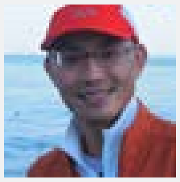
QASymphony

What You Need to Know About Software Testing in the Agile Era

About the Authors



Vu Lam is CEO of QASymphony, a leading developer of Quality Management solutions for software developers and QA testers. He was previously with First Consulting Group and was an early pioneer in Vietnam's offshore IT services industry since 1995. An entrepreneur at heart, he started multiple technology ventures, including Paragon Solutions, a technology consulting company with international development centers in Bangalore, India, and Ho Chi Minh City, Vietnam. Following Paragon's acquisition by NASDAQ traded FCG, he led the growth of the organization from 400 to 1300 resources. In 2011, he co-founded QASymphony, a vendor of software testing tools, with Josh Lieberman. He holds a MS degree in electrical engineering from Purdue University. You may reach him at **vulam@qasymphony.com**.

Sellers Smith is Dir. Quality Assurance & Agile Evangelist for Silverpop of Atlanta, GA, a digital marketing technology provider that unifies marketing automation, email, mobile, and social for more than 5,000 global brands. Prior to Silverpop, he led technical implementation of new Internet business initiatives for WebMD, ForestExpress.com, and the e-commerce site for NASCAR.com and GameTap.com. He has held software engineering positions at Turner Broadcasting, A.G. Edwards & Sons, and MITRE. Currently, Sellers is leading the implementation of Agile Testing practices including Acceptance Test Driven Development, Specification by Example, Test Automation using FitNesse and Exploratory Testing. He has a MS degree in Software Systems Engineering from George Mason University and a MBA in Information Systems from Rensselaer Polytechnic Institute. You may reach him at **sellers.smith@gmail.com**.

Chapters:

Introduction

The testing industry is undergoing some rapid and far-reaching changes driven by major shifts in the software development landscape. Software is moving from concept to end user faster than ever before and this throws up a whole new set of challenges for testers. The traditional procedures and techniques that have served QA departments in the past are no longer up to the job. The techniques and tools required for effectively testing modern software has evolved way beyond the traditional plan test – write test – run test model of yesteryear.

In this eBook we bring together the thoughts of a vendor -- intent on building something that truly speaks to the needs of the modern tester -- with a practitioner from the testing trenches, to bring you a multidimensional perspective on the issues testers are facing and how they might be mastered.

We begin with a chapter on reimagining the tester for Agile. The Agile mindset has transformed how software is developed, but QA departments have been slow to adapt their own methodology. Testers are uniquely placed to serve as advocates for the end user. Testing practices must be modernized; testers have to be in the development loop to gain the insight they need to test effectively, and realistic planning is paramount.

Our next chapter explores how testers need to identify the right balance of testing methods if they are to establish a comprehensive testing strategy. We discuss automated, exploratory, and user-acceptance testing, with advice on when to employ them and how to get the best from each approach.

In the third chapter we drill down into exploratory testing and analyze its origins to uncover its essence. The time pressures of modern development make it an indispensable technique. Continuous deployment and the fast feedback loop which drives new features, necessitates a flexibility and focus that can be found through exploratory testing.

The final chapter focuses on the practical side of building good tests by breaking the process down into a deceptively simple trio of questions: why, what, and how? By applying these queries testers can identify the value of individual tests to ensure that maximum value is extracted from each new test cycle. This ability to continually reassess and refocus effort in the right places is the key to delivering for the business.

QA departments are often left to their own devices. Excluded from exciting new developments and last to the table for budgetary consideration, testers are being squeezed to deliver more with less. The testing community has to get more active and vocal about the challenges facing them. A love of the craft and real dedication is the first step towards challenging that underappreciated status.

We offer this eBook in the spirit of opening a dialogue. The ultimate hope, which we'll delve into in more depth in our conclusion, is that the testing craft can be professionalized further. Good testers have key skills and expert knowledge that's in serious demand. By adapting to seismic shifts in the wider software industry, developing new approaches, and finding new tools, testers can stay relevant and effective.

# # #

CHAPTER 1

Reimagining the Tester for the Agile Age

Vu Lam

“The ideal tester is an advocate for the end user.”

