SHOWCASING THOUGHT LEADERSHIP AND ADVANCES IN SOFTWARE TESTING

LogiGear MAGAZINE

Testing In Agile

Disciplined Agile Testing

By Scott Ambler, Chief Methodologist for Agile and Lean, IBM Rational

Agile, Testing and the Need for Speed

By Michael Hackett, LogiGear

Ten Tips for Agile Testing

By Ralph van Roosmalen

Agile Testing: Key Points for Unlearning

By Madhu Venantius Expedith

July 2012 | Volume VI | Issue 3

LETTER FROM THE EDITOR

Worldwide Offices

United States Headquarters
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +01 650 572 1400
Fax +01 650 572 2822

Viet Nam Headquarters
1A Phan Xich Long, Ward 2
Phu Nhuan District
Ho Chi Minh City
Tel +84 8 3995 4072
Fax +84 8 3995 4076

Viet Nam, Da Nang
7th Floor, Dana Book building
76-78 Bach Dang
Hai Chau District
Tel +84 511 3655 33
Fax +84 511 3655 336

www.logigear.com
www.logigear.vn
www.logigearmagazine.com

Submission guidelines are located at http://www.logigear.com/magazine/issue/news/editorial-calendar-and-submission-guidelines/

What is testing in Agile? It's analogous to three blind men attempting to describe an elephant by the way it feels to them. Agile is difficult to define and everyone has their own perspective of what Agile is. When it comes to testing and Agile the rules are what you make them.

Agile is ideas about software teams working together with no details. SCRUM is a framework for managing software projects. Extreme Programming (XP) has practices about making the product and coding. Agile says nothing about testing. SCRUM definitely says nothing about testing. There is no *role* in SCRUM called *Tester*, QA, or QE! Extreme Programming comes the closest, but the focus of testing is on developer focused TDD (test driven development) through unit testing. The work of *traditional testers* is not defined by any of these famous practices or frameworks. We are making it up as we go.

While Agile evokes images of happy, self-directed teams working well together, there are problems. The Agile manifesto has been around over 10 years. SCRUM has also been around long enough to have been tried, failed, succeeded, abandoned, and tried again.

Two dirty words to describe the practices of any team are Agilefalls and Scrumbuts. Agilefalls is a practice that badly mixes agile and waterfall. Scrumbuts is the famous descriptor coined by Ken Schwaber (co-creator of SCRUM) of people adapting SCRUM as a loose process to suit whatever needs they have. SCRUM is anything but a loose process. As the 2011 Update (available at www.scrum.org) says, "The Scrum Guide is the definitive rule book of Scrum and the documentation of Scrum itself."

No one can tell you there is a definitive way to test in Agile projects. There simply isn't a best practice here. The better practices for your team are going to be based on many things. Among them; how much you automate, to how you implement *Lean* and the type of documentation your team writes, to the *definition of done*. How you test and how much you validate will probably be left up to you! Here is where we come in. We asked some insightful thinkers to share their perspective on testing in Agile to help you put together a test strategy or improve the one you have.

In this issue, IBM's Scott Ambler discusses the role of discipline in your Agile testing strategy; Pankaj Nakhat talks about the paradigm shift test teams must undertake for successful Agile projects; Ralph van Roosmalen looks at how to better organize and track the Agile process; I address the misconception that Agile is about speed; John Roets presents alternative office layouts that can promote knowledge sharing on Agile teams; Madhu Venantius Laulin Expedith examines aspects of traditional testing that must be unlearned when switching to Agile development; Gunnar Peipman reviews the book *Clean Code: A Handbook of Agile Software Craftsmanship* written by Robert C. Martin; and finally, I'll look at the manager-specific responses from our 2010-2011 global survey.

Whether a true-believer or late adopter, there is no arguing that the tasks test teams used to do in traditional projects are not defined. So, what do we do? In true SCRUM style, we try things. Hopefully we fail fast. To fail fast is to try something full-speed. If it works, great. If not, recognize it failed and try something else.

Testing in Agile projects is evolving. What we do, how and when we do it, is maturing. We need to constantly reflect on our practices. Keep what works, discard what does not. Share experiences and continuously improve. I hope this issue gives you food for thought and that other opinions and experiences help improve your practice of testing in Agile projects.

Michael Hackett
Senior Vice President
Editor in Chief

IN THIS ISSUE

IN THE NEWS

Chris Floyd Interview on Implementing Agile Development

Michael Hackett sat down with FNC's Chris Floyd to get his take on numerous Agile topics.

Chris address some of the burning questions surrounding Agile, such as teams switching from waterfall to Agile, challenges and successes his team encountered after implementing Agile, and the path to becoming a ScrumMaster.

Chris Floyd has worked in the financial services sector for 11 years. He loves to eat at dives, and in his spare time he trains diabetic alert dogs for Wildrose Kennels.

The video can be found here: http://www.logigear.com/magazine/agile/chris-floyd-interview-on-agile-testing/

LogiGear Agile and Testing Research Center Now Live

With the goal to provide our readers with the best and latest information about testing methodologies, we're launching the "Agile and Testing Research Center" on our website where you can find articles, videos, and more relating to all things Agile.

Our readers are active seekers and creators of software testing information. If you've written on this topic and think it would be helpful to your peers, please feel free to submit an article to us by emailing: logigearmagazine@logigear.com

To access the resource center, visit: http://www.logigear.com/lp-agile-and-testing-resource-center.html

Mobile Application Testing with TestArchitect

Mobile application development is booming, and so is the need for testing. LogiGear's TestArchitect now includes SDK UI controls for Android 2.1 and later! Currently in development are controls that will allow testing of iOS applications.

With TestArchitect, it's now possible to automate testing for mobile applications by utilizing keyword driven testing. With TestArchitect, it's possible to quickly create and update tests via a keyword UI. The result is significant time and cost saving, and better software.

One of the big advantages of TestArchitect is that it is the only test automation tool that is test module centric, not test case centric. The modular approach separates actions, interfaces and test lines making it possible to design and write tests before functionality is complete in order to support Agile development. If you are looking for a fast and scalable way to automate testing for mobile and desktop applications, call us at 1-800-322-0333 to request a trial copy.

Full 2010 - 2011 Global Survey Results Now Available

In 2010 and 2011, we conducted a broad survey called "*What is the current state-of-the-practice of testing?*" We began the survey in the first part of 2010 and collected data for an entire year. Invitations were sent out to testers all over the world.

Software testing is a global trend, with experts and engineers having widely differing ideas on how best to perform certain methods and practices. We wanted to capture and understand that diverse cross-section of ideas. From offshoring to Agile, we tackled some of the testing industry's hottest topics. We've assembled all the sections of the survey, and you can now download it in its entirety as a PDF from: http://www.logigear.com/magazine/category/issue/survey/

BLOGGER OF THE MONTH

A Tester's Perspective on Agile Projects

Agile is a philosophy focused on delivering constant value to customers incrementally and frequently, based on communication and feedback. These two ingredients are vital to a successful Agile recipe.

By Pankaj Nakhat

Agile is no longer a buzzword or an unknown territory in the industry. Agile has progressed leaps and bounds the last few years and has matured to a widely accepted methodology. Testing in Agile projects required a paradigm shift for traditional testing roles. It required a change in tester's attitudes from a relay race oriented approach to an upfront involved role. The Agile approach focuses on getting things right the first time, reducing the need for QA testers to get something over the finish line. But it's easier said than done. How does it happen in reality? Does it actually happen?

Agile is a philosophy focused on delivering constant value to customers incrementally and frequently, based on communication and feedback. These two ingredients are vital to a successful Agile recipe and testers have an important role to play in creating value in Agile projects throughout different phases of iteration.

I have outlined the QA activities as three phases in Agile projects. However, this is no golden rule and it can be flexible according to the project situation. The Agile QA tester's role is not limited to a set of pre-defined processes, as the methodology will dictate roles based on the situation.

Pre-iteration: This is the time where requirements are analyzed in detail by the BAs and acceptance criteria are detailed for that story. As QA testers are immediate consumers of those requirements, it is important to verify the requirements early and often.

Story verification (Requirement Verification): Agile testing is all about giving feedback early, not necessarily only by testing the requirements, but doing requirement testing early. QA testers really need to look at the requirement / stories upfront for clarity and testability. This will determine the requirements that are unambiguous and testable (I believe unambiguous in Agile is not much different in context compared to a typical waterfall project).

- Requirements should be small enough to make sense in the context
- Acceptance criteria (stories are generally used for acceptance criteria) should not be duplicated or overlapping from different stories

However, doing this can be very difficult and can only be achieved with strong communication between Development/Business Analysts/Quality Assurance.

