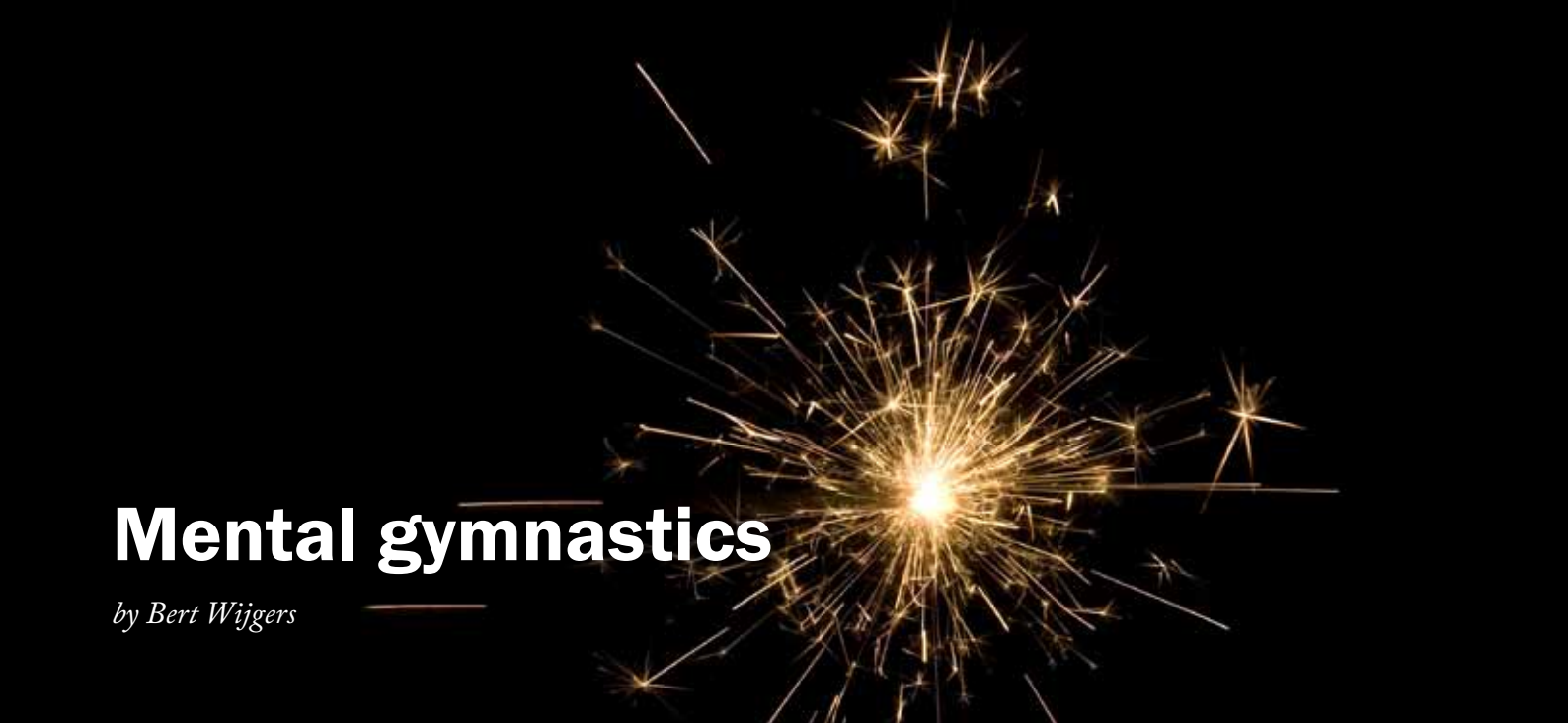
Agile Record

The Magazine for Agile Developers and Agile Testers

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

October 2011

issue 8



Mental gymnastics

by Bert Wijgers

What happens if we apply the famous four values from the Agile Manifesto to software testing, regardless of the development methodology that is being used? It turns out that agility in testing is something other than testing in agile projects. Testing in agile ways is actually very much the same as exploratory testing.