It’s been 12 years since the Agile Manifesto was created, and the majority of developers have jumped on the bandwagon since then. Agile methodologies are now dominating the software development landscape. It has overtaken Waterfall as the dominant approach. But in the rush to implement the Agile mindset, something has been overlooked – the tester.

The focus on delivering a good product without endless documentation is undoubtedly a positive move for developers. By keeping the needs of the end user in sight at all times, and creating feedback loops, projects generally deliver better software, and faster than before. Given that the Agile method was conceived by developers for developers, it left the testers in the dark. As a report from Forrester points out, Agile teams must invite testers to the table.

Dangerous Assumptions

It should be obvious that Agile development completely disrupts traditional testing practices, yet many companies have done nothing to update processes or arm testers for the new challenges they are facing.

Imagine a typical scenario on an Agile project with a team of five developers, and a pair of testers. From the development point of view, five developers can work at a steady, measurable velocity, adding new features to each new build. But that simply isn’t true for testers.

At first, testing each sprint is not a problem, because testers have a limited number of features to test, but as the software grows they have to test new features, verify bug fixes, and perform regression testing. Everything has to be tested within the sprint timeline; the shorter it is, the more rounds of regression testing is called for. Soon, the test team is bogged down and overloaded.

Automated Unit Tests developed alongside with the code helps to relieve some of the pressure, but the workload for the testers still remains high. How can the same two testers cover all the new features alongside fix verification and regression testing? The answer is simple: they can't.

A New Approach
In order to ensure that the quality remains high, management could scale up the test team significantly as the development progresses, but there is a much more cost-effective solution. Regression testing has to be documented and then automated as you progress. There's no way to write up these test cases or scripts before the code has been delivered.

New feature testing and bug validation is where you want your testers to focus. You can avoid duplicating work by using the test flow they are capturing as they test to generate test cases that can be used to automate the regression testing for the next build. It's an all-at-once approach that requires testers to be on their toes.

A New Tester
For Agile testing to work, testers need to adopt the same Agile mindset as developers, but ditching the traditional waterfall process for sprints does require some upfront planning.

Previously, the testers' routine involved reading and understanding requirements, writing test cases before the build arrived, and testing the software when the build was made available. For the Agile project, the tester cannot afford to write test cases before the build arrives since there's not enough detail available.

There has to be a change in routine: leverage Exploratory testing to put new functionality through its paces, write a limited set of core test cases for future regression testing, and automate them as soon as the software in that area is stable to reduce a manual testing burden.

In the absence of detailed requirements and documentation, Agile testers must understand the needs of the end user. It is vital they can put themselves

in the end user's shoes and have a good understanding of exactly what they need from the software and how they are likely to use it. This means full participation in Scrum meetings, absorbing user stories, and asking questions whenever something isn't clear. The ideal tester is an advocate for the end user.

For Every Nail There's A Hammer
Looking beyond the mindset, Agile testers need the right tools for the job. The vast majority of Agile project management software ignores the vital role of the tester. Test management solutions were created to support more traditional testing. In short, project and testing software have not caught up with the unique demands of the modern Agile tester.

They need tools to support the way they explore and test software in a fluid, unscripted manner. It should be fast and easy to identify and document test cases for regression and possible automation. They also need a test management tool that fully supports all the Agile tester's tasks, not a tagged-on module to the development team's project management software. Something dedicated that can link user stories and test cases, import and export bugs, and keep a complete record of the testing that has been completed and what the results were. Something that's truly built for the Agile tester.

Time For Agile Testers
So much effort has been put into creating Agile practices and tools for developers, isn't it about time that testers were given due attention? For all the good that Agile development brings, it doesn't dispense with the need for skilled testing if you fully expect to deliver an excellent product for users.

Integrate testers into the Agile process from the start, equip them with the right tools to help them develop exploratory skills and use automation to cope with the repetition of regression testing, and you have yourself a tester for the Agile age.

# # #

CHAPTER 2

Finding the Right Mix with Your Testing Methods

Sellers Smith

With the right balance of automated, exploratory, and user-acceptance testing, QA teams can help build great products.