Testable: The testability aspect of the story requires scanning through the story to see what needs to be done in order to test the story. These factors are generally:

- Finding hidden requirements
- Environment
- Test data
- Dependency on other requirement

Getting these details early helps the story to be prioritized accordingly in the backlog, and allows smooth execution of the story in the iteration.

QA testers also participate in the iteration planning meeting to give a testing perspective so the team can come up with a developer estimate. Participating in the iteration planning is a big role, as some of the implicit requirements are escalated by QA testers.

QA activities In the iteration

Once QA testers are happy with the acceptance criteria of the story, they can help define the acceptance tests for the story. Acceptance tests are requirements in terms of tests that can be executed in order to understand what is expected from the requirements. These acceptance tests are generally automated and used to drive development.

Acceptance tests should not cover too many corner case scenarios as this would create unnecessary delay and could end up producing too many similar automated tests.

People often fail to understand that acceptance testing in Agile projects is different from traditional projects. Unlike traditional projects where acceptance testing happens at the end of the software lifecycle, in Agile projects acceptance testing is performed before the software is delivered. Acceptance tests also tend to be automated so they can run as regression tests.

Automated testing is very important for any Agile project. Frequent builds require short feedback cycles, hence regression testing must be quick and accurate. It needs to be noted that automated testing in Agile is different from traditional automated testing. In Agile projects, automated testing is practiced by all levels - developers, QA testers and business analysts. Involvement from everyone increases the relevance of the tests and often helps identify the right tests. But, this does not mean that everyone needs to be writing test code.

It's always been debatable who owns test automation in Agile projects. For me, it's more of a responsibility than a role. In my experience, the most effective test automation is achieved when developers and QA testers work together.

Use automation purposefully

Automation in Agile can be quite controversial. Many people try to automate everything and end up in a trap of having a long feedback cycle. Automation is meant to provide early feedback on the latest code, and the key is to identify what is worth automating and what is not.

Every automated test has a cost associated with it. The cost of automation should be compared to the cost of not running it. The questions that need to be asked are: what if a test is not automated? What will be lost and what would be the cost of fixing the stuff around the code for which we are losing the coverage? Is it cheaper to test manually? The answers may not always be as straight forward as finding the value of a test. It's often a contextual decision depending on the size of the project and the number of people involved. In other words, a longer feedback cycle equals more people contributing time in getting instant feedback.

The typical QA activities in the iteration are to continuously measure the quality of the software. QA testers participate in the story handover to the developers. This helps them understand the testing requirements of the story so they are enabled for Test Driven Development (TDD). Also, handing over the acceptance tests and making developers understand the testability aspect of the story often helps catch common defects. These activities require a high level of communication between the developer and business analysts to clarify things on the fly and make sure the product is built right first time.

QA testers can help resolve issues beforehand by actively participating in the overall process. They may even pair up with developers to work on the story or tests for the story in order to get a better understanding of the requirements. It's also imperative that once the story is delivered, it is tested properly, in a proper environment. Once the QA testers are happy with the stories/requirements, they sign off on that piece of work and hand it over for further testing.

It is also important to think beyond the written requirements and experiment with exploratory testing to execute 'out of the box' scenarios and undertake negative testing to help assure the software is robust. Exploratory testing is not at all about executing pre-defined testing scenarios, it is the art of exploring the software beyond test cases and at the same time keeping the focus around specific requirements.

I have attempted to cover most aspects of the basic activities of a QA tester in an Agile project. However, the role is not limited to these activities and QA testers should play more of a collaborative role when possible. ■

About Pankaj

Over the past five years Pankaj has worked with leading organizations in verticals such as business intelligence, banking, retail and CRM. In addition to experience with tools like QTP, Selenium and Perl, he has also developed simple and business effective automation frameworks and conducted training on automation frameworks and advanced QTP for corporations. He has lead automation initiatives from scratch and provided consultancy on different projects during his tenure with RBS.

Cover Story

Disciplined Agile Testing

There is a multitude of Agile testing techniques that are quite sophisticated. The DAD process can help guide your process of tailoring decisions.

By Scott W. Ambler, IBM

Agile developers are said to be quality infected, and disciplined agilists strive to validate their work to the best of their ability. As a result they are finding ways to bring testing and quality assurance techniques into their work practices as much as possible. These strategies include rethinking about when to do testing, who should test, and even how to test. Although this sounds daunting at first, the agile approach to testing is basically to work smarter, not harder.

This article describes how to take a disciplined approach to tailoring a testing strategy best suited for your agile team. It adapts material from my existing Agile Testing and Quality Strategies article[1] which describes the strategies in greater detail than what I do here. I also put this advice in context of the Disciplined Agile Delivery (DAD) process framework[2], a new generation agile process framework.

Disciplined agile delivery

The DAD process framework provides a coherent, end-to-end strategy for how agile solution delivery works in practice. The DAD process framework is a people-first, learning-oriented, and hybrid agile approach to IT solution delivery. It has a risk-value lifecycle, is goal-driven, is scalable, and is enterprise aware. The basic/agile lifecycle for DAD is presented in Figure 1. As you can see it extends the Scrum lifecycle to encompass the full delivery lifecycle from project start to delivery into production. There is also an advanced/lean lifecycle which is loosely based on Kanban, but for the sake of this article the basic lifecycle is sufficient for our needs.

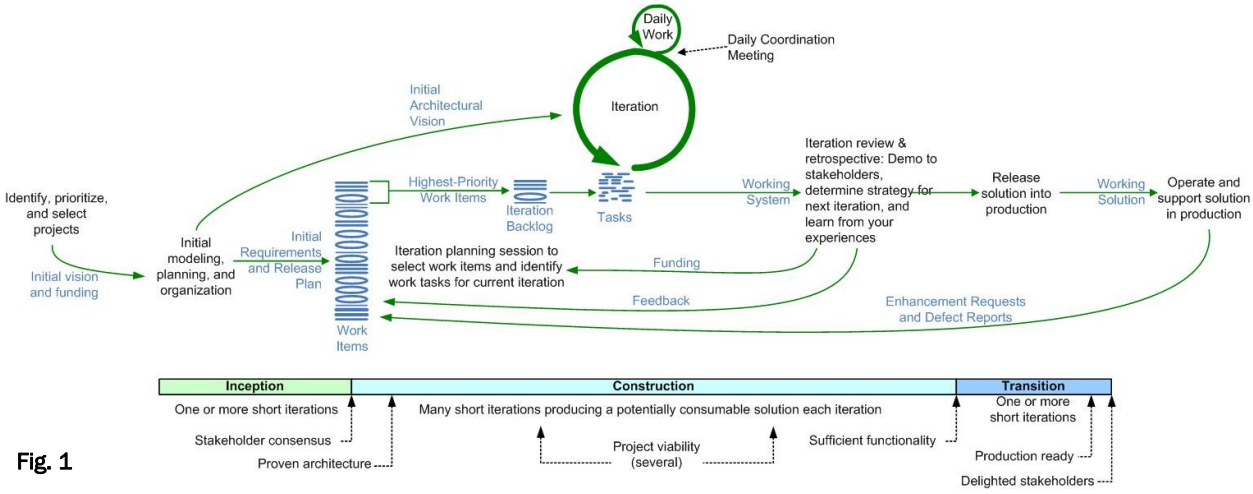


Fig. 1

Figure 1. The basic DAD lifecycle. (See Below)

On agile projects testing occurs throughout the lifecycle, including very early in the project. Although there will likely be some end-of-lifecycle testing during the Transition phase it should be minimal. Instead testing activities, including system integration testing and user acceptance testing, should occur throughout the lifecycle from the very start. This can be a significantly challenging change for organizations that are used to doing the majority of testing at the end of the lifecycle during a release/transition phase. So, the graphical depiction in Figure 1 aside, the Transition phase should be as short as possible – very little work should occur here at all.

An interesting feature of the DAD process framework is that it is not prescriptive but is instead goals based. So instead of saying "here is THE agile way to test" we instead prefer a more mature approach of "you need to test, there are several ways to do so, here are the trade-offs associated with each and here is when you would consider adopting each one." Granted, there is a minority of people who prefer to be told what to do, but my experience is that the vast majority of IT professionals prefer the flexibility of having options.

So what are some of your agile testing options? Although there is much written in agile literature about the merits of developers doing their own testing, sometimes even taking a test-first approach, the reality is that there is a wide range of agile testing strategies. These strategies include:

- Whole team testing
- Test-driven development (TDD)
- Independent parallel testing
- End-of-lifecycle testing

Let's look at each strategy one at a time and then consider if and when to apply them.

