WHITE PAPER ON The Reality of Software Testing in an Agile Environment (SHARED)

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 emphasising 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 >>
*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."
two
All the agile approaches have (at least) one characteristic in common in that they impact the role of the QA professional.
"
"
Original Software Ten QA Myths of 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 recognise the need for their armory to include a range of testing techniques. For example, Scott W. Ambler has a list of twenty-one 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.
three Original Software Ten QA Myths of Agile Testing
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 recognised 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.

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. The argument, that you need TDD to prove the requirements are captured accurately, isn't substantiated by history.

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. Whichever

approach you choose, 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 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.
five Original Software Ten QA Myths of Agile Testing
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.
six Original Software
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". "
" Ten QA Myths of Agile Testing
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 optimise a new strategy to testing.

seven Original Software
...next generation
technology for test
automation is similarly
reframing the
opportunities for testers
to automate earlier in
the delivery cycle...
"

" About Original Software

With a world class record of innovation, Original Software offers a solution focused completely
on the goal of effective quality management. By embracing the full spectrum of Application Quality Management across a wide range of applications and environments, the company partners with customers and helps make quality a business imperative. Solutions include a quality management platform, manual testing, full test automation and test data management,
all delivered with the control of business risk, cost, time and resources in mind.

More than 400 organisations operating in over 30 countries use Original Software solutions. Current users range from major multi-nationals to small software development shops, encompassing a wide range of industries, sectors and sizes. We are proud of our partnerships
with the likes of Coca-Cola, Cargill, HSBC, Unilever, FedEx, Pfizer, DHL and many others.

Original Software
Grove House, Chineham Court, Basingstoke,
Hampshire, RG24 8AG, United Kingdom
Tel: +44 (0)1256 338666
Fax: +44 (0)1256 338678
email: solutions@origsoft.com
www.origsoft.com
Original Software
Executive Place III, 1010 Executive Court,
Suite 230, Westmont, IL 60559, USA
Tel: +1 630 321 0092
Fax: +1 630 321 0223
email: solutions.na@origsoft.com
www.origsoft.com

Agile Testing Best Practices

Introduction

The testing phase of software development sometimes gets the short shrift from developers and IT managers. Yet testing is the only way to determine whether an application will function properly at deployment. Without an effective testing strategy, companies sometimes blindly take on significant risks that go well beyond simply having poor functioning software. Organizational ramifications can be fierce, including the risk of:

- Customer/end user alienation, or brand injury,
- Competitive threat, and even
- Product revenue loss.

At the development level, an oversight in testing can trigger countless technical and project outcome challenges:

- Protracted development cycles, or the reverse, a shortened testing cycle to compensate for longer development phases,
- Missed use cases triggered by developer-tester misalignment or poor communication and resulting in compromised quality,
- Cost overruns or simply poor resource estimations.

Agile testing mitigates these risks and presents an effective solution to ensure that you achieve the technical, project and business goals of the development process. In this whitepaper, we discuss Agile Testing, why it's important, and *how* to implement it.

What is Agile Testing?

Agile Testing is a software testing practice that follows the principles of Agile software development. Agile development integrates testing into the development process, verses having it as a separate phase. Testing therefore is an integral part of the core software development and actively participates though out the software coding process. Agile Testing involves a cross-functional Agile team actively relying on the special expertise contributed by Testers. This allows the combined team to better meet the project's defined business, software usability, quality, and timeline

objectives. Agile teams use a “whole-team” approach to “bake in quality” to the software product. This approach allows the team to work at a sustainable pace because testing occurs in real time, allowing testers to collaborate actively with the development team and giving them an

ability to identify any issues and transfer those into executable specifications that guide coding. Testing and coding are done incrementally and iteratively, building up each feature until it provides enough value to release to production.

Good Agile Testing Practices and Traits

Effective agile projects generally address the following important elements:

- Includes Testers and QA Managers as full members of the Agile development team.
- Leverage Testers as active contributors in planning and requirements analysis.
- Instills team value whereby each team member is responsible for the results, including quality.
- Promote the importance of Testers and encourage continuous feedback sharing with the programmers and the customer.
- Testers actively participate in meetings to define the main business flows.
- Testers complete short iteration activities alongside developers.
- Encourage maintenance of traceability between the requirements, test cases and bugs.
- Testers contribute to user story improvements.
- Leverage the specialized skills of test-driven development, including unit testing, continuous integration and unit level.
- Leverage Automation testing as a key way to do regression testing.