Any comprehensive testing strategy is going to rely on the right mix of tests for the product. There are various methodologies that can be employed, but rather than viewing them as separate, discrete approaches, it is worth remembering that they all lie on a common continuum. At one end we have mechanistic automated testing, and we pass through scripting, beyond exploratory testing, and on to private and public betas at the opposite end.

The testing journey starts from a tightly controlled base and winds its way towards real world conditions. We're going to focus on three important stops along the way in this article and explain a little about how they bring value to the process, what their weaknesses are, and how you decide when to use them.

Automated Testing
The testing of every product will start with some scripted, automated tests that are designed to exercise the system and validate that it works the way it is supposed to. Automated tests are repeatable, fast, and they can be run as many times as needed.

To give an example, if you were to test an e-commerce shopping cart, you might have an automated test that tries to add each individual item in your store and check out successfully. Does the cart add up the combined cost correctly? Is it possible to add each item? Can you successfully complete the checkout process?

These are all useful things to know, but there are some obvious limitations. Automated tests cannot go beyond their original scope. The lack of human involvement means that an automated test may run and successfully add items and then check out, but it wouldn't pick up on things like misspelled item names or stretched photos. If it wasn't included as a parameter when the automated test was written, then it won't be picked up when it runs.

Exploratory Testing
With any piece of software you can use documentation, user stories, and conversations with key stakeholders to determine how a system should work. There are certain concrete rules to how the software should behave and automated or scripted testing can cater for those, but what happens when someone goes off script? Exploratory testing is about combining your knowledge and experience as a tester with an understanding of what the product is supposed to do to uncover potential problems that automated tests will not catch.

It goes beyond ad hoc testing, because there has to be a clear area of focus and a predetermined scope, but the tester is not bound by a check-list approach. They can delve into how the experience feels, unusual scenarios that may have been missed by the developers, and leverage specialist knowledge related to the domain or industry that the software is aimed at. Using our shopping cart analogy again they might add and then remove products from the cart, test how it deals with multiples, and see if they can exploit the system to produce negative values and checkout with a refund.

User-Acceptance Testing
There is ultimately no substitute for real world conditions. You can test a product professionally for months on end, but as soon as you release it into the wild your end users will do a variety of unexpected things with it. Whether you opt for a private beta, which involves inviting some of your prospective customers to use the software, or you open it up publicly and work fast to meet expectations, you need to collect this feedback before your product can truly be finished.

End users will always come up with scenarios that development and QA didn't consider. With the shopping cart, they might fill it up and then come back a week later with the expectation that their shopping items will still be waiting for checkout. Unencumbered by knowledge of the documentation, and not biased by proximity to the project, end users will try unusual things that seem intuitive to them. That can be great for discovering defects, but it also teaches you a lot about expectations. This is how your product will be used in the real world.

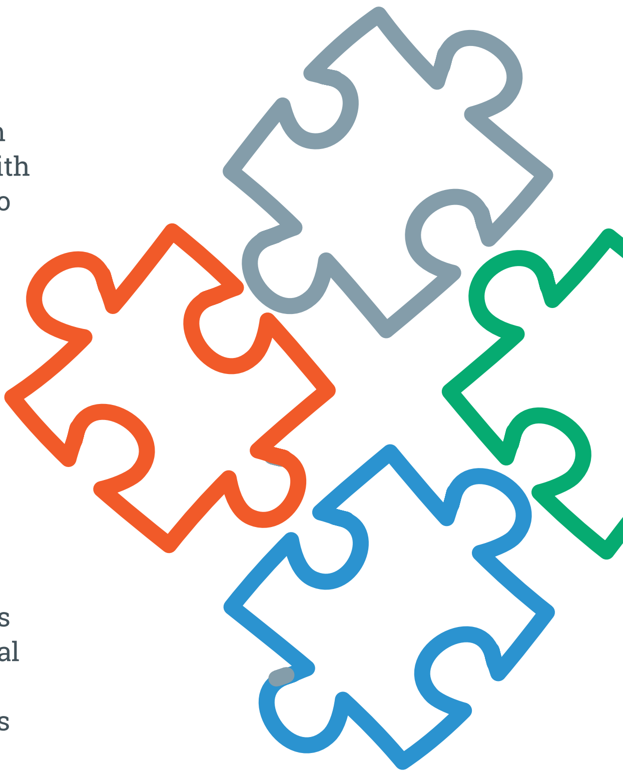
Bringing It All Together
The exact mix of tests will vary depending on the type of system and users. Generally, systems with heavy business logic are going to favor more test automation (e.g., verification of business rules), while consumer-facing systems will favor more beta testing.

