The Reality of Software Testing in an Agile Environment.

Ten QA Myths Blown Apart

By George Wilson, Operations Director of Original Software

Introduction

The definition of agile testing can be described as follows: “Testing practice for projects using agile technologies, treating development as the customer of testing and emphasizing a test-first design philosophy. In agile development, testing is integrated throughout the lifecycle, testing the software throughout its development.”*

Agile is a methodology that is seeing increasingly widespread adoption and it is easy to understand why—especially if you consider the developer and user point of view.

Users: Don’t want to spend ages being quizzed in detail about the exact requirements and processes for the whole system, and then have to review a large specification, which they know could come back to haunt them.

Developers: Don’t want to have to follow a tight specification, without any expression of their own imagination and creative talents, especially if they can see a better way.

Yet for the QA professional an Agile approach can cause discomfort – In the ideal world they would have a ‘finished’ product to verify against a finished specification. To be asked to validate a moving target against a changing backdrop is counter intuitive. It means that the use of technology and automation are much more difficult, and it requires a new approach to testing, in the same way that it does for the users and the developers.

All the agile approaches have (at least) one characteristic in common in that they impact the role of the QA professional. This in itself is not a bad thing when the outcome is a step change for the better. However, when decisions are made on the basis of an invalid paradigm, change is not always analogous with progress. When a new paradigm is proposed for software development, by software developers, it is not a surprise that it is developer-centric. Abraham Maslow said that *“He that is good with a hammer tends to think everything is a nail.”* The responsibility of the QA profession is not to bury its head and pretend that agile development will go away, it is our responsibility to engage in discussion to ensure that someone with a hammer is not pounding on a screw!

With the emergence of Test Driven Development** some suggest the role of QA is now questionable citing Test Driven Development (TDD) as the key to testing. But, what is most important is that QA is directly involved in the agile scrums all the way through, to be an integral part of the team designing the tests, at the same time as the requirements and solutions evolve.

QA teams need to know the real impact of an Agile methodology, there are boundless myths circulating the industry. Here is our reply to ten of our favorite myths:

Ten QA Myths Regarding Agile Testing.

There are a number of comments and statements regarding TDD and the QA function beginning to appear in articles on the internet by so-called specialists, that are, at best, misguided. This article responds to some of these myths and highlights challenges that QA teams will need to face up to and address in order to be successful in an increasingly agile world.

Myth One: "You only need to unit test - TDD testing is sufficient"

For the vast majority of commercial developments this simply isn't true. Even strong proponents of agile development recognize the need for their armory to include a range of testing techniques. For example, Scott W. Ambler has a list of 21 different testing techniques as part of his FLOOT (Full Life Cycle Object-Oriented Testing) methodology, including white box, black box, regression testing, stress testing and UAT. (http://www.ambysoft.com/essays/floot.html)

TDD programmers rely on these tests to verify their code is correct. If the requirements (test cases) are specified incorrectly, then you'll build robust code that fails to meet the objective.

Therefore, most agile projects include investigative testing efforts (black-box) which support rather than compete with white-box testing. Good investigative testing will reveal problems that the developer missed before they get too far downstream.

Myth Two: "You can reuse unit tests to build a regression test suite"

Some TDD proponents argue that conventional downstream testing is unnecessary because every line of application code has a corresponding test case; they suggest that by reassembling unit tests, everything from User Acceptance Tests to Regression Tests can be performed.

Although this sounds feasible, it isn't realistic, and here's why:

The granularity and objectives of white-box unit tests developed in TDD serve a different purpose from downstream black-box testing.

While the overall objective of a unit test is designed to prove that the code will do what is expected, the aim of regression testing is to ensure that no unexpected effects result from the application code being changed. These two objectives are not synonymous – e.g. checking that an attribute has a valid date format, is not the same as checking that for a given input, the value of the field contains an expected date.

Myth Three: "We no longer need testers, or automation tools"

Professional testers fulfill a different and equally valid role from their development colleagues.

