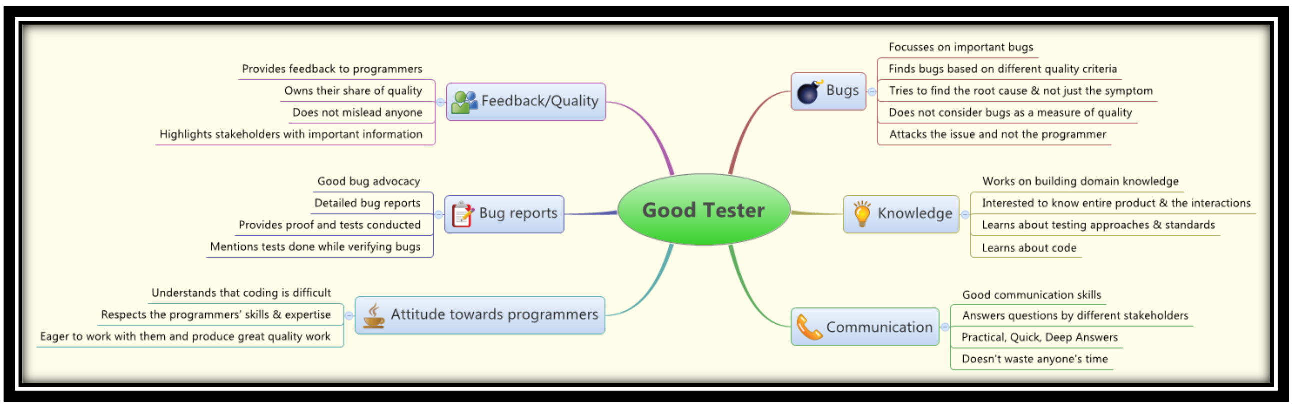
Chaitanya Ram

Software Developer and currently Engineering R&D Manager, Philips Innovation Campus, Bangalore

Contents

The Battle Continues… Mind map





Few Terms

Bug, Issue, Problem, Defect - According to me, these are different terms used to mean the same thing -

“A problem in the software”. I prefer to call it as ‘bug’ throughout the book even though I might use them interchangeably.

Programmer, Developer - A programmer is one who codes. Usually known as developer too, I prefer to use the term ‘programmer’.

Sev 1 - Severity 1 bugs. The bugs with the highest severity - a crash, missing functionality, incorrect functionality causing data loss, corruption and will cause a lot of money fixing the loss.

Bug Report, Problem Report, Incident Report - A report highlighting the bug.

Bugs - The weapon

Most of the tester-programmer relationship starts with a bug. There have been many instances when I have started interacting with a programmer for the first time over a bug. Maybe, they did not understand the bug or did not agree with some elements of the bug. So, let me start with the "Bugs" as the first section.

Tip 1: Hunt for important bugs first

No one likes missing important bugs. Programmers like to know about the Sev 1 bugs before the other bugs. As a tester, your reputation will be on the firing line when you log unimportant bugs. Learn to find bugs that matter to the business. A bug that talks about a crash, missing functionality or incorrect functionality will be preferred over username not accepting more than 200 characters.

Think of bugs which will get the programmers rush to your desk to see the bug. Hunt for such bugs as soon as you get the product. Well, you might like to do a round of sympathetic testing and then hunt for important bugs. Read more about sympathetic testing here:
http://enjoytesting.blogspot.in/2012/03/sympathetic-testing.html

James Bach gave me few tips to find Sev 1 bugs when I was at Orcas Island for Rapid Test Intensive (RTI):

- Boost up your test lab
- Sev 1 bugs are obvious
- Test the very assumptions
- Raise the standard of quality of the product
- Learn about the product internals and the underneath technology

Tip 2: Log them early

Though testers might be considered to always bring bad news, programmers prefer the news early. They are definitely happy when they get more time to fix the bugs. Suppose you find a bug but log it after few hours or few days, you are hiding valuable information from many people. You are also indirectly giving more time for other bugs to hide. If many bugs are found early, programmers will plan their schedule accordingly and most probably give enough time and effort to each bug fix. This helps the product, project and indirectly you. As there is no hurry to fix the bugs, there is less chance of re-opens saving you a lot of time usually spent in regression.

What if something crops up at the last minute? Its better to get the bugs out as early as possible in the cycle and save time, cost and effort for the project.

Tip 3: Focus on root cause and not on symptom

Very few testers understand the difference between a symptom of a bug and the root cause of the bug. You might log a bug as soon as you find an unexpected behavior but are you sure that it is the only problem? What if the symptom is hiding the real problem?