As an example, Silverpop provides marketing automation and email marketing in a SaaS model. The company focuses on a larger automation test suite, and does exploratory testing in order to test complex usage of its application. Silverpop has several mechanisms, things like feature switches and pilot environments -- to support beta users and early adopters.

A good approach is to focus on automated tests (or scripted tests), until you have a generally usable platform. Exploratory testing is of limited use if the basic features are not working. Once exploratory testing is underway, you can focus on introducing beta user or early adopters. Feedback from beta users is of limited value if primary usage scenarios are not working. This approach can be used iteratively around release and or features (for folks doing feature-based development).

In short, start with a well-defined set of automated tests to validate core business functionality, shift to exploratory testing as the core functions are working, and then begin introducing real users as the exploratory testing shows an acceptable level of usability.

# # #



CHAPTER 3

The Surprising Truth About Exploratory Testing

Vu Lam

Exploratory testing is an important tool in any experienced tester's kit. In the modern age of Agile development, with software being updated and released faster than ever, exploratory testing is nothing short of essential. The trouble is that there seems to be some confusion over what the term actually means.

What Is Exploratory Testing?

Real exploratory testing is not the same as ad hoc testing; it's much more sophisticated and organized than that. The term was originally coined by Kem Caner in his book, Testing Computer Software, to describe an alternative approach to traditional formal planning. He implored testers to "Trust your instincts." But he also warned that you should "always write down what you do and what happens when you run exploratory tests."

Another author, James Bach, described exploratory testing as "scientific thinking in real-time." In his 1999 paper, General Functionality and Stability Test Procedure, Bach expounded on that, explaining that "unlike traditional informal testing, this procedure consists of specific tasks, objectives, and deliverables that make it a systematic process..." and went on to say, "In operational terms, exploratory testing is an interactive process of concurrent product exploration, test design, and test execution. The outcome of an exploratory testing session is a set of notes about the product, failures found, and a concise record of how the product was tested. When practiced by trained testers, it yields consistently valuable and auditable results."

More than a decade has passed since Bach wrote those lines, yet his wisdom is still lost on many people.

When To Employ It
It's very common for testers to be under time pressure, and the shift to Agile methodology in software development has had a tangible impact on documentation. To put it simply: testers have very little time to test a new build and must often do so without access to detailed documentation.

Exploratory testing is ideal for this scenario, but should not be an exclusive approach. It complements scripted testing activities. Exploratory testing is simply part of an overall plan that must include new feature exploration, bug validation and regression testing.

If you're not sure what to test next, or you want to dig deeper into a complex piece of software, then it's time for some exploratory testing.

How To Do It Right
What you're really looking to do is design and execute tests at the same time. Although you won't be using requirements to prepare detailed test cases before you begin, that doesn't mean you should test randomly. There should be a focus on specific parts of the product, or an agreement to employ specific strategies during the testing. Combine your intuition with the knowledge you have amassed on the software from previous tests, and your perceptions about the needs of the end user. You must understand what the software is supposed to do. Then you can appreciate how developers tried to solve specific problems and decide whether they accomplished their objectives. By leveraging your insight and targeting your exploration, new issues will be discovered. This is not a random bug hunt.

Good testers will naturally develop the skills required for exploratory testing; the ability to think creatively and generate useful tests in real-time. A dogged, rigid approach to testing software, especially software that is rapidly evolving, does not make the best use of a tester's intellect, and won't result in the best final product possible.

Extracting Value
You've established your boundaries and the scope of your testing session upfront. In order to get the maximum benefit from your discoveries you have to generate reports of your findings.