How does Agile Testing work? The process diagram at the back of this paper describes how the process works? Please note that for the purposes of this paper, some steps of the Scrum have not been included in diagram.

1. The combined team, including testing, takes responsibility for analyzing the business requirements (e.g. user stories). They together define the sprint goal.
2. The QA team defines the testing scope (i.e. test plan). That is then validated by the whole team and the client.
3. Simultaneously, while the development team starts the implementation, the QA team begins work on the test case design. These are properly documented and handed over to the client and the development team for review. This is to ensure the complete test coverage avoids unnecessary or redundant test cases.
4. QA defines along with the development team and the client which main flows (test cases) will be automated.

5. When code is ready to test, QA agrees with development to do quick testing on development environment, in order to identify the early stage defects so developers can fix them during the next round, on priority basis, and then progressing with further development. This iteration continues until the end of the code implementation.

6. Automated test cases are run daily. Any defect found is reported and fixed, based on its priority.

7. The QA team then begins testing on the QA environment. Any defect at this stage is again reported. At the end of the iteration the team determines, along with the client, which defects are to be fixed in the current iteration.

Automated Testing-Regression

Automation is a critical component of Agile testing. It would be impossible to keep pace with the Agile development schedule otherwise. Automation is used to run regression testing. The combined team (Developers, Product Owners and Testers) usually predetermine, at the start of the project, which parts of the software will be tested using automation.

Continuous integration/builds, unit, functional and integration test execution as well as continuous or automated deployment are common areas where automation may work better than traditional tests.

The entire project team agrees upfront on which of the main flows will be automated. They also determine at this point how to prioritize defects identified by automation, and how to fix that during sprints.

Automated tests consist of unit tests, capable of verifying even the most minute segment of software. Furthermore, it does so rapidly. This makes it possible to execute the test set multiple times per day, per hour or even more frequently if needed.

The benefits of automation include:

- Allows re-use of tests
- Enables faster execution for the most important test cases.
- Facilitates greater test coverage
- Delivers higher test accuracy and identifies defects sooner
- Facilitates regression testing

Advantages of Agile Testing

● Testing requirements are discussed and refined as a team (stand-ups / Scrums) allowing combined team to better address the business/technical aspects of the requirement. This enables overall alignment and prevents misunderstandings.

● Agile process often requires having an entry and exit criteria for stories (a compression of things to do in a particular release/iteration). Agile testing ensures that each requirement is well defined and is measurable, allowing you to determine whether the requirement was actually completed or not.

● QA participates in the big picture requirements writing stage. This ensures testing estimates aren't overlooked.

● Automated tests are fully leveraged to implement Regression.

● Quality becomes the combined team's responsibility, rather than just solely that of QA. The entire team agrees to the testing strategies, test cases and defects prioritization plan.

Disadvantages of Agile Testing:

If the ultimate goal or big picture requirements are unclear, the details can become muddled. Normally for new products the software architecture takes a path based on the initial requirements. If the requirements change (as allowed for in Agile), the following scenarios can occur:

● Engineering struggles to adapt to changes because significant effort has already gone into the initial requirements development and testing process.

● Challenge in estimations and sizing requirements given people dependency. Sometimes QA gets the short shrift since it's logically the last task in marking a user story done. Therefore, any delay in the prior development task risks impacting QA timelines. Sometimes, QA is prevented from executing a test case for the whole iteration, leaving QA to struggle to finish the task.

● Not asking the right questions. It very dangerous for QA not to ask questions, especially at the point where the use story is picked up for implementation. Ensure daily team meetings avoids this problem.

● Addition of new user stories into the current iteration. If QA is not included in the addition of the new use story, the appropriate commitments, estimations are not built in, resulting in misalignment and protracted timeframes.

Agile Testing Portfolio

Testing tools include:

Selenium, IBM-Rational Functional Tester, MS-Team Foundation Server, HP Mercury, Apodora, Ranorex, Eclipse, Watir, Watin, Paros proxy, Nunit, MS SQL,

Jira, Rally, TestLink, Jmeter, Pywinauto, Virtual Machines, WinSCP, XML, XLST, etc.
Relevant Technologies include:
C#, Java, Python, Silverlight etc.
The key message is that testing should be considered a crucial process to the final software outcome. Employing an Agile Testing approach can ensure overall alignment between Testing and Software Development, yielding overall project outcome and teaming benefits.