Whenever you find something unexpected, pause for a minute and think. Is this the only problem? Can there be more severe manifestation of this issue? BBST Bug Advocacy course http://testingeducation.org/BBST/bugadvocacy/ highlights few important follow-up steps on finding a bug - http://testingeducation.org/BBST/bugadvocacy/BugAdvocacy2008.pdf#page=47

Tip 4: Assign severity carefully

Severity and Priority are two fields in a bug report which are debated very often.

Severity is how severe the problem is and relates to the impact on the users. Priority is about when the bug will be fixed. It highlights the order of priority for programmers. Testers should assign severity and not priority. Before assigning severity to any bug, think carefully. Is the bug really a critical bug? Or is it a major bug but you wanted to impress your manager and increased the severity of the bug?

Tip 5: It is not a competition

Testers need to remember that they are not in some sort of competition against the programmers. I have seen few testers think of testing as a competition. They focus on the number of bugs alone and forget about test coverage. Do you feel that you are in a testing project or some bug hunting competition? A tester should never feel that they are competing against any programmer. The moment one considers programmers as competitors, the chances of them competing against you also increases. Instead of working together towards one goal, competing against each other affects the overall quality of the product.

Tip 6: Never insult anyone

During CAST 2010 testing competition, I was part of Miagi-Do testing team. Our test report was deemed to be insulting the programmer. More details here: http://www.satisfice.com/blog/archives/605. Never commit such a mistake. Do not write any bug report that insults the programmer. Write about the product but never about the programmer. Do not get personal with your bug reports.

Tip 7: Do not make them look bad

Sometimes, managers look at bug counts and make decisions. Suppose the project managers look at the bug database on every Friday, do not log bugs on Thursday evening. Log it after the evaluation by the manager so that programmers are not blamed for poor performance. It is easy for anyone to notice that you log bugs to prove that programmers are not delivering good quality code. The behavior of logging bugs just before the evaluation will definitely lower your reputation among project members.

As a tester, you have no rights to make someone look bad. Remember that this is one mistake many testers do it unintentionally as well.

Tip 8: Investigate intermittent issues well.

Every now and then, we testers encounter bugs which are hard to replicate. It gives a lot of satisfaction to the tester when (s) he is able to replicate the bug. More than the testers, the programmers will be more satisfied. It is usually hard to solve an intermittent problem compared to the one that is easily replicable. Some testers take pride in investigating such issues more compared to those who think that the testers' job ends at pointing out the issues.

If you want to have a better relationship with your programmer, it's time to hone your bug investigation skills - especially the ones surrounding the intermittent problems. I refer to the following two excellent resources on this topic.

http://testingeducation.org/BBST/bugadvocacy/BugAdvocacy2008.pdf#page=77

www.satisfice.com/blog/archives/34

Tip 9: Try to avoid duplicate bugs

There are instances when the programmers get very upset and dealing with duplicate bugs is one such scenario. Imagine multiple people asking you the same question one after the other. It is the same scenario with duplicate bugs for programmers. Instead of working on new bugs, they have to spend time on the issue they have already worked on. As a tester, we can help the programmers by going through the bug repository and check if the issue is already logged. This is a useful practice when multiple testers have worked on the same feature at different times of the project. If it takes a lot of time to go through the bug database, ignore this tip and log the bug.

Considering the cost of avoiding a duplicate bug to the value gained by finding another bug, it is sometimes better to just log the bug. Let the programmer comment that it is a duplicate of another bug. But again, try to spend as much time as you can to avoid duplication.

Tip 10: Highlight related bugs

There are two points I want to highlight regarding 'related bugs'. When we find bugs, programmers will like to know the related bugs. What are the similar bugs? Are there other bugs which display similar symptoms? If you suspect that some other bug or a bug fix has introduced a bug, feel free to highlight it. Which other features display the same bug? Which other features are affected by the same bug? As a tester, we can help the programmers by highlighting related bugs. It saves a lot of time for everyone. Instead of modifying the code for n bugs, it is better to modify it once keeping n scenarios in mind.

Tip 11: Close bugs quickly

Nothing irritates a programmer than watching a bug stay in 'Ready to be tested' state for a long period. (S)he spends a lot of time fixing a bug and if the tester does not test the bug quickly, it gives the impression that the tester does not care for the programmer's time. Closing bugs quickly helps in knowing if the bug is really fixed and if it has introduced any new bugs. In most of the projects, decisions are based on the bugs in progress. As the bugs waiting for verification can turn into either a closed bug or re-opened bug and affect the decision, they need to be closed early. Closing bugs late in the cycle also gives more time for other bugs to hide.