Whole team testing

An organizational strategy common in the agile community, popularized by Kent Beck in Extreme Programming (XP)[3], is for the team to include the right people so that they have the skills and perspectives required for the team to succeed. To successfully deliver a working solution on a regular basis, the team will need to include people with analysis skills, design skills, programming skills, leadership skills, and yes, even people with testing skills. Obviously this isn't a complete list of skills required by the team, nor does it imply that everyone on the team has all of these skills. Furthermore, everyone on an agile team contributes in any way that they can, thereby increasing the overall productivity of the team. This strategy is called "whole team".

With a whole team approach testers are embedded in the development team and actively participate in all aspects of the project[4]. Agile teams are moving away from the traditional approach where someone has a single specialty that they focus on – for example Sally just does programming, Sanjiv just does architecture, and John just does testing – to an approach where people strive to become generalizing specialists[5] with a wider range of skills. So, Sally, Sanjiv, and John will all be willing to be involved with programming, architecture, and testing activities and more importantly will be willing to work together and to learn from one another to become better over time. Sally's strengths may still lie in programming, Sanjiv's in architecture, and John's in testing, but that won't be the only things that they'll do on the agile team. If Sally, Sanjiv, and John are new to agile and are currently only specialists that's OK: By adopting non-solo development practices such as pair programming and working in short feedback cycles they will quickly pick up new skills from their teammates (and transfer their existing skills to their teammates too).

Test driven development

With a whole team testing approach the development team itself is performing the majority, and perhaps all of, the testing. They run their test suite(s) on a regular basis, typically several times a day, as part of their overall continuous integration (CI) strategy. People new to agile will often take a "test after" approach where they write a bit of production code and then write the tests to validate it. This is a great start, but more advanced agilists will strive to take a test-driven approach.

Test-driven development (TDD) is an agile development technique practice which combines refactoring and test-first development (TFD). Refactoring is a technique where you make a small change to your existing source code or source data schema to improve its design without changing its semantics. Examples of refactorings include moving an operation up the inheritance hierarchy and renaming an operation in application source code; aligning fields and applying a consistent font on user interfaces; and renaming a column or splitting a table in a relational database. When you first decide to work on a task you look at the existing source and ask if it is the best design possible to allow you to add the new functionality that you're working on. If it is, then proceed with TFD. If not, then invest the time to fix just the portions of the code and data so that it is the best design possible so as to improve the quality and thereby reduce your technical debt over time.

With TFD you write a single test and then you write just enough software to fulfill that test. The first step is to quickly add a test, basically just enough code to fail. Next you run your tests, often the complete test suite although for sake of speed you may decide to run only a subset, to ensure that the new test does in fact fail. You then update your functional code to make it pass the new tests. The fourth step is to run your tests again. If they fail you need to update your functional code and retest. Once the tests pass the next step is to start over.

There are two levels of TDD: acceptance TDD (ATDD) and developer TDD. With ATDD you perform TDD at the requirements level by writing a single customer test, the equivalent of a function test or acceptance test in the traditional world. Acceptance TDD is often called behavior-driven development (BDD) or story test-driven development, where you first automate a failing story test, then driving the design via TDD until the story test passes (a story test is a customer acceptance test). Developer TDD is performed at the design level with developer tests, sometimes called xUnit tests.

With ATDD you are not required to also take a developer TDD approach to implementing the production code although the vast majority of teams doing ATDD also do developer TDD. As you see in Figure 2, when you combine ATDD and developer TDD the creation of a single acceptance test in turn requires you to iterate several times

through the write a test, write production code, get it working cycle at the developer TDD level. Clearly to make TDD work you need to have one or more testing frame-works available to you – without such tools TDD is virtually impossible. The greatest challenge with adopting ATDD is lack of skills amongst existing requirements practitioners, yet another reason to promote generalizing specialists within your organization. Similarly, the greatest challenge with adopting developer TDD is lack of skills amongst existing developers.

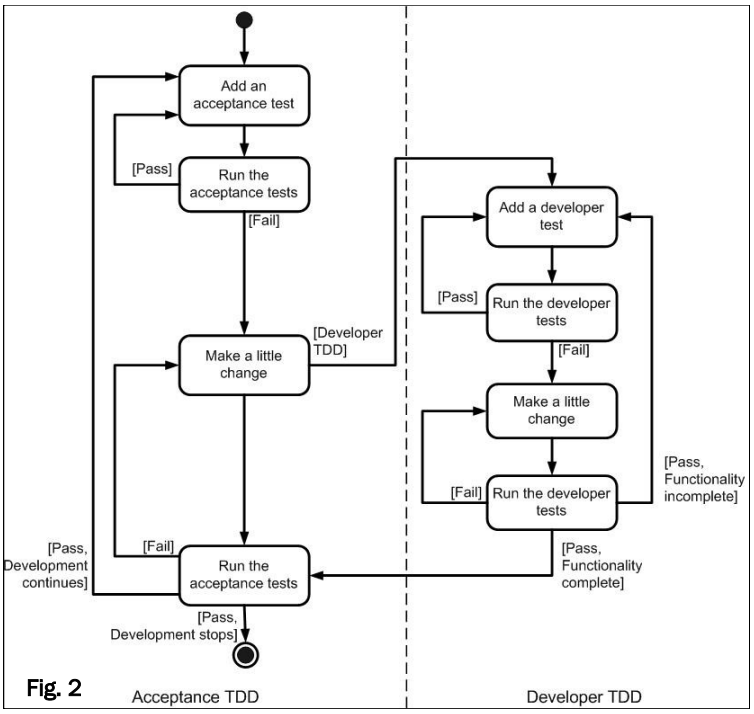


Fig. 2

Figure 2. ATDD and TDD together. (See Above)

With a TDD approach your tests effectively become detailed specifications which are created on a just-in-time (JIT) basis. Like it or not most programmers don't read the written documentation for a solution, instead they prefer to work with the code. And there's nothing wrong with this. When trying to understand a class or operation most programmers will first look for sample code that already invokes it. Well-written unit/ developers tests do exactly this – they provide a working specification of your functional code – and as a result unit tests effectively become a significant portion of your technical documentation.

Similarly, acceptance tests can form an important part of your requirements documentation. This makes a lot of sense when you stop and think about it. Your acceptance tests define exactly what your stakeholders expect of your solution, therefore they specify your critical requirements. The implication is that acceptance TDD is a JIT detailed requirements specification technique and developer TDD is a JIT detailed design specification technique. Writing executable specifications[6] is one of the best practices of Agile Modeling (http:www.agilemodeling.com/).

Although many agilists talk about TDD the reality is that there seems to be far more doing "test after" development where they write some code and then write one or more tests to validate. TDD requires significant discipline, in fact it requires a level of discipline found in few traditional coders, particularly coders which follow solo approaches to development instead of non-solo approaches such as pair programming. Without a pair keeping you honest it's easy to fall back into the habit of writing production code before writing test code. If you write the tests very soon after you write the production code, in other words "test immediately after", it's pretty much as good as TDD, the problem occurs when you write the tests days or weeks later if at all.

The popularity of code coverage tools such as Clover and Jester amongst agile programmers is a clear sign that many of them really are taking a "test after" approach. These tools warn you when you've written code that doesn't have coverage tests, prodding you to write the tests that you would hopefully have written first via TDD.

Parallel independent testing

The whole team approach to development where agile teams test to the best of their ability is a great start but it isn't sufficient in some situations. In these situations, described below, you need to consider instituting a parallel independent test team which performs some of the more difficult (or perhaps advanced is a better term) forms of testing. As you can see in Figure 3 the basic idea is that on a regular basis the development team makes their working build available to the independent test team, or perhaps they automatically deploy it via their continuous delivery tools, so that they can test it. The goal of this testing effort is not to redo the confirmatory testing which is already being done by the development team, but instead to identify the defects which have fallen through the cracks. The implication is that this independent test team does not need a detailed requirements speculation, although they may need architecture diagrams, a scope overview, and a list of changes since the last time the development team sent them a build. Instead of testing against the specification the independent testing effort will instead focus on production-level system integration testing, investigative testing, and usability testing to name a few things.

Figure 3. Parallel independent testing throughout the lifecycle.