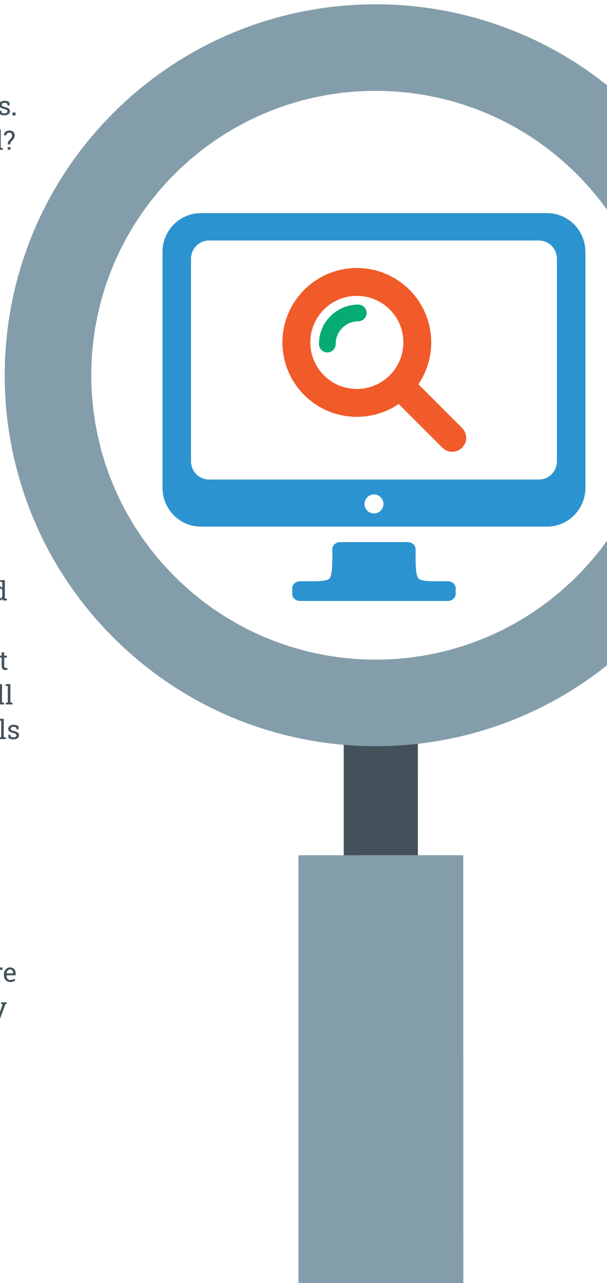
You should be recording your expectations and the actual results. Were you able to achieve your goal? Did it take longer than expected? Did you encounter weaknesses in the software? Remember that another tester, or a member of the development team, should be able to follow your thoughts and reasoning. Clarity is vital.

Because you have a frame of context the data you generate can really benefit the development team. Instead of the isolated defects, suggestions, and concerns that an ad hoc testing approach might generate, you get in-depth information on how well the software meets intended goals and how it will be perceived by the end user.

In addition to verifying software and discovering issues, exploratory testing can help identify useful test cases for future regression testing. An exploratory testing session can be the bridge between unscripted and scripted testing for your project.

The Right Approach
The surprising truth is that exploratory testing encourages the right approach to testing any software. You need to build a real insight into the product and think imaginatively about how it will be used. Combine that creativity with metrics and detailed reporting, and you can see the big picture. Unlike ad hoc testing, exploratory testing is an organized, well-structured approach that lets testers excel in the ever changing landscape of agile development.

# # #



CHAPTER 4

Building Good Tests: Why, What, How?

Sellers Smith

Good testers and good tests always retain and use an awareness of what the intended audience wants and expects.

Tests are an investment in the quality of any given system. There's always a cost to build, run, and maintain each test in terms of time and resources. There's also a great deal of value to be extracted from running the right test at the right time. It's important to remember that for everything you do test, there's something else you're not testing as a result.

Understanding that some tests are more important than others is vital to creating a useful and fluid test plan capable of catering for modern software development techniques. Traditional waterfall development -- where everything is delivered for a specific release date in a finished package -- has been succeeded by continuous feature roll outs into live systems. This necessitates a different approach from QA departments.

How Do You Build Good Tests?
You can't design or execute the right tests without understanding the intended purpose of the system. Testers need to have an insight into the end user's expectations. Communication between the product people at the business end, the engineers working on the code, and the test department enables you to score tests in terms of their importance and work out where each test cycle should be focusing.

We can break it down into three simple queries: why, what, and how.