Tip 12: Provide appropriate comments while closing bugs

As we have realized that closing bugs early helps everyone, we should also note that appropriate comments by the tester during bug verification helps. It lets the programmer know of scenarios tested. If a scenario is missed, the tester can be informed immediately. Many testers write a one word - 'Tested' and close the bug. If the bug is closed, it is understood that it was tested. Folks are more interested to know the tests conducted during bug verification. The version in which the bug was tested, the credentials used and so on.

Assist the programmer

Testers who assist the programmer in their tasks are more likely to receive similar help from them during testing. It is natural that if you help someone when in need, they will help you when you need help. It is appropriate in software projects as testers can assist the programmers well before the product is ready for testing. Two pairs of eyes are in most of the cases helpful. As a tester, one can highlight test scenarios missed by the programmer.

Tip 13: Help them even before testing cycle starts

How does the new code look in different environments? Different browsers, browser versions, screen resolutions, different mobile devices? As a tester, can you take a look at their code and provide them early feedback? There are many projects where tester and programmer sit next to each other and help build a great quality product. The feedback loop is short and quicker.

As a tester, when you provide feedback to the programmer even before they have completed their coding, it helps them fix issues quicker. As they are still working on the code, it saves time as they understand which part of the code is causing the issue. Also, the side effect of the fix is very minimal in most of the cases. As they say, the sooner (in the cycle) the bugs are found and fixed, lower the cost induced by the bug.

Tip 14: Ignore the bugs, concentrate on what is working

Sometimes programmers are busy developing a prototype and not worried about high quality of the prototype. The objective is to visually compare two contrasting designs and not focus on bugs. Finding bugs is the last thing programmers expect from the tester at this stage. As a tester, can you ignore the bugs and focus on what is working? Can you ignore the blockers and still give feedback even though the end to end is totally broken? Some testers want each feature to be working before they can even start looking at the product. Are you one such tester?

Understand the objective of the task. Is it finding bugs or providing the feedback?

Tip 15: Get them the tools, remove the blockers

Usually programmers do not have access to the test environment or the machines testers use for testing. If a programmer wants to check a part of his code but doesn't have the required resources, help him. Give him access to resources reserved for testing team. If the team is stuck because of some issue out of their control, check if you can be of some help. Can you talk to your manager and get it escalated or resolved? Do you know anyone who can solve the issue?

Programmers will be busy with developing the product and might not focus much on resolving the blockers. It could be as simple as 'need for IE 7 browser, iPad of iOS 4.0.1 version or uploading 100 files on a particular website'. As a tester, you try to save time & effort of the programmer, thereby improving the relationship between you and the programmer. If you have different types of test files, do share them so that bugs can be found earlier.

Tip 16: Share the test ideas

As testers and programmers, we have different experiences and usually different style of thought process. Pair up with a programmer for a testing session and observe how different each one of us thinks in terms of the test ideas. Based on my experience, few programmers have helped me with some great test ideas which I did not think of. At the same time, as a tester I was able to think of scenarios which they had missed. Make use of this difference in thought process. While the product is being developed, sit together with the programmer and share test ideas. Make a list of these test ideas. Some of them might not be applicable when the product is ready but do not restrict yourself from thinking of different test ideas.

Tip 17: Share the list of probable bugs

The product is not yet ready. The requirement document is ready. What do you do? I was in a similar situation once. I prepared a list of probable bugs. The list was my reference checklist once testing cycle began. The programmers were happy that they knew the kind of bugs I might find even before the testing cycle started.

http://testingcircus.com/Documents/Testing-Circus-Vol3-Edition8-August-2012.pdf#page=9

Imagine how time saving it is when most of the straight forward bugs are found by the programmers themselves. This checklist can help them do that. Once the bugs found have been fixed, you can start testing the product. You can dive deep into the product and not spend time highlighting obvious issues.

Communication

As a tester, you might be very good at finding bugs and testing the product. Your bug report is one of the major deliverables to major stakeholders. In addition to bug reports, testers will be asked many questions by different teams in different meetings. Are you prepared?

Tip 18: Work on your skills - written and spoken

What is the use if you have a great point but you are unable to express it clearly? Communication is one of the important skills a professional must develop in additional to the technical skills. There is never a better time to work on the skills than now. As testers, we need to communicate regularly about the status of the product, project and any issues we face. There are multiple meetings that we need to attend. If our communication skill is not up to the mark, our reputation is affected to a great extent.

Tip 19: Learn to talk in their language

It is very interesting to talk to programmers, especially when they have their own terminology to describe the product. As a tester, one can learn more about the product by talking to the programmer. You will get to know the difference between a frame, panel, section and pane. While testers talk in terms of features, options and values, the programmers might refer to the same terms as sections, class, CSS, panes or models. If you really want to improve the relationship, talk in their language. Do not insist that the programmer should use the terms used by the testing team. Instead, talk in their language.

