16

December 2011

testing experience

The Magazine for Professional Testers

The Future of Testing

ISSN 1866-5705 www.testingexperience.com free digital version print version 8,00 € printed in Germany

The cutting edge

by Bert Wijgers

Traditionally, testers operate on the cutting edge between business and technology. Since the axe of software development used to be very blunt there was plenty of room for everybody. As the axe gets sharper, most testers will have to choose sides: business or technology. The cutting edge will accommodate only a few; the rest of us will be sharpening the axe.

There are lots of ways to deliver bad software and by now we have tried quite a few of them. There are also ways to deliver good software. Let's assume that, slowly but surely, we are getting better at making good software. In the future the right software will be delivered timely and within budget. This implicates that Waterfall projects will become an extinct species.

In agile development, testing and coding are intimately interwoven. Through daily and intensive interaction with coders and designers, testers will learn more about the inner workings of the software. This will change their mindset; testers will abandon the idea of the Black Box. Testing will be guided by knowledge of the code and will inevitably move away from the user perspective.

The agile approach to software development has brought business representatives into the development teams. Testers still play a role in facilitating communication between the business people and the technology people, but no longer in the role of translator. Business representatives interact directly with the designers and coders. Testers can only assist to help clarify misunderstandings.

Another consequence of agile practices is that everybody learns to care about quality. Testers will guide designers and coders on how to check their own work. This will give them time to do other things, like (exploratory) testing.

Specs and checks

An important part of what is now called software testing is confirmatory in nature; test cases are executed to see whether or not the software reacts conform to specifications. Another part of software testing is exploratory; test cases are executed to see whether or not something goes wrong. This is the distinction between checking and testing (Bolton, 2009) and it limits the scope of what testing will be in the future. All testing will be exploratory by definition. Confirmatory test activities will be called checking and they will be an integrated part of the software development process.

Checking can be planned, scripted and automated to the extent that specifications are stable. Unfortunately, specifications are never complete nor ever completely valid, so they change. If they don't, the product will not be able to satisfy the stakeholders. There is enough evidence by now that the idea of "big specifications up front" is just not workable for most types of software. So, work on specifications will go on throughout the project.

The testers of today can be the requirements engineers of tomorrow since they know about the business and they know about the technology. On top of that, they are analytical and can express themselves objectively in human language in a clear, concise and complete way. And they know when a spec can be checked.

A check can even be a specification itself, and as such drive design and coding activities. This is now commonly known as Test Driven Development, but in the future we might call it Specification by Example (Adzic, 2011). Software can be checked fast and often by making specifications executable.

The specification and the check will become one thing and because it is executable it is easy to keep up to date. At the same time the check is a piece of documentation. In the future there will be no separate requirement documents, functional designs or technical designs. There will be one description of the software in the form of something that we used to call test cases. These will be automated from the day they are born as examples of specifications.

Looking for trouble

And then, of course, there are the non-functional specifications. Unlike functional specifications those cannot be stated in a straightforward, simple and checkable way. Therefore non-functional testing has a bright future. It is easy to check whether or not a particular screen is loaded within two seconds under particular circumstances, but it is not easy to check whether it does so under all circumstances. The same reasoning holds for security testing and different sorts of "-ility testing". Non-functional testing is about looking for trouble and not about confirming that all is well.

Only when it has been checked that the software does what it is supposed to do, in terms of features and functionality, it is ready to be tested. The software that is released for test will get better, but it will never be perfect. The defects will be fewer and harder to find, so the functional tester of the future has to be very good at bug hunting. There is no sure way of telling where bugs are hiding, so traditional testing methods are of limited use. The context-driven school of testing (Kaner and others, 2002) will become more prominent, because it is focused on individual skills.

Traditional functional testers who stick to their scripts and plans will play no significant role in the future; they will fall of the ever sharper cutting edge. If they want to cling on, there is plenty of room on both the business side and the technology side. Projects will benefit greatly from requirement engineers with a background in testing. Good specifications are the basis for confirmatory checking activities. Those checks can be scripted and automated.

Testing can be aided by tools, but it cannot be automated. A human user has to interact with the software to experience its quality. The actual end users can and should do this. Professional testers can take acceptance testing to a higher level, but they can never replace the actual users.

The code and its automated checks are developed by interdisciplinary teams. Before releasing the software, it has to be tested by an outsider. In the future, software testing will be what it has always been: a highly explorative quest for bugs as a last line of defense. The big difference will be in the quality of the software to be tested.

References

Adzic, G. (2011), Specification by Example, How succesful teams deliver the right software, Manning, New York.

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

Kaner, C., Bach, J. and Pettichord, B. (2002), Lessons Learned in Software Testing, A Context-Driven Approach, Wiley, New York.

› biography

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 his 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 about testing and is a passionate speaker on topics related to software quality.