"Why" is a higher level overview that really ties into the business side. It's the big picture thinking that reveals why you're building the software in the first place. What audience need is your product fulfilling? For example, we need to build an e-commerce website to sell our product to the general public.

"What" is really focused on individual features or functions of a system. Using a shopping cart analogy for an ecommerce website, you might say that users must be able to add and remove items from their shopping cart, or that they shouldn't be able to add something that's out of stock.

"How" relates to the practical application of your testing. How exactly is the software going to be tested? How is the success and failure measured?

Good tests are always going to cover our trio, but it can be a useful exercise to break things down.

The Why

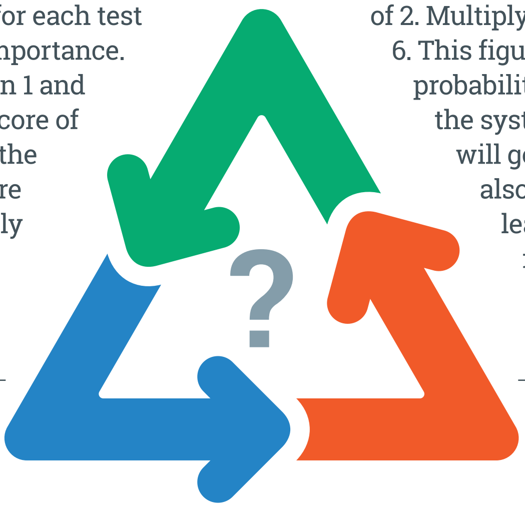
If you get too caught up in the "what" and the "how" it's possible to miss the "why" completely and it's the most important element because it dictates that some tests are more important than others. The business case for developing your software in the first place has to remain front and center throughout the project. If you begin to lose sight of what the end user needs, then you could be focusing your efforts in the wrong places. Delivering value to your customers is critical. Good testers and good tests always retain and use an awareness of what the intended audience wants and expects.

One technique we can employ is risk-based analysis of tasks. With risk-based analysis, we can arrive at a numerical value for each test which gives you a sense of its importance. We can assign a score of between 1 and 9 to each test. At the top end, a score of 9 would be a critical test, and at the other end of the spectrum, a score of 1 might indicate a test that only needs to be used sparingly.

The value is determined by multiplying two factors:

- **Impact to the user** – what are they trying to accomplish and what would the impact be if they couldn't? How critical is this action?
- **Probability of failure** – how likely is it that this code will fail? This is heavily influenced by how new it is and how much testing it has already undergone.

If we return to our ecommerce website analogy then we could take the action of being able to buy goods, clearly that's essential, so it would be assigned a 3. However, the functionality for adding goods to the basket and checking out has been there since day one, so it has already been tested quite extensively, but some new features have been added which could impact on that code, so that might result in a score of 2. Multiply the two together and you've got a 6. This figure will change over time, because probability of failure will go up if this part of the system is significantly altered, and it will go down over time if it isn't. There's also a discretionary factor that might lead you to bump that 6 up to a 7 if you feel it's merited.



The What

Testers come up with specific scenarios of how an end user might interact with the software and what their expected outcome would be. A typical test may consist of many steps detailed in a script, but this approach can cause problems. What if a new tester comes in to run the test? Is the intent of the test clear to them? What if the implementation of the feature changes? Perhaps the steps no longer result in the expected outcome and the test fails, but that doesn't necessarily mean that the software is not working as intended.

The steps and scripts are delving into the "how," but if the "what" is distinct from the "how" then there's less chance of erroneous failure. Allow the tester some room for an exploratory approach and you're likely to get better results. If something can be tightly scripted, and you expect it to be a part of your regression testing, then there's an argument for looking at automating it.

Adopting a technique like Specification by Example or Behavior Driven Design, you're going to lay each test out in this format:

- **Given** certain preconditions
- **When** one or more actions happen
- **Then** you should get this outcome

Regardless of the specifics of the user interface, or the stops along the way between A and Z, the "Given, When, Then" format covers the essential core of the scenario and ensures that the feature does what it's intended to do, without necessarily spelling out exactly how it should do it. It can be used to generate tables of scenarios which describe the actions, variables, and outcomes to test.

(Continued on next page)

Building Good Tests: Why, What, How?