Focus on completing the task rather than forcing everyone to follow one terminology.

Tip 20: Keep it Simple & Short (KISS)

One of the time wasters is to read a long email or attend a meeting that just goes on and on and realize that it was unnecessary. Consider the following email snippet:

Email 1:

> Hi Programmer,
>
> The drop XX was deployed last night and the feature A will be tested today. To test feature A, there are three steps to be followed:
>
> - Upload huge files
> - Check in database
> - Download from UI
>
> We have tested with files of sizes up to 80MB. Do you think we should test with file sizes more than 80MB? If you have such files, can you please share it with us?
>
> Regards,
>
> Tester

Email 2:

Subject: Request for files >80MB

Hi Programmer,

To test Feature A, we need files of size >80MB. **Please share such files if you have**.

Regards,

Tester

Do you notice that the additional sentences don't add any value? The tester is interested in files and wants to know if the programmer has them. Instead of keeping it short and to the point, the first email highlights the requirement only at the end. It is the same case with meetings. People go on and on without conveying anything valuable. Don't be such a tester.

Tip 21: Have courage to speak the truth.

When Jerry Weinberg was asked about the most important traits for a software tester http://qablog.practitest.com/2012/11/five-testing-question-with-jerry-weinberg/ his answer was:

"Courage, communication skill, self-esteem." Testers test under extreme time pressure. The testing team seems to have many test ideas to test compared to the time on hand. Some of the testers are pressurized by their managers or other testers to fake testing results. But do you have the courage to tell the truth? Are you someone who has high standards in terms of personal ethics? Can you resist the temptation of keeping quiet and courageously speak out the truth?

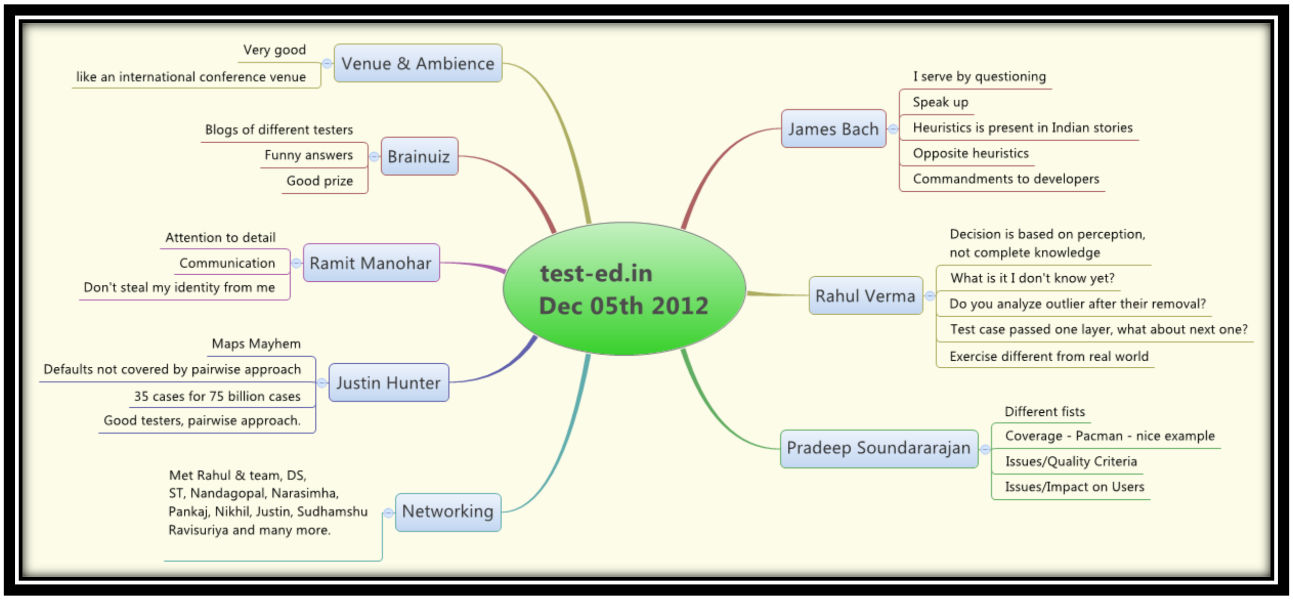
Think about it.

Tip 22: Safety language

At Tested conference at Bangalore this month, I attended Rahul Verma's talk on 'Death of Passed Test Case'. The entire conference was a blast. There is always a risk of someone misinterpreting a tester's word. Suppose a tester says that the product is tested and there are no bugs. Even a single bug proves that the tester's word is wrong. The tester might actually intend to mean that there are no bugs the tester is intentionally hiding from other stakeholders. As testers, we might often use the phrase - "I have tested the product completely." Don't we realize that any product cannot be tested completely? It is better to use safety language than commit and repent later.

