Agile Record

The Magazine for Agile Developers and Agile Testers

January 2010
issue 1

www.agilerecord.com free digital version made in Germany

Software Testing Craft

by Markus Gärtner

Recently I have heard more and more developers, managers and consultants asking for more testers who “know what they’re doing”. They were referring to more testers knowing their craft: the craft of software testing. Though there are many publications about the craft of software testing (most notably, Brian Marick’s book by that name [1]), there are different opinions about how best to define this term. I decided to write down what I understand about this craft, which I practice on a daily basis.

The craft of software testing has matured over the past decades. Recent software development processes such as eXtreme Programming, Scrum and other Agile methodologies have challenged software testing practitioners to refocus. Software testing thought leaders have claimed that when developers take responsibility to deliver high-quality code, and collaborate continually with testers, they’ll get good results. Involving testers right from the start of the project, rather than treating testing as a separate ”phase” that happens after coding, mandates a mind-shift for testers in Agile projects.

Take responsibility

As a software tester, you should take responsibility for the outcome of every decision you make. I call this the zeroth rule of professionalism. As a tester, you have many occasions to take responsibility. Here’s an example: when finding a serious bug you will have to defend your point of view. The development team might put your approach to software testing into question. Prepare to defend your considerations in the face of these challenges. Be prepared to explain why you think the particular observed behavior of the system under test is a problem to the user - or maybe to your company’s reputation with its customers.

Next, when you're in the position to decide upon the test approach or strategy to take, prepare to be faced with questions about your decision. Each time you report a software problem, you may have to defend your viewpoint. Make sure to regularly take a step back, analyze the particular situation at hand and decide whether you need to adapt your process. If you do, take small steps. If you find a gap in your test strategy, brainstorm with the whole team about actions to take about it. Maybe the unit testing coverage needs to increase, or maybe the team needs to bring in a specialist on performance testing. Take responsibility to manage and steer these improvements based on feedback from your colleagues and your customers. Over time you will learn how to tackle problems in your context before they start to drag the team down.

More often than you will like you'll be confronted with difficult questions about your decisions. Be prepared to explain your reasoning. By making your considerations transparent and involving your colleagues in the decisions about your test process, you are more likely to get their buy-in. Over time this will foster trust among your team members. Your colleagues will start to trust your decisions, and over time they may start to trust you.

Be pro-active

Every tester needs a pro-active mindset. Sitting in your cubicle and waiting for the decisions and software packages to flow in is counter-productive in multiple ways. First of all, your colleagues will recognize you as someone waiting for something getting thrown over your wall. If you then notice some bugs in the software and simply throw bugs back over that wall, you will find out over time that your developers and project managers will react strangely to your efforts. For example, they might not ask you to join them at the bar in the evening. Maybe next they refuse to invite you to that important meeting. Secondly, you will be seen as lazy. Your colleagues will see that they work from nine to five while you're just sitting in front of your computer reading the latest online comics or surfing the web. Though this perception might not reflect reality, it will certainly de-stabilize morale on your team.

Don’t hide in front of your monitor - seek out ways you can help. You may find out that you can become a first-class team member

by leveraging your abilities. When you've earned some credibility, you might be invited to some important meetings. Go to the meetings to which you were not invited, if you think you can make meaningful contributions. Maybe people will recognize the value you bring to the project and invite you the next time around. Go to your developer to help with that bug fix. You may want to ask questions about the changes he/she makes, but resist the temptation to ask about every little change. Seek opportunities to show your meaningful contribution to the team and the project, and maybe - over time - you will be included in the decisions right from the start. Meanwhile, try to maintain your own motivation.

Never be blocked

By being pro-active, we can avoid being blocked. Instead of waiting for that bug-fix to get passed over the wall to us, testers can go out to their developers and help them fix the bug. Maybe you recently learned some design patterns and want to interview your developer how these are used in the production code. Showing interest builds trust and adds to the team's stability.

Another variation of "never be blocked" is staying directly in touch with your customer. Interview your customer on how they do a particular task in their business. Ask your customer honestly to show you how they work in order to learn about their daily work. Pair with your customer to gain additional insights. You will increase your understanding of the requirements and whether your software is going to be useful, which will be a huge contribution to your team's success.

