# Agile Record

The Magazine for Agile Developers and Agile Testers

## Agile Adoption Stories – What Worked and What Didn't

May 2013
issue 14

www.agilerecord.com free digital version made in Germany ISSN 2191-1320

# Column

# Keeping up?

*by Huib Schoots*

"Testing cannot keep up with development. They need to test quicker!" This was one of the statements that were discussed during a DEWT peer conference [1] in the Netherlands that took place from the 20th to the 21st of April 2013.

Is this also familiar to you? I have often heard people say testers are not able to keep up with the pace of development. Is that really true? When conducting projects in an agile context the whole team is supposed to be responsible for getting things done together. Programmers are equally responsible for testing as any other member of the team. So how could this be true? I have met several programmers claiming that the tester in their project was not able to thoroughly test everything that they were building: "testers are too slow and testing is taking too much time." When talking about their problems with testing it became clear to me that those programmers were not contributing to the testing in their projects very much.

## DEVOPS

While visiting a client, I experienced that DEVOPS was taken quite literally. The organization only valued programmers, which they referred to as developers and people who could do maintenance, which they called OPS. Other disciplines were not appreciated very much. Here I met programmers who were not willing to do testing at all! Code in this case often went untested. One of the most popular excuses used was the time pressure to meet deadlines. I was not surprised that this organization made international headlines not so long ago, referring to serious problems with their IT.

James Bach turned the "keeping up with programmers" statement around during the DEWT weekend: in teams where testing is alleged to not be able to keep up, programmers are quickly introducing risks that they cannot keep up with. I like his view to this problem. Teams should notice that they are creating risks faster than they can understand and cope with. He also gave food for thought: can we automate development? Or management for that matter? No! So why do people think we can automate testing?

## Reducing Risks

So how can agile teams reduce the risks, while creating new products or changing the existing products? When I look at teams I see two types of people: those who are optimistic and dream that everything will be all right and those who are skeptical and often unfairly judged as pessimistic. Excellent testers know that things can be different and are trained to identify risks, learn fast, and think critically about what they see in order to find as much potential problems as possible. I seriously cannot imagine any agile team can function well without a trained tester. Does this need to be a dedicated tester? It could well be, but there are many different ways to implement the testing role. As long as the one(s) who picks it up, is well trained or at least coached by a highly skilled expert.

## Learning about the product

Why do people think testing can be fully automated? Developing software is often seen as factory work [2] . But to me developing software or even configuring an off-the-shelf product is like research and development. Humans decide what they want to build, but they learn over time and they adapt current desired methodologies. Together with the team, the product owner explores what is wanted and the team helps him to achieve their wishes.

Here are some things I have learned through my experience that you may want to considering order to lower the risks in your project, while addressing the ridiculous claim that "testing cannot keep up with development".

## Excellent unit testing

Excellent unit testing is essential. In any project unit testing is important but in an agile context it may even be more important! Developers need to test their own stuff and I know they can do that. Teams can learn how to do it properly and efficiently. They can do an awesome job automating their own checks and build a solid pack of automated unit regression checks [3], making sure that the basic stuff works when releasing code to be tested by others. This also reminds us to automate checks in a smart way. Do not try to automate tests on a GUI level, which can be done on lower level; for example integration or even unit testing level.

## Regression testing

In an agile context software is developed iterative and incremental in sprints so regression testing is very important in many cases. Regression testing is often checking if untouched functionality still works. In many agile contexts you will want to have an automated

suite taking care of regression risks. Test automation (some prefer to call it automated checking [3] or tool assisted testing) is essential for fast feedback and continuous integration. However, this does not make testing unnecessary! Thinking of the right tests to do and learning about situations that can occur is why testing is always needed when creating or changing software. Even if you only change a process or a single artifact, it is worthwhile to consider getting a trained tester to look at your work.

## Lightweight visual documentation

In an agile context we want to move fast. This does not mean creating no documentation, as this introduces even more risks in your project. So what can we do since we do not have the time to create test plans, test cases and test data as we are used to? Test documentation needs to able to deal with change by being transparent, easily accessible and maintainable. Using simple dashboards and lightweight test documentation (like mind maps) maybe a solution you want to consider. I wrote about visualization on my blog here [4] .In my opinion it is very important to use visualization. This does not only help you communicate but also aids your thinking. Drawing models, diagrams or other visualizations can help you conduct your thinking in a more productive way, as well as making learning generally easier.

## Work together

Work together? Duh! Everyone involved in software projects does that, do they not? Yes, in a way they all do. But let us take a closer look at how you do that in your project. Are programmers and testers really working together? Are they creating test strategies together? Are testers assisting developers to do excellent unit testing or review code? Are developers allowing the testers to test as fast as they can by making testing easy? Think of what developers can do to make testing life easier: create extra logging, build little nifty tools, automate checking, scripting stuff that is error prone or create test data. I bet you can think of much more ideas to help your team work together efficiently.

## Testability

Testability is tremendously important: the better testability is, the faster your testing will be. It is hard to believe that testability is often not a topic in software projects. To really improve your testing this is the "trump card" only a few "play". Improve the availability and stability of the software being tested to speed up the testing process. This may seem pretty obvious, although this point can often be overlooked. What are the other less obvious aspects of controllability that can make automation easier, helping the team observe and analyze what is going on? Have look at this list of testability heuristics [5] by James Bach to learn more.

## Pairing

Pairing is widely misunderstood by managers. Managers often argue that it is twice as much work to do the same job. Is that really the case? I think not. Pairing can be a very useful way to work! Apart from the social aspect that you learn faster, you make less mistakes and your creativity is stimulated by other people's ideas. Another huge advantage of pairing is people learn to understand and appreciate each other's work. Some examples could be: testers learning to read and understand code or developers learning about test techniques. You really build and empower your team by having them work together closely.

There is much more to talk about and I hope this column makes you think about testing while adopting agile processes in your context. I wrote a blog post [6] claiming agile testing is not very different from traditional testing. I believe adopting agile practices gives you the opportunity to improve efficiency through cooperation, exploration, learning and evaluating. I leave you with the 7th principle of context-driven testing [7]: "Only through judgment and skill, exercised cooperatively throughout the entire project, are we able to do the right things at the right times to effectively test our products."

## References


1. DEWT3 see: *http://dewt.wordpress.com/2013/04/24/dewt3-experience-reports/*
2. See: *http://en.wikipedia.org/wiki/Software_factory*
3. Testing and Checking Refined by James Bach & Michael Bolton at *http://www.satisfice.com/blog/archives/856*
4. Visualization at *http://www.huibschoots.nl/wordpress/?p=927 OWASP Top 10 Project at https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project*.
5. Heuristics of Software Testability by James Bach at *http://www.satisfice.com/tools/testable.pdf*
6. What makes agile testing different? at *http://www.huibschoots.nl/wordpress/?p=1072*
7. Context-Driven Testing at *http://context-driven-testing.com/*


■

## > about the author


***Huib Schoots***

*Huib Schoots is currently an agile test consultant at codecentric where he shares his passion for testing and agile through coaching, training and giving presentations on a variety of subjects. With fifteen years of experience in IT and software testing, Huib is experienced in different testing roles. Curious and passionate, he is an agile and context-driven tester who attempts to read everything ever published on software testing. A member of the Dutch Exploratory Workshop on Testing, black-belt in the Miagi-Do School of software testing and coauthor of a book about the future of software testing. Huib maintains a blog on* ***magnifiant.com****.*