Think of these phrases - 'It seems', 'As per the following tests, it looks like', 'It appears that'.

I have pasted the mind map of test-ed.in conference. Rahul also highlighted that a decision made by a tester is based on perception and not complete knowledge.



Tip 23: Keep them in the loop

In software projects, requirements and feedback is provided by multiple stakeholders. It may so happen that the product management sends you an email about a particular item not matching his/her expectations. It is your responsibility to share this news with the programmers as early as possible. Also if you communicate about the product/project with any of the stakeholders, remember to include the programmers in the loop. The relationship between the tester and the programmer will be impacted to a great extent if they get to know of this communication from someone else. Instead, while initiating any communication with others, do include the programmers if you feel that they need to be aware of the details.

Tip 24: Let them know about your test strategy

Every tester and the test team have their own style of preparing a test strategy. Some have it very detailed and some don't have a test strategy at all. If your team has a test strategy on how you will test and the list of test ideas, share it with the programmer. Let them know your plan of attack. By not letting them know your test strategy, you are letting go of a very valuable source of information. As they have worked on the code the most, they might have insights totally unknown to you. Take advantage of that fact. Early feedback from programmers about your test strategy is also useful to correct your test strategy, change your plan and find bugs early.

Tip 25: Use the MIP-ing strategy

MIP stands for Mention-In-Passing. If your programming team is tracked based on the number of issues raised against their code, try to educate their manager that it is incorrect. When you log issues, you might be considered as someone who is making the programmers look bad. Programmers might start to send back the issues or merge two issues or mark one as duplicate of a related issue. To avoid such cases, do not log the issues but mention them verbally or in a written report to the team. James Bach writes about Mention-In-Passing here: http://www.satisfice.com/blog/archives/97

Appreciate good work

Every professional wants to be appreciated, especially when they have completed a challenging task. It is interesting to know that more than the incentives, genuine and timely appreciation goes a long way in inspiring the professional to take up future challenges. If this is the case, programmers also might be in need of such appreciation.

Tip 26: Thank you email

Every product release, most of us work hard to deliver a good quality product to the market. Programmers talk to the product engineers, multiple stakeholders, work under strict deadlines and strive hard to deliver good quality code. And of course, their job does not end there. They fix the bugs found by the testers, answer their questions and take up last minute requirement changes too.

If you feel that your programming team works hard and contributes indirectly to your success as well, do not hesitate to drop an email thanking them. Be specific in highlighting about the good work done. Provide examples. Send the email quickly and just after the product is released to the market. Do not wait till customer provides his/her feedback. Do remember to include their manager in the email.

Tip 27: Comment on bugs

This is easy. Whenever you find bugs which have been fixed after a lot of hard work or you notice that it was a complex scenario and yet there are no side-effects found, appreciate the programmer. Comment on the bugs itself - “Nice fix. No more issues found.” People reading the comments will know about the good work of programmers. This will definitely improve the relationship between testers and programmers.

Tip 28: Challenging problem faced/solved

Most of us like to talk about our work and programmers are no exception. Talk to them. Ask them about their work. Talk to them about the most challenging problem faced and how they solved it. Who helped them? Did they discover any new resource which can be used by your team as well? How can similar challenges be avoided in future? Which other project or feature can they think of where similar problems can be anticipated? This would be a very informative talk as you would get a chance to learn more than the product. The information gained can be applied across projects.

Tip 29: How did they implement

Apart from talking about the technical challenges faced during the project, talk to the programmers about the product. Ask them how they implemented a particular feature of the product. Check if you know of any better alternative to implement the same feature. Have you tested similar feature before? Did multiple programmers work on any particular code? Was there any feature where they had to learn a new technology or was there any feature which had multiple interactions with other features? Which feature did they spend the most time on? Which part of code they used from similar feature of a different product? Did they take care of the bugs in that code?

Tip 30: Which area are they afraid of

In any product, few features get developed fairly quickly while some get delayed for a variety of reasons. The reasons could be any of the following:

- Delayed requirement documents
- Late feedback
- Unknown technology
- Inexperienced programmers
- Delayed delivery to testing team

How confident are the programmers about such features? Has it undergone enough unit testing? Some programmers have a gut feel about a particular feature breaking under certain conditions. Pay attention to such statements. It is better to prove it as a false positive than a missed opportunity.