It is widely recognized that traditional test automation tools have failed to live up to the hype of the vendors. Original Software's roots are as providers of products that improve the productivity of the database testing environment. It was because there were no adequate tools to provide User interface testing that we extended our solutions into this market sector; our heritage is aligned to effective testing rather than screen-scraping automation. When we developed TestDrive, we did it with the benefit of twenty years hindsight of missed opportunities from the traditional vendors.

Often, TDD projects have at least as much test code as application code and, therefore, are themselves software applications. This test code needs to be maintained for the life of the target application.

Myth Four: **“Unit tests remove the need for manual testing”**

Manual testing is a repetitive task; it’s expensive, boring and error-prone. While TDD can reduce the amount of manual effort needed for functional testing; it does not, remove the need for further black-box testing (manual and automated).

By automatically capturing the tester’s process and documenting their keystrokes and mouse-clicks, the tester will have more time for the interesting, value-add activities, such as testing complex scenarios that would be difficult or impossible to justify automating. Though manual testing is a time-consuming (and therefore expensive) way to find errors, the costs of not finding them are often much higher.

Myth Five: **“User Acceptance Testing is no longer necessary”**

Within agile development, acceptance testing is often positioned as working with the customer to resolve “incorrect requirements”, rather than correcting functionality that does not map to what the user needs. When the user initially defines their requirements, they do it based on their initial expectations. When they see the system “in the flesh” they will invariably come up with different, or additional, requirements. While agile methods might reduce the frequency of this happening, it is unreasonable to expect the approach to resolve them entirely, so there should be no expectation that user acceptance testing will be obviated. This is especially true when it comes to the business user signing off approval of the User Interface, since they may have envisaged something different from the developer; which brings us nicely to myth six...

Myth Six: **“Automation is impossible”**

Automation in the early stages of an agile project is usually very tough, but as the system grows and evolves, some aspects settle and it becomes appropriate to deploy automation – automation that can cope with changes by using approaches such as self healing scripts.

To begin with, almost all testing from a user and QA perspective will be manual but this testing effort and design can be beneficial later if captured and re-used.

Knowing the right time to automate is tough, so using technology that proactively supports early manual testing but provides a path to evolve this into automation is key.

Myth Seven: **“Developers have adequate testing skills”**

If testing was easy everybody would do it and we’d deliver perfect code every time.

An independent testing team serves as an objective third-party, able to see “the big picture”, to validate the functionality and quality of the deliverable. While developers tend towards proving the system functions as required, a good tester will be detached enough to ask “what happens if…?” When you include business user testing as well, you are more likely to have a system that is fit for purpose.

Finally, and I’m sure this point will be disputed, most developers don’t actually want to spend much time first writing tests and then developing code to prove the tests work. Using the collaborative process described below, the developer gets ample assistance in describing the functional tests and can focus on writing lean, accurate and robust code.

Myth Eight: **"The unit tests form 100% of your design specification"**

Whichever development method you use, before you develop code you have to think about the requirements. While TDD "done well" may identify a fair percentage of the design specification, there are still concerns about gaps between the unit tests. There are other equally viable approaches. The argument, that you need TDD to prove the requirements are captured accurately, isn't substantiated by history.

Defining test cases to check the accuracy and succinctness of the requirements is nothing new. The V-model, for example, is a valid approach to understanding testing requirements – and by implication, functional requirements. Like TDD, the V-model and most other models, fall down when the practitioner's approach is rigid, while development processes are fluid. Software engineering is not like mechanical engineering and trying to force conformity is wasted effort. Whichever approach you chose, correct thinking challenges each user requirement by asking "how would I test that?" The important factor here is that test cases are challenged before they are committed to code, otherwise you'll be doing a lot of recoding and calling it "refactoring".

When the requirements are refined through collaboration, the developer receives a more robust specification that is less likely to change because it has been openly appraised from several people's perspectives.

Myth Nine: **"TDD is applicable on every project"**

As the size of the project increases, the time required to run the tests also increases. This can be addressed by partitioning either/both the project and/or the tests. Either route results in tests that run at different frequencies depending upon their relevance to the current coding effort. This approach introduces the need for test planning and execution management. To achieve this efficiently, in addition to white-box testing, you need to be thinking about:

Integration Testing – "Which tests do I need to run to ensure the new code works seamlessly with the surrounding code?"