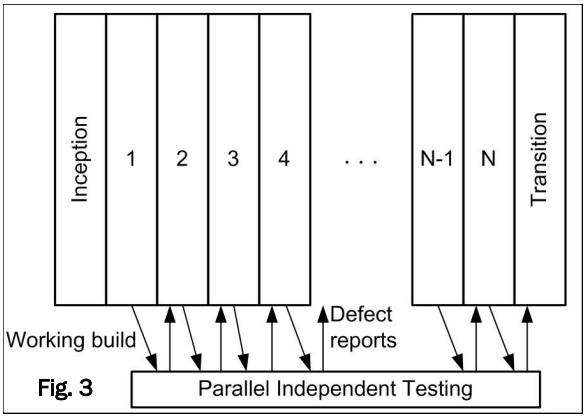


Fig. 3

It is important to recognize that the development team is still doing the majority of the testing when an independent test team exists. It is just that the independent test team is doing forms of testing that either the development doesn't (yet) have the skills to perform or is too expensive for them to perform.

You can also see in Figure 3 that the independent test team reports defects back to the development. These defects are treated as type of requirement by the development team in that they're prioritized, estimated, and put on the work item stack.

There are several reasons why you should consider parallel independent testing:

Investigative testing: Confirmatory testing approaches, such as TDD, validate that you've implemented the requirements as they've been described to you. But what happens when requirements are missing? User stories, a favorite requirements elicitation technique within the agile community, are a great way to explore functional requirements but defects surrounding non-functional requirements such as security, usability, and performance have a tendency to be missed via this approach.

Lack of resources: Many development teams may not have the resources required to perform effective system integration testing, resources which from an economic point of view must be shared across multiple teams. The implication is that you will find that you need an independent test team working in parallel to the development team(s) which addresses these sorts of issues. System integration tests often require expensive environment that goes beyond what an individual project team will have.

Large or distributed teams: Large or distributed teams are often subdivided into smaller teams, and when this happens system integration testing of the overall solution can become complex enough that a separate team should consider taking it on. In short, whole team testing works well for agile in the small, but for more complex solutions and agile at scale you need to be more sophisticated.

Complex domains: When you have a very complex domain, perhaps you're working on life critical software or on financial derivative processing, whole team testing approaches can prove insufficient. Having a parallel independent testing effort can reduce these risks.

Complex technical environments: When you're working with multiple technologies, legacy solutions, or legacy data sources the testing of your solution can become very difficult.

Regulatory compliance: Some regulations require you to have an independent testing effort. My experience is that the most efficient way to do so is to have it work in parallel to the development team.

Production readiness testing: The solution that you're building must "play well" with the other solutions currently in production when your solution is released. To do this properly you must test against versions of other solutions which are currently under development, implying that you need access to updated versions on a regular basis. This is fairly straightforward in small organizations, but if your organization has dozens, if not hundreds of IT projects underway it becomes overwhelming for individual development teams to gain such access. A more efficient approach is to have an independent test team be responsible for such enterprise-level system integration testing.

Some agilists will claim that you don't need parallel independent testing, and in simple situations this is clearly true. The good news is that it's incredibly easy to determine whether or not your independent testing effort is providing value: simply compare the likely impact of the defects/ change stories being reported with the cost of doing the independent testing.

End of lifecycle testing

An important part of the release effort for many agile teams is end-of-lifecycle testing where an independent test team validates that the solution is ready to go into production. If the independent parallel testing practice has been adopted then end-of-lifecycle testing can be very short as the issues have already been substantially covered. As you see in Figure 3 the independent testing efforts stretch into the Transition phase of the DAD life cycle because the independent test team will still need to test the complete solution once it's available. There are several reasons why you still need to do end-of-lifecycle testing:

It's professional to do so: You'll minimally want to do one last run of all of your regression tests in order to be in a position to officially declare that your solution is fully tested. This clearly would occur once iteration N, the last construction iteration, finishes (or would be the last thing you do in iteration N, don't split hairs over stuff like that).

You may be legally obligated to do so: This is either because of the contract that you have with the business customer or due to regulatory compliance. Many agile teams find themselves in such situations, as the November 2009 State of the IT Union survey discovered[7].

Your stakeholders require it: Your stakeholders, particularly your operations department, will likely require

some sort of testing effort before releasing your solution into production in order to feel comfortable with the quality of your work.

There is little publicly discussed in the mainstream agile community about end-of-lifecycle testing. It seems that the assumption of many people following first generation agile methods such as Scrum is that techniques such as TDD are sufficient. This might be because much of the mainstream agile literature focuses on small, co-located agile development teams working on fairly straightforward solutions. But, when one or more scaling factors (such as large team size, geographically distributed teams, regulatory compliance, or complex domains) are applicable then you need more sophisticated testing strategies.

Got discipline?

A critical aspect of the DAD process framework is that it is goal driven. What that means is that DAD doesn't prescribe a specific technique, such as ATDD, it instead suggests several techniques to address a goal and leaves it to you to choose the right one that is best for the situation you face. To help you make this decision intelligently the DAD process framework describes the tradeoffs of each technique and suggests when it is best to apply it. This advice for the testing techniques discussed earlier is captured in Table 1. The DAD process framework itself describes a much larger number of techniques than what is overviewed in the table.

Table 1. Comparing agile testing strategies. (See Page 12)

One aspect of discipline is to recognize that you have to tailor your process to reflect the situation that you find yourself in. This includes your testing processes. Other aspects of discipline pertinent to your agile testing efforts is to purposely adopt techniques such as ATDD and TDD which shorten the feedback cycle (thereby reducing the average cost of addressing defects), streamlining your transition/release efforts through automation and performing testing earlier in the lifecycle, and of course the inherent discipline required by many agile techniques.

There are far more agile testing strategies that what I've described here. My goal with this article was to introduce you to the idea that there is a multitude of agile testing techniques, and quality techniques for that matter too, and that many of them are quite sophisticated. I also wanted to introduce you to the DAD process framework and illuminate how it can help guide your process tailoring decisions. With a bit of flexibility you will soon discover that agilists have some interesting insights into how to approach testing. ■

References

[1] Ambler, S.W. (2003). Agile Testing and Quality Strategies: Discipline Over Rhetoric. www.ambysoft.com/essays/agileTesting.html

[2] Ambler, S.W. and Lines, M. (2012). Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. IBM Press. www.ambysoft.com/books/dad.html.

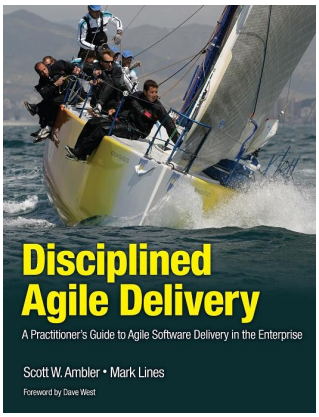
[3] Beck, K. and Andres, C. (2005). Extreme Programming Explained: Embrace Change, 2nd Edition. Addison Wesley.

[4] Crispin, L. and Gregory, J. (2009). Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley.

[5] Ambler, S.W. (2003). Generalizing Specialists: Improving Your IT Career Skills. www.agilemodeling.com/essays/generalizingSpecialists.htm

[6] Adjic, G. (2011). Specification by Example: How Successful Teams Deliver the Right Software. Manning Publications.

[7] Ambler, S.W. (2009). November 2009 DDJ State of the IT Union Survey. www.ambysoft.com/surveys/stateOfITUnion200911.html



The book: *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise* by Scott W. Ambler and Mark Lines was released in June 2012 by IBM Press.

About Scott

Scott W. Ambler is the founder of the Agile Modeling (AM), Agile Data (AD), Disciplined Agile Delivery (DAD), and Enterprise Unified Process (EUP) methodologies and creator of the Agile Scaling Model (ASM). Scott is the (co-) author of twenty books, including *Refactoring Databases*, *Agile Modeling*, *Agile Database Techniques*, *The Object Primer 3rd Edition*, *The Enterprise Unified Process*, and *Disciplined Agile Delivery*.