Bring variety to the table

Testers know that things can be different. If you cannot offer anything new to the project, what is the purpose of having you in the team? While programmers focus on building a product, can you think in terms of the user, bring in your testing skills to the forefront and help deliver a great quality product?

A good explanation of tester's roles is here: http://www.satisfice.com/presentations/etta.pdf#page=4

Tip 31: Different quality criteria

Very few programmers think of multiple quality criteria. It would be safe to say that very few testers think in terms of overall quality of the product. I like how James lists some of the quality criteria here:

http://www.satisfice.com/tools/htsm.pdf#page=5

Not all bugs are discovered by similar tests. The more diverse the tests, the more diverse the bugs you might find. Try different tests and check if you have at least thought of the various quality criteria listed. If you become the tester who can find bugs that others miss and you find important bugs, programmers will like you. They will want to have you as the tester for every product they develop. Your relationship with them will improve and your reputation will increase.

Tip 32: User experience and user groups

Due to the time pressure, programmers tend to focus on functionality more compared to usability and user experience. You as a tester might be required to provide information about the user experience. Which part of the product is difficult to use? Do you feel frustrated or confused while you use the product? Also, can you think of different user groups and test on their behalf? Someone who is computer-savvy v/s someone who never uses the mouse. Will your product be a hindrance to any of the user groups? Have you tested? It might be a good idea to test and help the programmers.

Tip 33: Consistency across features and products

People do not like change. They love things to be constant and at the same time consistent. Same applies to software. How irritating it would be if Esc closes a popup on one page and does not do anything on the next page. A slide from Rapid Software Testing course highlighting about consistency is here: http://www.satisfice.com/rst.pdf#page=53

There are at least two points to note with respect to consistency.

Is the product consistent when compared to similar products on the market?

Is the product consistent with itself?

Think on these lines to avoid customer calls and complaints.

Bug Reports

After reading about various tips on bugs, communication, quality criteria it is time to concentrate on bug reports. Every tester needs to log a bug one day or the other. Bug reports play an important role in taking your opinion across multiple stakeholders.

Tip 34: Crisp, important things first

The first thing that usually strikes about any bug is the summary. Let the summary be crisp and highlight important stuff about the bug. There is a separate section for details - the description. For the summary, mention the problem first.

> **Good Example:** Call Log: Missing entries after five calls in twenty minutes.
>
> **Bad Example:** When a user calls many times, he does not see few entries in the call log.

If you think that your bug reports need improvement, consider taking the BBST Bug Advocacy course.

http://testingeducation.org/BBST/bugadvocacy/

Tip 35: Detailed steps

Many testers write bug reports as if the programmer sat next to the tester when the bug was discovered. As programmers, it is very difficult to work on an issue that is badly reported. It is very common that programmers do not have access to the test environment. It is a good practice to provide detailed steps to replicate the bug so that anyone who reads your bug report can replicate the bug. Do not assume that the same programmer who coded the feature will fix the bugs. I came up with list for bug reporting.

- Credentials
- URL
- Pre-requisite
- Steps/Description
- Attachments
- Log
- Related ticket
- Browser
- Language

Tip 36: Attachments

An attachment - an image or a video says a lot more than detailed steps. Feel free to attach as many proofs of the bug. When attaching a screenshot, highlight the area where you want the programmer to focus on. Avoid capturing areas which are unnecessary. Remember that one can hide the taskbar or switch to full screen. Also, mention the name of the attachments you are uploading. This helps the programmer to know if you attached the right screenshot/video or if the bug report does not have any attachment. It is also a good practice to mention the duration of the video if it is attached along with the bug report. Avoid huge attachments. Place the file in the common file server and provide the path in the bug report.

Tip 37: Mention tests conducted

When a tester logs the bug, the programmer has no idea of the tests conducted. Few questions the programmer might want answers could be:

- Is there a workaround?
- Is it limited to a specific browser, file type, file size, speed, version, order of steps?
- Is there a more general way of replicating the bug?
- Is there a more severe effect of this bug?
- Was this happening in earlier builds too?

By mentioning the tests conducted, we testers are providing a lot of information to the programmers which will help them solve the issue quicker. It also helps them so that they understand the root cause and not just fix the symptom.

Tip 38: Why should it be fixed

The programmers are not the only ones who read your bug reports. Anyone interested in the project will want to go through the bug reports. Some of them might be influential enough to decide if a bug must be fixed in that release or postponed. They might not be aware of the importance of the bug or the number of customers affected by reading the bug report. Instead, when testers also mention the reason as to why the bug must be fixed, they get the whole picture.

Attitude

This is one of the most important traits measured during most of the interviews for testers. The tester might be very good in technical skills but if you somehow feel that his/her attitude is not proper, the tester is not offered the job. As Robin Sharma says, people like to do business with those whom they like. Programmers also like to work with testers who have a good attitude.