System Testing – "Does the functionality supported by the new code dovetail with functionality elsewhere in this system, or in other systems within the process flow?"

Regression testing – "How often do I need to run a regression test to ensure there are no unforeseen impacts of the new code?" Automated regression testing provides a safety net that can affirm agile development techniques.

User Acceptance Testing – "While TDD (in collaboration with business users) should ensure that a specific function performs correctly, is the cumulative impact of changes still acceptable to the business users?"

However, in today's environment it is unacceptable to think of these testing stages as discrete serial activities. Often they will need to be run in parallel as we get a new 'code drop'. As the size of the project team increases (along with testing effort), it is no longer acceptable for the tests to be considered "self-documenting". Whenever the number of participants increases, the project's risks become exposed to its members' different interpretations of the requirements – the definition of what constitutes "success" can be misconstrued.

As the size of the project increases, so would the amount of test code that needs to be developed. Any test code developed needs to be supported for the life of the application – effectively doubling the ongoing maintenance effort.

As the size of the testing workload increases the project needs to add test automation to its armory, to minimize the human intervention and elapsed times for each of these testing cycles.

Myth Ten: "Developers and Testers are like oil and water"

Since the dawn of time there has often been a "them and us" tension between developers and testers. This is usually a healthy symbiotic relationship which, when working correctly, provides a mutually beneficial relationship between the two groups resulting in a higher quality deliverable for the customer.

The discussion should be focused on:

- Software delivery that meets business objectives (fit for purpose, on time and on budget), not who owns which part of the process.
- What can testers contribute to the requirements gathering phase to ensure they are involved in the TDD process?
- How can testers maximize reuse of the assets created during the development phase?
- Is there a role for the "traditional tester" in TDD, or should they (like the developers) be acquiring new skills to enable them to adapt to the new paradigm?
- How can developers and testers find mutual benefit in exploiting new advances in software development and testing tools?

Just as improvements in developer's software tools and methods have enabled a shift in development approaches, next generation technology for test automation is similarly reframing the opportunities for testers to automate earlier in the delivery cycle without incurring the heavy burden of script   maintenance so often associated with traditional automation tools.

Conclusion

Agile projects are in fact an excellent opportunity for QA to take leadership of the agile processes; who else is better placed to bridge the gap between users and developers, understand both what is required, how it can be achieved and how it can be assured prior to deployment? QA should have a vested interest in both "the how" and "the result", as well as continuing to ensure that the whole evolving system meets the business objectives and is fit for purpose. But it requires QA professionals to be fluid and agile themselves, discarding previous paradigms and focusing on techniques to optimize a new strategy to testing.

* Definition taken from The Glossary of Software Testing Terms from Original Software (www.origsoft.com)

**Defined on Wikipedia as "A software development technique consisting of short iterations where new test cases covering the desired improvement or new functionality are written first, then the production code necessary to pass the tests is implemented, and finally the software is refactored to accommodate changes."

About Original Software

Original Software offers automated software testing and quality assurance solutions that deliver tangible benefits across a wide range of IT and application environments. As a recognized innovator, Original Software's goal is to reduce business risk and improve application time to market for IT departments through the development of class-leading automated solutions.

Over the last 10 years, more than 400 organizations operating in 25 countries have come to depend on Original Software for their software testing solutions. Current users range from small software development shops to major multinationals, including Cargill Global Financial Solutions, Circuit City Stores, Pfizer Pharmaceutical (Ireland), DHL, Coca-Cola, Skandia and hundreds of others.

Original Software operates central offices near Chicago and London. Their solutions can be obtained through these offices or through a network of qualified and knowledgeable business partners throughout Europe, the Middle East, Australasia and the Americas.

The company's TestDrive solution is ideally suited to complement the test-driven approach familiar to agile developers. Because of the code-free scripting and the self-healing script capabilities, companies are finding that TestDrive's tolerance to change enables it to be introduced far earlier in the delivery cycle than a traditional automation tool. TestDrive's assisted manual testing can be used by developers as soon as they have the user interface available.

www.origsoft.com
www.manualtesting.com