Table 1. Comparing agile testing

Strategy	Potential Advantages	Potential Disadvantages	Considerations
Test-driven development (TDD)	Results in better code since it needs to conform to the design of the unit tests. Gives greater confidence in the ability to change the solution knowing that defects injected with new code will be caught. Refactoring code as you go reduces the cost of maintenance and makes it easier to evolve the solution in the future.	Takes discipline to ensure tests are actually written before the code. Takes time, tests may have their own defects, or be poorly designed.	Test-driven development is an advanced practice. While some feel TDD is mandatory for effective agile development, others question its value. Refactoring is a necessary discipline to ensure longevity of the application through managing technical debt.
Acceptance test driven development (ATDD)	Business-readable regression tests can be written by stakeholders. The code is continually validated against the requirements.	The product owner or stakeholders will likely need coaching on how to write well formed tests.	Technical expertise is required to hookup the tests to the underlying solution (or to the appropriate developer tests). This requires maintenance of the test framework.
Parallel independent testing	Speeds delivery of value to customers by reducing end-of-lifecycle testing. Independent testers avoid tester bias of the business and development teams.	Requires the development team to deploy their working builds to the independent testing team on a regular basis. Requires a mechanism for the test team to easily report defects back to the development team.	The development team should still do the majority of testing. The independent testing effort should be risk-based, focusing on the issues that the development team isn't able to address effectively.
End-of-lifecycle testing	May be required to satisfy the operations team that the solution you intend to deploy has been fully testing.	Tends to be very expensive and time consuming if this is your primary source of testing. Can result in project schedule slippage if defects discovered are more than expected.	Ideally this should just be the rerunning of the regression test suite(s) created by the development team and by the independent test team (if any).
Test after development	It is easier to write tests after the code itself has been written.	Teams often find reasons to not write unit tests, such as time pressures or forgetting.	Developer regression unit testing is a good step towards doing TDD.
Code now, fix later	Quick approach to writing code that appears to work.	Leads to poor quality designs, which in turn are more difficult and expensive to evolve later.	Valid approach for prototyping code that will be discarded afterwards. Valid for production code only if you only if your stakeholders knowingly accept the consequences, perhaps because time to market is a greater consideration for them than quality.

Feature

Ten Tips for Agile Testing

This article presents ten tips for Agile testing based on our experience. However, don't expect to find the perfect test approach for your company or software project in this article. That is still something you will have to find out yourself!

By Ralph van Roosmalen

Several years ago I started as test manager on a J2EE project. The project team had switched from a waterfall approach to an Agile approach with Scrum a few months earlier. The first question the project manager asked me was, "Can you write a test plan for our first release?"

I quickly produced a project test plan that called for a test phase of a few months and a separate test team. It came complete with a capacity calculation per week for the testers, a MS-project document, and a matrix with all the quality attributes and the effort we should spent to test every attribute. What a mistake!

A few years later, we've learned a great deal about Agile testing. This article presents ten tips for Agile testing based on our experience. However, don't expect to find the perfect test approach for your company or software project in this article. That is still something you will have to find out yourself!

1. Integrate the testers in the development teams

Teams are responsible for delivering software that meets expected requirements and quality. However, if we want teams to test the software, we must give them the knowledge to do it right. Testers have that knowledge. By integrating testers into the development teams, teams obtain the skills they need to test their software. When you try this, make sure you choose the right mix: one tester for three programmers is a fair but minimal number.

2. Use risk based testing

You can never test everything with the same (extensive) depth; even in a waterfall project you have to make choices. In an Agile project all the activities are time boxed so you have to make choices about how extensively you want to test each feature. We use a risk based approach to determine which test activities we are going to carry out for a feature during the iteration. The risk level of every feature is determined by the customer and the teams. It is a very transparent process so the customer knows exactly which test activities are executed for every feature.

3. Have testers review unit tests

In our organization the developers are responsible for creating and maintaining the unit tests. Unit tests are a very important part of our test harness. Developers often have a different way of thinking; for example, they tend to test only the success scenario.

To create unit tests that are as good as possible, our testers review the unit tests for all our high-risk items. The review has two advantages. First, the unit tests are improved because testers and developers supplement each other: the developer knows where the weak places in the source are, and the tester can use his knowledge of testing to give tips to improve the unit tests. Second, the testers know exactly which test cases are executed in the unit tests and can concentrate on executing other (e.g. higher-level) test cases.

4. Create a test automation framework and appoint a toolsmith

Automated testing is very important because new features and refactoring can introduce problems that can be difficult to find. By using an automated test framework, we can maintain quality levels during the iteration. Our testers are able to create new tests easily and quickly in the framework. We have a dedicated test engineer (we call him a tool smith) that maintains and optimizes the test automation framework, reviews the new automated tests of the testers and analyzes the daily test results. Testers in the teams can spend more time creating and extending automated tests because the toolsmith supports them.

5. Display quality metrics in a public location

Almost every software project has a problem registration system, automated test results, and in some cases nightly or continuous build results. But how often do team members look at the results or count the open problems? We installed a monitor in the coffee room that displays the actual metrics of the currently open problems, the percentage of successful unit tests, the percentage of successful nightly builds, and the current state of the continuous build. By displaying the metrics in public the teams are confronted with the information. The information is no longer just a number in a system or a record with some other information in a metrics database.

6. Add a test scrum

One advantage of having a separate test team in one room is that the communication between the testers is improved. When you have a project like ours where the testers are distributed across several teams, the communication becomes more difficult. To solve this problem, we use a test scrum to align the test activities. Our test scrum is held twice a week and every team is represented by one tester. The test scrum is a scrum like the daily team scrum but focused on test activities. The test manager is the ScrumMaster of the test scrum.

7. Implement test retrospectives

Every team in our project holds a retrospective meeting at the end of the iteration. In the retrospective, the team discusses the process: what went well and what went wrong. The testers in the team learn and discover new ways to improve their tests. The sharing of knowledge with testers from the other teams helps everyone.

We have a test retrospective every iteration so the testers can exchange knowledge and experience and discuss problems they have. It is important that the retrospective is only related to test issues; you shouldn't discuss team issues (they should be discussed in the team retrospective). As with the test scrum, the test manager is the ScrumMaster of the test retrospective.

8. Plan for open problems

We try to fix all the problems that we find during the iteration in that same iteration, but sometimes we end the iteration with open problems. The best way to handle open problems is to add them to the sprint backlog for the next iteration. By explicitly planning the management of the open problems, the chance that they are "forgotten" and pile up is very small.

9. Remember: Testing is still testing

When you test in an Agile software project you can still use the "traditional" test techniques. We use exploratory testing but also apply test techniques such as boundary value analysis, equivalence partitioning, cause/effect diagram, and pair-wise testing. Which test technique we choose for a feature depends on its risk category. Exploratory testing is used in every category; but if the risk is higher we also apply more formal test techniques. The challenge for the tester is to use a formal test technique without delivering extensive test documentation.

10. Start doing it!

The last but most important tip is start doing it! Don't talk too much about how you are going to introduce testers, or which test approach is perfect for your organization. You are going to make mistakes or discover that the chosen approach doesn't fit your organization. That's a given. But, the sooner you start, the sooner you can begin learning and improving your test approach. With the retrospective meetings, you have the opportunity to adapt your test approach every month.

Conclusion

We delivered the new release of our software that we began developing in January. During the last two weeks of May, we did some more exploratory testing, solved the remaining problems, and prepared the delivery. I wrote no extensive test plan, did no capacity calculation, nor create a matrix with quality attributes. We just started testing and improved our testing every month. ■

About Ralph

Ralph van Roosmalen has been working in the software industry since 1997. He started as software developer, has worked as project manager, test manager and development manager. He is currently Vice President Research and Development at RES Software. Ralph has written several articles about Agile testing and has spoken at several conferences on the topic of Agile testing and software development in general. RES Software products enable IT professionals to manage and deliver secure, personalized and compliant desktops independent of the underlying computing infrastructure – thin clients, virtual desktops, physical desktops, or server-based computing environments.

Feature

Agile, Testing and The Need for Speed

Agile, in terms of software development, has incorrectly and for too long come to mean fast and "getting product out the door quicker." But Agile is not about speed; it is about being flexible.

By Michael Hackett, LogiGear

I always begin my discussions on Agile development by getting a definition for the word Agile. Agile, in terms of software development, has incorrectly and for too long come to mean fast and "getting product out the door quicker." But Agile is not about speed. It is about being flexible and responding to change. When I say this, I then pull out the Agile Manifesto. Nowhere is *speed* or *faster* mentioned.

The fact of the matter is many companies have switched to Agile development practices not because it's a smarter or better way of manufacturing software, not because they've made a decision to apply lean manufacturing practices or, in fact, because teams will wind up being happier if Agile is well implemented. Instead, this *Agile* thing is implemented uniquely for the purpose of delivering code/product faster to the customers.

So, let's leave aside discussion about Agile's pros and cons and teams being happier or not -- and talk only about the need for speed and its impact on *traditional* test teams.

To restate a few facts about Agile for this discussion: the Scrum is about project management, XP is about development practices, and neither deals with traditional testing tasks.