Tip 39: Serve them by questioning

This was one of the lessons I learnt from James' talk in Bangalore - The rise of thinking Indian tester. As testers, we need to serve the project by questioning. Questioning the very assumptions to the documents to the processes to the value you provide by testing. A tester brings in his/her expertise to the forefront by questioning and providing the information to the stakeholders. Do remember that you are not the gatekeeper of quality. You - the tester - provide a service. A tester should strive to provide a service with a different mindset, very different from the rest of the team on the project.

Tip 40: Respect them and their skills

Every team in the project is needed to deliver a great quality product. Programmers need help of testers and so do testers. As testers, we need to respect the programmers and their skill-set. We should be very clear that our job is to provide information about the product and not insult the team. A bug is a problem with the code and not the person. Some testers think of teaching a lesson to the programmers by logging bugs. It is the worst attitude and does not help to improve the relationship between the two teams. If you want to be respected for your skills, learn to respect others - especially the programming team.

If you feel that you can do a better job than them, go ahead and code. Only when you get to code that you realize that it is a different ball game and quite difficult for the inexperienced. Respect them and thank them for their awesome work.

Tip 41: Tune to their timings & breaks

This is a strange but a very effective tip. Every professional has their own style of work and break timings. Some work effectively before noon while some are extremely focused in the evening. No matter what your style is, get to know the schedule of the programmer. Tune to their timings and breaks. It is better if you schedule meetings when they are not busy or else you will be disturbing them. Disturbing someone will definitely not earn you their respect.

If they come to office early and leave early, do not schedule meetings late in the day. They might be already switched off and ready to leave office. Instead, come early and talk to them when they are actively working and thinking about the project. Different programmers have different schedule. Do not assume that the current programmer will work similar to the one you interacted last.


ATTITUDE IS IMPORTANT
1. SERVE THEM BY QUESTIONING
2. RESPECT THEM AND THEIR SKILLS
3. TUNE TO THEIR TIMINGS & BREAKS


YOUR BUG REPORTS ARE REALLY HELPFUL
1. CRISP, IMPORTANT THINGS FIRST
2. DETAILED STEPS
3. ATTACHMENTS
4. MENTION TESTS CONDUCTED
5. WHY SHOULD IT BE FIXED

Collaborative Learning

Have you realized that a talk with a programmer is highly informative for both? Tester learns about the product internals and the programmer gets to know the different styles of testing. Once both of them understand the difficulties of the other team, it's easy to work together towards a common goal.

Tip 42: Highlight heuristics, OFAT, MFAT

Tell them how every test is a form of a question. Demonstrate how you are applying different heuristics based on different contexts. List out the most commonly used heuristics. Explain how you start with sympathetic testing, followed by aggressive bug hunting and then change your approach based on the context. Explain them the differences between One Factor at a Time and Multi Factors at a Time. Talk to them about critical thinking, lateral thinking and safety language.

Tip 43: Show them the tools you use

Every tester must have their toolkit handy. Do you have your toolkit ready? Show it to the programmers. Ask them about their toolkit. Some of the handy tools I use have been recommended by programmers. Broad categories of tools are as follows:

- Screen capture and recording
- File parser
- To-do lists
- Countdown timer
- Browser add-ons
- Mind mapping tools
- File comparison tools
- File sharing, test data generators

Ask them if they know of any better alternative. It would be great if they can teach you to write one by yourself.

Tip 44: Highlight testing challenges

We have been talking about programmers facing lots of challenges during the project. Programmers are not alone. Even testers have their own share of challenges. It starts from outdated requirement documents, inappropriate test data, time spent in setting up the test environment, replicating bugs, investigating intermittent issues and finally administration related work. Highlight how programmers can help you test better and quicker. Tell them how your time is also spent on resolving these challenges.

Tip 45: What does the code do

Ask them what does the code do? Ask them to take you through the entire code of a feature or module. Let them explain the checks and conditions and you can question the logic. Also, note down test ideas to test the feature. Ask them which conditions have they ignored on purpose and which conditions are they planning to incorporate in future. Also ask them about the probable bugs or kinds of bugs they expect in the feature. What bugs did they discover during their unit testing? This is valuable information which can be used during your testing.