Before we get to the values of the Agile Manifesto (Beck and others, 2001), there are two distinctions to make in regard to the scope of this article and the perspective of its author. These are the distinction between testing and checking and the one between Agile and agile.

Big A Agile and small A agile

"Agile" with a capital A is a belief system about software development, whereas "agile" with a small A is a metaphorical adjective, when used in relation to software development. For the remainder of this article I will write the word "agile" with a small A. In my opinion there is no place for belief systems in software development. Perfect software does not exist, neither is there a perfect way to make it. We can only try and try to get better at it. In doing so, mistakes are inevitable because they are necessary for learning. Learning without making mistakes only happens in class rooms. I will treat the word "agile" like everybody does its counterpart "waterfall", as a metaphor and not as a holy concept.

Testing and checking

This article is about testing and not about checking (Bolton, 2009). Checking whether a system acts according to specifications can be automated and, if the same check has to be done often, it is worthwhile to do so. Unlike checking, testing cannot be automated. Whenever specifications are not complete or not completely valid, the human mind is needed. Since specifications are never complete and never completely valid, human testers will always be necessary.

In terms of the agile testing quadrants (Marick, 2003) I will talk about tests that critique the product, not those that support the team. Tests that critique the product are done to expose weaknesses and can only be done when the product is capable of doing what it is supposed to do in terms of features or functionality. Once that has been checked, the product is ready to be tested. Checking is looking for confirmation that all is well, testing is looking for trouble.

People and interactions over processes and tools

We test on behalf of others, so we have to imagine how those others would experience a piece of software. Therefore we must know what they need and what they want. For a tester it is imperative to understand what the requirements are and documentation can never be complete, nor is it ever completely valid. Neither is verbal communication, but in face to face interaction there is the bonus of non-verbal communication. Lots of important information about expectations, needs and desires is transmitted in non-verbal ways. This information we can use to imagine what a user's experience will be like.

We are used to talk about tools as executable pieces of software that support the development process of another, new piece of software. In the broader sense a tool is something that can be used to make something else. Documentation can also be considered as a tool in software development. One specific kind of documentation is the description of processes. Processes happen, whether they are documented or not. The extent to which they are documented is a matter of choice.

Then there are generic process descriptions also known as methods. These are also to be considered as tools; they should not be followed but they should be chosen carefully and they should be used wisely. As for best practices – they don't exist. There are only good practices in context (Kaner and Bach, 2009).

In the software development processes, the human mind is the most important tool. The best way to improve testing is by training individual testing skills.

Working software over comprehensive documentation

The most direct way in which testing contributes to working software is by the bugs that it reveals. When those bugs get fixed, the software gets better. Any activity that does not contribute to bug finding should be minimized. Test scripts should therefore not be too detailed, neither should test plans. Since a tester will not find bugs while he is writing scripts or plans, the time spent on making these documents should be as short as possible. The most important writing that testers do is in their bug reports.

In exploratory testing (Bach, 2003) the preparatory documentation is nothing more than a mission. The mission describes roughly where to look for what kind of trouble. With nothing but this mission, his skills, a toolkit and an uninterrupted stretch of time, a tester sets out on his quest for bugs. The exploratory testing cycle of test design, test execution and learning can be recorded and notes can be made. Usually it is a good practice to debrief an exploratory test session; the tester tells what he has done and why. In doing so he gives an idea about test coverage and he learns ever better to reflect on his own work.

Only in its most undocumented form an exploratory tester delivers no other documentation than his bug reports. As pointed out earlier, documentation is a tool. Before writing it, consider who is going to use it and what they already know.

Customer collaboration over contract negotiations

Since we test on behalf of others it is very welcome that those others are available. They will give us ideas and they can tell us whether or not our testing makes sense to them. Ideally we do paired testing, combining business knowledge and testing skills, to optimize the chances of finding relevant bugs.