My main discussion points in this series have been what testers need to beware of in Agile development and how to implement Agile so that better software gets to your customers and testing can be successful! However, I don't want to talk about specific test methods here. That is worth its own study.

It is a fact we will be getting faster deliveries or more code. Traditional test teams must respond to this need for speed and faster releases by changing our ways and changing our perceptions of testing. Given this need for speed, how can test teams respond? The following seven points will help you find your way.

First, unit testing and automated user story acceptance testing, done by developers, will release better quality software to test teams than when there is no testing by developers.

Second, continuous integration, including the re-running of unit tests and the running of whatever kind of smoke test or automated regression tests you have available, should increase the speed of qualifying your build for testing.

Third, test teams being involved from the start with sizing and estimating user stories will increase a test team's understanding of functionality and design for more efficient test case design.

Fourth, a hardening sprint, release sprint or regression sprint - whatever you call it - for end to end system test or *beta* testing will help deal with the inability to run integration and bigger, longer scenarios versus small, narrow new functionality tests. We will discuss what happens in a hardening sprint elsewhere.

This testing happens after a *feature freeze* once the soon-to-be delivered functionality is completely integrated.

All of these things need to be in place to help test teams deal with the need for speed and the pressure of very short, quick release cycles.

Fifth, and unequivocally the most important, is that the need for the *massive* test automation in Agile projects IS the defining difference between releasing good software or releasing a time bomb to your customers. Test automation is not optional.

All those excuses like:

- Our UI changes too much
- Our developers never designed for testability
- We don't have the skill
- The tools are too expensive
- Our managers won't invest in automation
- Our managers don't understand automation
- Automation doesn't help me find bugs

… these arguments, which were valid arguments for a long time in the history of software development, are no longer valid. I am not the saying that you need to automate 100% of your testing, but I am saying, to be successful in Agile you need massive automated regression testing.

If any of these lingering arguments about the "why not to" automate persist in your company you should deal with them. Do not accept them as fact! There are many reasons why a subject matter expert/black box, perhaps non-technical skill set can automate significant areas for testing. First and foremost is your job security. Also, there are far too many good tools and far too many solutions for automating to think none will work for you. Keyword driven testing, with one automation engineer supporting up to a dozen black box testers, or developers building harnesses are just two of the many solutions available to you. Remember, it's 2012 so get with the program.

If significant automation is not happening you need to change that!

Sixth, when we talk about the need for speed, we have to talk about test artifacts. In the move to Agile, your business analysts stop writing huge requirements documents. Your developers don't have to write engineering specifications. Why should test teams have to write test plans? Why should test teams have to write test cases for that matter? In Scrum and XP, there are no test cases documented except unit tests and the automated user story acceptance tests. There are no test case managers. User scenarios, workflows and end-to-end tests that test teams once labored over do not need to be documented. Did you talk about that in Scrum training?

There is no call for them in lean manufacturing. If it doesn't go to the customer, don't do it. In the great majority of test situations, your test plans and test cases don't go to customers so don't write them!

So now let's move from theory to practice. I am not advocating testers stop documenting their work completely, but let's look at other teams. If other teams cut back on their documentation of the product they will increase their speed. We can not be expected to continue to write as much as we have and increase speed. If you want speed you have to cut back on documenting the test process. The seventh and last point- and the most optional- is in *strict* Scrum/XP implementations, there is no notion of a bug. Design and development are followed by unit test runs and re-runs, and testing happens on that functionality. When what we used to call a bug happens, the person testing tells the developer in the same bullpen or at the daily scrum and the developer either fixes it or the team decides not to. The idea of having big bug databases is not prescribed in XP or Scrum!

Now let's talk about more common implementations of Agile processes. Most teams still use bug databases, but with distributed teams and off-time-zone groups the need for tracking issues is too great to only rely on a daily standup.

It is worth reiterating that with lean manufacturing principles in mind we should cut back on test artifacts or we will remain project artifacts. Most teams that have become Agile have stopped writing test plans the way they did in the past. Many test teams have significantly cut back on documenting test cases. In the end, what will work for your team is up to the team and is also reliant on an attitude of striving for continuous improvement. I will stress again that you need to re-examine you artifact production to become faster.

To summarize, test teams must respond to the need for speed. In doing so, do not start by looking within. Look for other team members to share the test workload and automate, automate, automate! ■

Feature

Proposal For An Open Work Space

Agile stresses instant and easy communication and is built on teams working efficiently together. This necessitates an open work space environment.

By John Roets, ITX Corporation

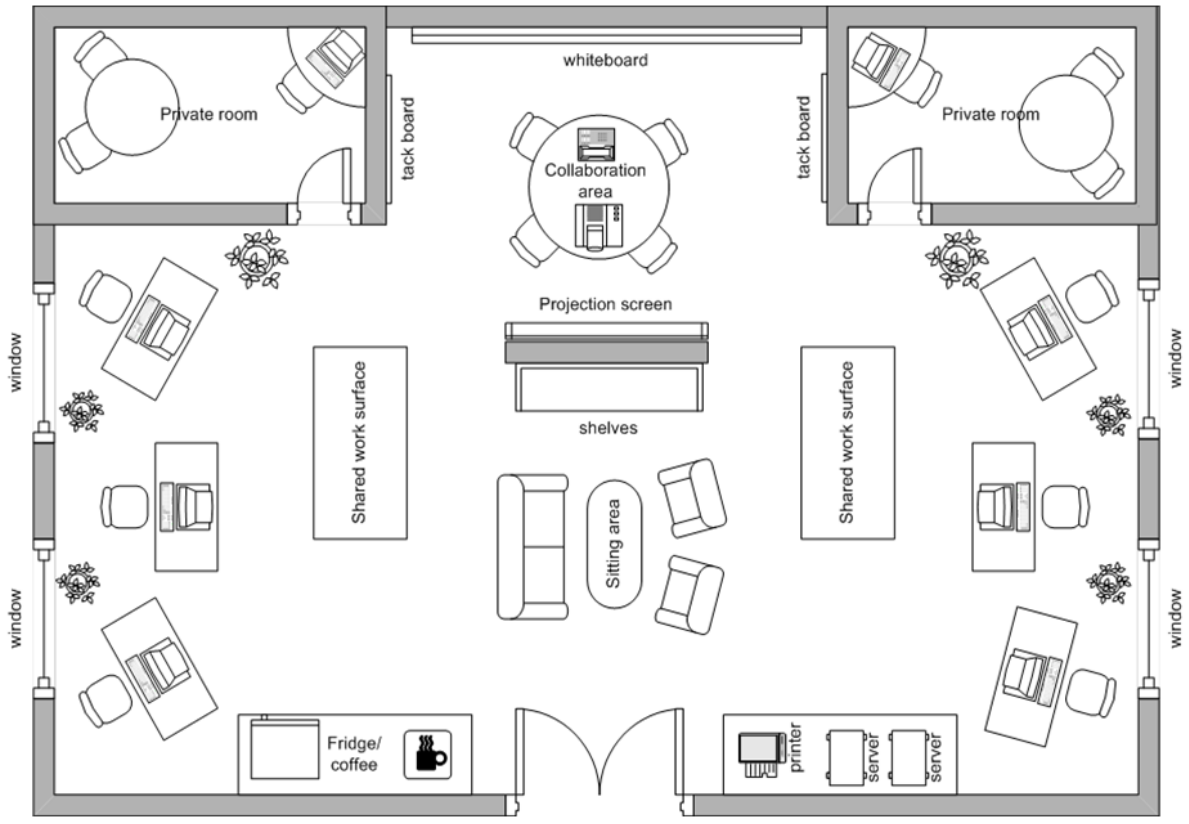
A characteristic of an effective team is a high level of collaboration, making the physical work environment an important factor. Cubicles should be eliminated in favor of an open work space in an effort to produce a higher level of collaboration, and thus provide more value to the business and a higher level of employee satisfaction.

My claims follow a logical progression. Work space design is important. Collaboration is important. Employee satisfaction is important. Cubicles are a poor choice for work space design and an inhibitor to collaboration. An appropriately designed open work space will improve collaboration and employee satisfaction, and this leads to improved results.