Tip 46: What are the technical limitations

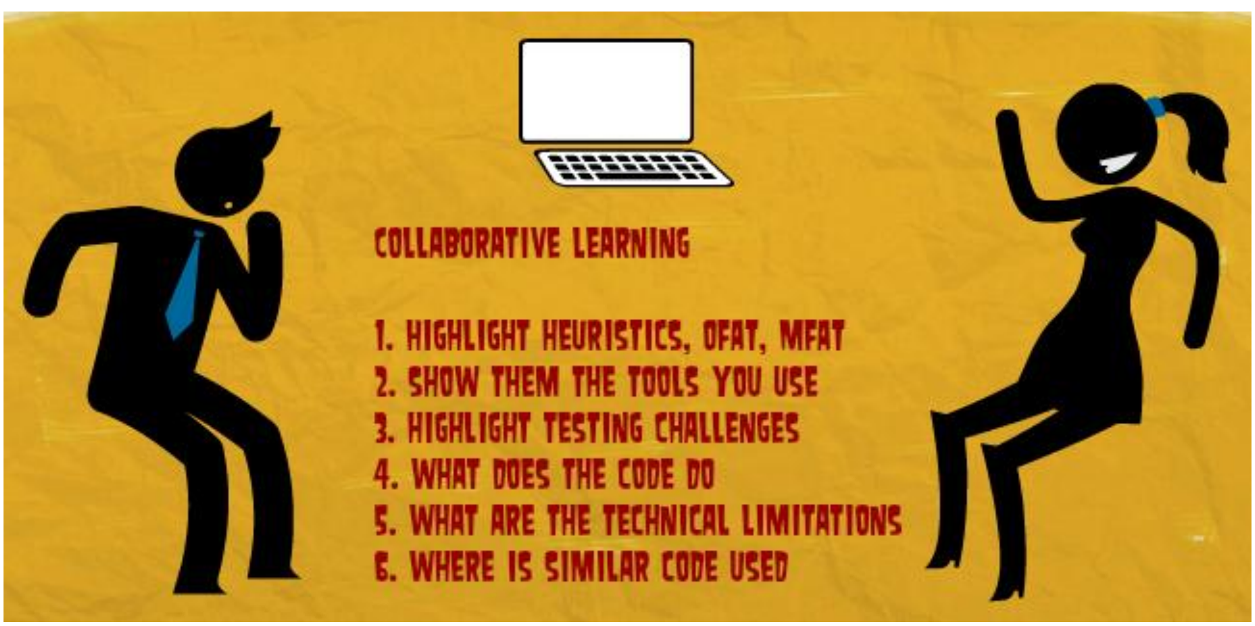
Knowing about the technology used is a good hint to use during testing. What kind of bugs is specific to a particular technology? Programmers might be able to help you in this regard. In addition to the bug you discovered, you might learn about patterns of bugs and find other bugs quicker. You will also know about the bugs which cannot be fixed and stop finding similar bugs.

Tip 47: Where is similar code used

It's good to know where similar code is used for two reasons:

- Bugs found in one area can be expected in other area where similar code is used
- It is better to test two features with different code first than test the same code being used across features.

Highlight to the programmers that this tip helps you test quicker and in turn find bugs across features. Especially in products with multiple features, this strategy is useful to plan tests and their order of execution.



Tip 48: Have a programmer as a mentor

Someone who codes every day might understand the problems faced by programmers better than a tester. As a tester, talk to him/her every day. The discussions will give you a different perspective towards programming. Talk to the mentor about the problems you face with other programmers. The mentor might help you consider other side of the issue. Even if you do not get a solution to the problems you face, you might get to know the practices followed by programmers.

Tip 49: Never say the following to any programmer

- Your code sucks
- Why do you introduce this bug
- Can't you even unit-test your code
- I am busy. Replicate the issue for yourself
- Fix the issue. It is important
- Who hired you

Tip 50: Never get angry at their remarks

- Why did you find this bug so late
- Do not test anything. It is a small fix
- Can you try once more
- Maybe it's an environment issue
- Can you test in next build
- You should learn automation
- It's not in my code

If you have any comments on any of the chapters or the book in general, feel free to contact me. I will be more than happy to discuss any of the topics.

My contact details:

Email: ajay184f@gmail.com

Skype/Twitter/GTalk: ajay184f

LinkedIn Profile: http://in.linkedin.com/in/ajaybalamurugadas

```
☺☺☺☺☺ ☺   ☺  ☺☺  ☺   ☺ ☺  ☺ ☺☺☺☺
  ☺   ☺   ☺  ☺☺  ☺☺  ☺ ☺ ☺  ☺
  ☺   ☺☺☺☺☺ ☺☺☺☺ ☺ ☺ ☺ ☺☺   ☺☺☺☺
  ☺   ☺   ☺ ☺  ☺ ☺  ☺☺ ☺ ☺     ☺
  ☺   ☺   ☺ ☺  ☺ ☺   ☺ ☺  ☺ ☺☺☺☺
```


Ajay Balamurugadas