As a tester you will not often play a role in contract negotiations, but you will be asked about the quality of the software. Is the product good enough? The truth is that you don't know. The only ones who do know are the actual users, when they use it. So when the question is asked, make clear that quality cannot be quantified. Acceptance criteria cannot be stated in terms of number of unsolved bugs of certain severity classes since there is no sure way of telling how these bugs will influence actual use of the software. And what about the bugs that you have missed? The only sure thing about them is that they exist, but nobody knows how many and how mean they are. What you can do, is point out what has been done to assure that the product does what it should do and what has been done to look for trouble.

Testing can only cover a limited subset of possible usage scenarios. The amount of possible combinations of circumstances and inputs, even for a simple system, is infinite. Test coverage, in this sense, can therefore never be more than zero percent. With the quantification of risks we also have to be cautious. The multiplication of the impact of the event, usually expressed in terms of money, and the probability of occurrence can be no more than an indication. Risks always exist, but the associated events will not always occur; either they do or they don't. Eventually all carefully estimated probabilities will turn out to be wrong.

Responding to change over following a plan

Some sort of planning is necessary to get around in the world if you happen to have, even the most moderate, ambitions. The question is: how do you plan? Do you plan far ahead or do you wait till the last moment? In how much detail do you plan and do you write your plans down?

In software testing written plans come in two varieties: test plans and test scripts. More often than not they are written before test execution and in the meantime changes take place. Requirements change, risks change, the software itself changes. Test plans can be quite easily adapted to those changes, and they should. Test scripts on the other hand, often contain a higher level of detail and are less adaptable. Therefore, if necessary, they should be written as late as possible.

Designing tests is planning test execution. In exploratory testing the result of a test influences the design of the next test. Unlike the execution of tests following a script, in which the tests are defined before test execution, exploratory testing is influenced by the knowledge that is acquired by executing tests. Exploratory testing is to software testing what agile development is to software development; plans take form as we go along and they are informed by the latest information available.

The quality of testing

Being agile is good, but agility is not always needed, nor is it ever enough. A gymnast also needs strength, balance and concentration, to name a few things. At the end of a testing day you know whether or not you did a good job. A jury might give you points, but only you can truly judge. Guilty or innocent, there is always something to learn.

Software development still involves a lot of testers, people who occupy themselves exclusively with the quality of other people's work. This will change, partly because confirmative testing or checking will be ever more automated and partly because everybody else involved will get better at testing. Testers will have to be very good at their job, or they better learn how to be part of the software creation process.

As a software tester you can never say: "I made this." Your only claim to fame is: "I prevented disaster from happening." But who is going to believe that? And when disaster does happen, you will be the one to blame. Somehow everybody always believes that software fails because it was not tested well and never ever because it was badly designed or badly coded. So when it comes to the quality of your work as a tester, you have to be able to account for what you did and did not do to expose weaknesses. You will stay clear from blame if you can explain and defend the choices you made. Unlike in checking, in testing these choices cannot be made upfront, since trouble seems to have a will of its own. Therefore all testing is exploratory.

References

Bach, J. (2003), Exploratory Testing Explained, http://www.satisfice.com/articles/et-article.pdf

Beck, K. and others (2001), Agile Manifesto, http://agilemanifesto.org/

Bolton, M. (2009), Testing vs. checking, http://www.developsense.com/blog/2009/08/testing-vs-checking/

Kaner, C. and Bach, J. (2009), The Seven Basic Principles of the Context-Driven School, http://www.context-driven-testing.com/

Marick, B. (2003), My Agile testing project, http://www.exampler.com/old-blog/2003/08/21/

> About the author

Bert Wijgers

is a test consultant with Squerist, a service company in the Netherlands. Squerist focuses on software quality assurance and process optimization. Its mission is to inspire confidence through innovation.

Bert holds a university degree in experimental psychology for which he did research in the areas of human-computer interaction and ergonomics of the workspace. He has worked as a teacher and trainer in different settings before he started an international company in web hosting and design for which a web generator was developed. There he got the first taste of testing and came to understand the importance of software quality assurance. Bert has a special interest for the social aspects of software development. He uses psychological and business perspectives to complement his testing expertise.

In recent years Bert has worked for Squerist in financial and public organizations as a software tester, coordinator and consultant. He has written several articles for the Testing Experience magazine and is a passionate speaker on topics related to software quality.