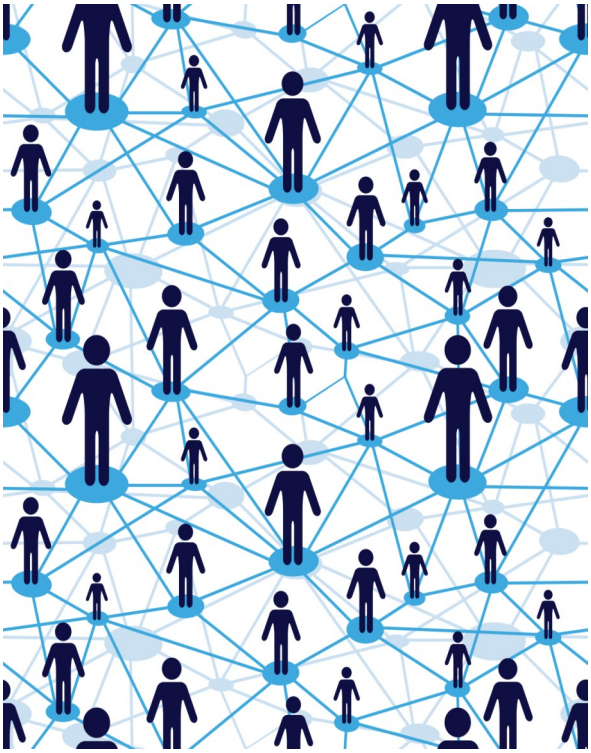
Definition of an "open work space"

An open work space is a physical work area with a defined boundary, dedicated to a team. The whole team shares a common room without cubicle borders and works on a closely related set of tasks. The work space design must facilitate ad hoc collaboration by providing things such as visual line of site among the team members and open meeting areas with plenty of whiteboards, etc. Within the defined boundary, it must also provide spaces for privacy, such as 1 or 2 private offices or meeting rooms for the occasions when privacy is needed. And it should (ideally) provide physical and visual elements that make it a desirable place to be and be reconfigurable -- people should be able to move desks around.

Here is an example of what an open work space could be:



As much as possible, the team should have control over the look and feel, and have the ability to reorganize their space if they so choose.



Communication and collaboration problems are an inherent challenge

If you look at studies that discuss project failure -- e.g. the Standish Group's annual Chaos Report -- you can categorize failure into three main buckets: poor requirements management, poor communication and collaboration, and lack of domain expertise. We can often think of the work processes of our individual organizational teams as microcosms of the processes of a larger software project. In other words, anything an organizational team does Involves elements of planning, analysis, design, execution, and validation. The same things that cause failure of the larger efforts are problematic for the day-to-day work of individual teams. Poor communication and collaboration are core problems to getting work done on any team.

Improved collaboration leads to improved results

There are essential communication aspects of successful work. Collaboration is a key factor to a highly productive and happy team. The work produced by a collaborative team is better than the sum of any work produced only by its individualistic parts.

Some popular business books have been written about the benefits of collaboration and teamwork:

- *The Wisdom of Crowds*
- *The Wisdom of Teams: Creating the High Performance Organization*
- *Group Genius: The Creative Power of Collaboration*

Academia also has studied the benefits of collaboration, in the context of collaborative learning. For example, from Collaborative Learning: Group Work and Study Teams:

> *Students learn best when they are actively involved in the process. Researchers report that, regardless of the subject matter, students working in small groups tend to learn more of what is taught and retain it longer than when the same content is presented in other instructional formats. Students who work in collaborative groups also appear more satisfied with their classes.*

Software development is (or should be) a highly collaborative activity. We see much recent evidence of the understanding of the critical importance of collaboration when we look at trends in software development processes; e.g. the Agile approaches to software development. From *Agile Software Development EcoSystems*:

> *ASDEs focus on people -- both in terms of individual skills and in terms of collaboration, conversations, and communications that enhance flexibility and innovation. As Alistair says, "Software development is a cooperative game." Most traditional "methodologies" place 80 percent of their emphasis on processes, activities, tools, roles, and documents. Agile approaches place 80 percent of their emphasis on ecosystems -- people, personalities, collaboration, conversations, and relationships.*

The momentum of the movement towards Agile processes signifies evidence of the importance of collaboration in software development. From the Manifesto for Agile Software Development:

...we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

The items that Agile values have an enormous amount to do with communication and collaboration. We want to

eliminate those things that are barriers to improving collaboration. And one of those things is cubicles.

An open work space is a better choice

An open work space will improve a team's level of collaboration, which will result in more value to the business and improved employee satisfaction. From *Open Plan and Enclosed Private Offices: Research Review and Recommendations*:

> *A seminal three-year research project conducted by UCLA revealed that companies who had modified their business processes to encourage collaboration and supported new work processes by moving from private spaces to open, collaborative environments realized performance increases (speed and accuracy of work) averaging 440 percent (Majchrzak and Qianwei, 1996). Research examining human resources, procurement, finance and other functional areas (O'Neill, 2007; Majchrzak et al., 2004) confirms the notion that workspace designed to foster group work has a positive impact on business process time and cost. O'Neill (2007) found a 5.5% reduction in business process time and cost for employees who moved from traditional enclosed office space into a mix of non-assigned and assigned open plan furnishings.*
>
> *And, Vietch, et. al. (2007) studied 779 open-plan office occupants from nine government and private sector office buildings in five large Canadian and US cities. They found that open-plan office occupants who were more satisfied with their environments were also more satisfied with their jobs, suggesting a role for the physical environment in organizational well-being and effectiveness.*

Research also shows cubicles are an inhibitor to collaboration. See *Brain death by dull cubicle*, or *Cubicles: The great mistake*. Physical work space design is part of the overall design of a team, and evidence shows that team design is a critical factor in team performance and employee satisfaction. From: *How Leaders Foster Self-Managing Team Effectiveness: Design Choices Versus Hands-on Coaching*:

> *Findings show that how leaders design their teams and the quality of their hands-on coaching both influence team self-management, the quality of member relationships, and member satisfaction, but* ***only leaders' design activities affect team task performance****.... The difficulties of fostering self-management teams -- particularly in organizations with histories of individualistic, manager-directed work –* ***have been...attributed to deficits in the...ability of managers to create the conditions that foster self-management****...*

Also, from *What Makes Teams Work: Group Effectiveness Research from the Shop Floor to the Executive Suite*: "Team effectiveness [is] a function of task, group, and organization design factors, **environmental factors**, internal processes, external processes, and group psychosocial traits."

Keith Sawyer is an author and university professor and is considered one of the country's leading scientific experts on creativity. He is the author of *Group Genius: The Creative Power of Collaboration*, which was the 2007 winner of Best Business Book on Innovation. He is a professor of education and psychology at Washington University in St. Louis. He says:

> *What kinds of offices foster creativity and collaboration? They are offices that support flexible work arrangements and frequent spontaneous reconfigurations, of people, furniture, walls, and cubicles. In innovative organizations, you find a blend of solo work, work in pairs, and collaborative teams. But most of today's offices are designed to support only one kind of work: solitary work, alone in an office (or a cubicle). In innovative organizations, people are always moving around, bumping unexpectedly into others, and stopping for a few minutes to chat.*
>
> *...for a group's genius to be fully realized, several things have to happen. First, the members of the group have to share a common body of knowledge -- and on top of that, they each have to know some uniquely different information. Second, they have to trust in each other. But trust doesn't mean everyone always agrees; it allows them to challenge other's ideas, and to propose crazy ideas of their own. Third,* ***they need to interact with each other, in a special kind of open, improvisational conversation*** *[my emphasis] -- where something unexpected can emerge. The*

best genius groups are like improvisational jazz ensembles. The outcome is unpredictable, and it depends on a complex sequence of small actions and interactions.

A framework for an open work space

The open work space and the team's work processes need to exist in such a way as to minimize undesirable effects, such as *unwanted* or *irrelevant* interruption. There needs to be a balance between the need for collaboration and the need for concentration. Taking a page from Agile development processes such as Scrum, to minimize unwanted interruption we need to set up a framework for the open work space that includes the following:

- The team's physical work space should have a physical boundary, such as walls or an entire room, to shield the team from external distraction.
- The team's work space should include one or two shared private offices, which can be used by team members when needed, for individualized focused work.

Within this framework, any negative effect of distraction or interruption is mitigated. By setting up the open work space within the context of an appropriate framework, an environment is established which facilitates a team's journey to become highly collaborative and more effective. The article "Scrum and Group Dynamics" helps explain some of the theory and principles for how a team becomes successful within such a framework.

An open work space may arguably be more costly than a typical cubicle environment, depending on how much space you give to each person and how much collaboration space you give to the team. In an open work space, an individual's personal work space is actually less costly than it would be in a cubicle space, because open work spaces do not require walls. A person just requires a desk, chair, computer, and some storage (desk shelf, filing cabinet).