(Continued from pg. 11)

The How
Getting down to the nuts and bolts of how testers will create, document, and run tests, we come to the “how.” Since projects are more fluid now and requirements or priorities can change on a daily basis, there needs to be some flexibility in the “how” and a willingness to continually reassess the worth of individual tests. Finding the right balance between automated tests, manually scripted tests, and exploratory testing is an ongoing challenge that’s driven by the “what” and the “why.”

Traditionally, manual tests have been fully documented as a sequence of steps with an expected result for each step, but this is time consuming and difficult to maintain. It’s possible to borrow from automation by plugging in higher level actions, or keywords, which refer to a detailed script or a common business action that’s understood by the tester. There’s also been a move towards exploratory testing, where the intent of the test is defined, but the steps are dynamic. Finally, there’s a place for disposable testing, where you might use a recording tool to quickly document each step in a manual test as you work through it. These tests will need to be redone if anything changes, but as it’s a relatively quick process, and you’re actually testing while you create the test, that’s not necessarily a problem.

Continuous assessment
Each individual test should be viewed as an investment. You have to decide whether to run it, whether it needs to be maintained, or if it’s time to drop it. You need to continually assess the value and the cost of each test, so that you get maximum value from each test cycle. Never lose sight of the business requirements. When a test fails, ask yourself if it’s a problem with the system or a problem with the test, and make sure that you always listen to the business people, the software engineers, and most importantly the customer.

# # #

Conclusion

Every decade or so, we see a radical new idea or technology emerging that completely changes the software development scene, and moves the goalposts for testers. A trio of trends, in the shape of cloud computing, Agile development methodology, and the explosion in mobile devices, have converged to give us software that is put together quickly and made available cheaply. Drastically cutting the time from concept to market has a big impact on the way software is tested and a big impact on testers.

The waterfall method necessitated meticulous planning from detailed requirements, and a linear development schedule that dictated testing should follow the design and implementation phases. It's hopelessly outdated.

We are in the midst of this transformative period and if you find that you're struggling, then it's time to realize why. You need to catch up on what is changing, understand how you can be a better tester, and adopt new techniques and tools that will help you to be as valuable as possible to your company.

It's not a level playing field. There are lots of conferences for developers, plenty of networking opportunities, magazines and online articles sharing new ideas. There's a whole community that targets developers with new tools and products, a community that's geared towards getting developers up to speed on new languages or methodologies. Unfortunately, the support offered to developers does not match or exist on the same scale for testers.

Too many testers have their heads down, working away on their own projects. There's a lack of appetite for forum discussions, conferences, or new techniques. This lack of a community evangelizing new procedures and spreading new ideas, makes it that much harder for best practices and exciting techniques to spread. The field is relatively slow to modernize. New trends are in motion, and there are QA professionals pressing for change, but many testers unfortunately are not aware it.

In order to spark the same level of activity and even controversy in testing that we see in software development, we need to work together to drive the craft to become more professionalized. Generally speaking, testing roles may not be valued in the same way that developer roles are, but we can push to change that. Seasoned testing professionals can act as advocates for that change, open lines of communication, and agitate for the whole community to pull together.

The testing profession should move to an expertise value base, focus on improving skills and knowledge to create better career paths and ultimately take greater satisfaction and pride in the important work it does. Finding the right blend of techniques and tools, the perfect marriage of process and function cannot be achieved if we can't get beyond testing as an afterthought, or an addendum to development.

It's time to reevaluate how we approach testing. Consider tools that have been designed specifically to meet the needs of real testers in the field, and blend techniques that are capable of providing the speed, transparency and flexibility that modern software development demands. Our comprehensive test management solution, qTest, was designed to empower modern QA departments with the features and functionality they demand. We also offer qSnap, a completely free, cross-browser snapshot tool that enables testers to capture screenshots, edit, annotate, and share them directly. We welcome your input and feedback as we strive to produce tools that put testers first.

We hope you found this eBook useful and welcome your comments and input. We hope to foster a greater spirit of community across the testing profession.

# # #

QASymphony

Join the conversation!

For more information on
QASymphony please contact us at:

Email: **info@qasymphony.com**

Phone: **1-888-723-8654**

Web: **www.qasymphony.com**