Practice Testing

Seek out opportunities to learn new testing practices. Today's Internet is full of applications with which you may test and train your skills. Some testing leaders offer testing challenges on their blogs that will help you hone your skills. This is how I became a black-belt of the Miagi-Do school of testing, which Matt Heusser founded about a year ago. Initially, I took his testing challenge and mastered it. Since then Matt and I have kept in contact. Regularly, we exchange thoughts. Matt made me start to write about testing. This helped me get in touch with other professional software testers more regularly. Today, if I have a difficult problem at work, I can ask one of my contacts over the Internet for their opinion on it.

You may also join any of the testing-related mailing lists and ask for a challenge, a small program to train your testing approach. This will introduce you to new ways of dealing with software and new ways on how to do testing. Download an open-source tool from the Internet, install it and try it out. If you find issues, write a test report to the contributing programmers to let them know about your perception of their program. You might file a bug on their website. You may earn respect for your valuable feedback and contributions by doing so.

You can gain valuable experience by testing Internet software and participating in testing challenges in your spare time. Expect to screw up - more often than you would like to. By taking a step back and reflecting on your failure, you will realize how to improve the next time. Since your errors will not have dramatic consequences for your company, you will be able to learn and improve your testing skills in a safe environment. And besides that, it might turn out to be pure fun for you.

Continuous Learning

Learning is a key practice for self-improvement. You can achieve continuous learning in several different ways. Start by reading some books on topics related to your work. These do not need to be technical at all. For example, psychology books give insights into your team structure and how people at your work interact with each other. For example "Please Understand Me" from Kiersey and Bates[2] and "Fearless Change" from Rising and Manns[3] are two books on the topic that are worth reading. Over time you may want to peek into some blogs on the Internet, read articles in online magazines and participate in newsgroups and online communities. Over time you may be able to talk at conferences about your work and spread and share your knowledge with others while getting to know their particular context. Remember: When you stop learning, you are dead.

Apprenticeship

When you have sufficiently mastered your craft, you may be ready to take on apprentices. By teaching new students how to become a software tester in your particular context, you may even learn new things from your colleagues. When teaching how you think testing can be done, you may realize through self-reflection how to improve your very own testing skills. In order to know what to teach, you need to reflect on your course of action, you need to observe yourself while testing, you need to interview yourself. Not only does this enhance your teaching abilities for your apprentices, but thinking about what and how to teach you might find out how to improve yourself. Self-blindness is the first big obstacle to innovation, according to Jerry Weinberg in "Becoming a Technical Leader"[4]. By reflecting on yourself, you can avoid this obstacle.

What should I do tomorrow?

Being a software testing craftsperson is mostly an attitude of professionalism. Taking responsibility for your decisions and actions in software testing should become general common sense. Defending your position and your approaches takes this concept even further. Stick around in testing long enough, and you are bound to be asked some tough questions. Since every software problem is a quality problem, you will need to defend your strategy and your approach to testing more often than you like. By being pro-active and avoiding to become blocked, you may be able to lengthen the time between "impossible to answer" questions you get from your colleagues. Keeping your knowledge up-to-date also helps you in that quest. By taking testing challenges from a blog or testing professional on the Internet, you can train your testing skills continuously and seek for different approaches in other contexts.

Training new colleagues and help them master the craft on their

own helps you avoid the fallacy of self-blindness, and might also give you some new ideas and approaches over time. Altogether, this should help you get to know how to master the craft of software testing. ■

Bibliography

[1] The Craft of Software Testing, Brian Marick, Prentice Hall, 1995

[2] Please Understand Me, David Keirsey, Marilyn Bates, Prometheus Books, 1984

[3] Fearless Change, Linda Rising, Mary Lynn Manns, Addison-Wesley Longman, 2004

[4] Becoming a Technical Leader, Gerald M. Weinberg, Dorset House, 1986

> About the author

Markus Gärtner

Markus Gärtner is a software tester for Orga Systems in Paderborn, Germany. Personally committed to Agile methods, he believes in continuous improvement in software testing through skills. Markus also blogs at blog.shino.de and is a black-belt in the Miagi-Do school of software testing.