People who have not worked in an effective open work space are typically going to be hesitant. We need to be cognizant of these facts and work to convince people to try it, and if possible, provide the ability to opt-out. Employee satisfaction is impacted by how an open work space is designed. The work space needs to be set up in such a way that people do not perceive the space as crowded.

Metrics

I claimed at the beginning of this article that an open work space would result in the team providing more "value" to the business. I purposely didn't frame the goal in terms of productivity -- e.g. "the team will become more productive". Attempting to measure productivity is problematic, especially in the context of software development. I believe it is better to frame the goal in the more abstract and qualitative concept of "value".

It is important to note that saying "increase the value" is the same as saying "improve the quality". We could restate the claim of benefit of an open work space as "improving the quality of the team's work", and it would be saying the same thing as "increasing the value of the team". *Quality* and *value* become synonymous according to Jerry Weinberg's definition: "Quality is value to some person."

We *could* think of increasing value or quality in many ways:

- Increase the volume of work with the same number of people
- Get the same amount done with less people
- Get work done quicker
- Become more effective (better tests, more coverage, reduced test errors)
- Increase the variety of work that the team can handle
- Find more bugs

- Let less bugs escape to the customer
- Get the team more knowledgeable, and thus more valuable

But we need to avoid the pitfalls of trying to measure these things. Knowing that *quality* is *value to some person*, and is qualitative, and that we want to keep things simple, it might be best to use survey mechanisms to measure a team's value. Identify the groups for which the quality of the team's work is important – e.g. business/product owners, developers, management, and the team members themselves -- and ask them to rate the quality of the team's work or their perspective of the value that the team provides (on a 1-10 scale). E.g.:

- How would you rate the value of the team?
  1 2 3 4 5 6 7 8 9 10
- Please comment on your rating

I also claimed that an open work space will improve employee satisfaction. We can also measure this with a survey question. E.g.:

- How would you rate your level of satisfaction with your work space?
  1 2 3 4 5 6 7 8 9 10
- Please comment on your rating

Summary

An open work space is just one part of what should be a bigger effort to maximize collaboration and create highly effective teams. For teams that are stuck in the traditional command-and-control management environment, moving to an open work environment can be a great catalyst to positive change, because it is so visible and different. The following comment, which comes from *Creative Class*, is typical:

> *[Our] space is almost completely open plan. We designed it with a library lounge filled with books and sofas and tables; a big room for seminars that seats 15-60; a conference room that seats 12-16; a bullpen for group work; and a cafe for socializing. We included 3 private offices; and were told that was too few. Let me tell you. Our big room has turned into an incredible project space. The offices are doubled up in. Our library/ lounge has become a seminar room, we had a class of 20 or so there today. If we had to do it again, we'd probably give up all 3 offices. I spend precious little time in mine. We'd take as much reconfigurable group space as we could get out hands on. Yet we are by far the exception in an academic environment, which is built around private offices for professors, classrooms, and cube farms for research assistants. I may well be the first professor in my unit to give up a private office.*

An open work space is the best choice. ■

About John

John Roets has been providing software engineering leadership, specifically to application development for the web, since 1998. John is currently a Senior Architect at ITX Corporation in Rochester, NY; providing solutions to a multitude of clients, focusing his attention (at the moment) on mobile applications. He is just as likely to engage you in discussion about design patterns, databases, and the latest web technologies as he is about business strategy, software development methodologies and organizational structure. John can be contacted through: http://jroets.blogspot.com/

Feature

Agile Testing: Key Points for Unlearning

When quality assurance teams and management who have adopted Agile practices first put the ideas to work, they face a significant impediment in un-learning the traditional mind-set and practices that experience in traditional practices has instilled in them.

By Madhu Venantius Laulin Expedith

"He who knows to unlearn, learns best." — Anonymous

The following are some of the key aspects that need to be unlearned before attempting to deploy Agile practices from a QA perspective:

- The testing team needs to be independent and independently empowered in order to be effective.
- Without a separate test strategy and test plan, it's tough to manage testing.
- The V-model for verification and validation cannot be applied in an Agile sprint.
- Independent testing teams don't do white-box testing.
- The value of testing is realized only when defects are logged.
- Automation is optional and is required only when regression testing is needed.
- Testing nonfunctional aspects, such as performance of the system, is not possible in a sprint.
- Testing must follow planning, specification, execution, and completion sequentially.
- We don't have to write new test cases for detected defects.
- Poorly written code is not the testing team's focus, as long as the code addresses the required functionality.
- Test-process improvement models do not address aspects of Agile testing.

Let's look at these assertions one by one.

The testing team needs to be independent and independently empowered in order to be effective.

Traditionally, testing teams have had followed different organizational styles, from having no independent testers while developers perform the testing, to having independent testers within the development teams, to having independent testing performed by a separate division within the organization — even outsourcing independent testing. Often the testing team would like to be empowered and report directly to a senior project manager rather than to the development or the technical lead. The logic, or at least the perceived logic, is to allow the testing team to report and escalate technical defects without potential inhibitions from the technical lead. The Agile testing mind-set change that's required is that testers are an integral part of an Agile team. Their focus is to deliver a quality shippable product at the end of each sprint and to achieve the "done" state for the backlog items committed without any technical debt. The testers report to the Agile team and are accountable to the product owner or the business.

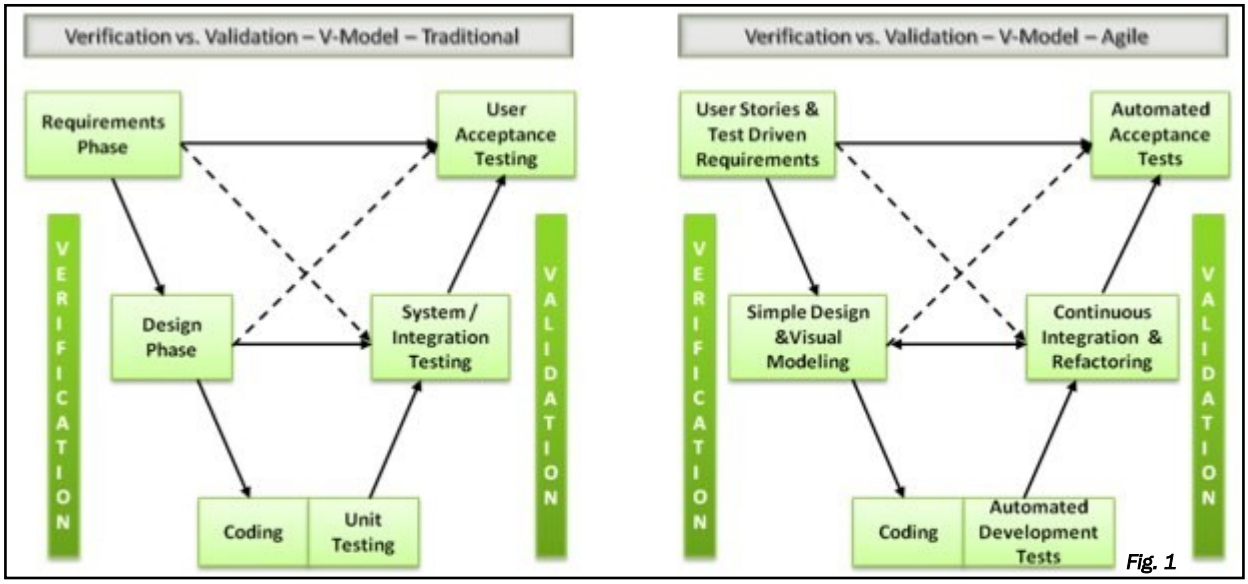


Fig. 1

Without a separate test strategy and test plan, it's tough to manage testing.

A test strategy document can typically be defined at the organizational level, the division or portfolio level, or even at the product level. Seldom must the test strategy be defined for each project, unless the project is large and the duration spans many years. The project-specific test approach is documented in the test plan for the project.

In the case of Agile projects, the test approach can be documented in the release plan, and the sprint-specific testing activities during sprint planning. A separate test plan may not be required. However, having a test strategy at a level higher than the project could be useful, especially when the organization is undergoing transformation to Agile. The test strategy can define the Agile testing practices and the techniques to be followed across the organization or division; subsequently, Agile teams can adopt one or more of these practices while defining the test approach in the release plan for the particular project.

The V-model for verification and validation cannot be applied in an Agile sprint.

Within an Agile sprint, verification and validation are addressed by adopting Agile practices. This includes verifying whether INVEST criteria for documenting requirements is followed, creating and reviewing evocative documentation and simple design, reviewing visual modeling, holding daily stand-up meetings, reviewing radiator boards, following continuous integration, refactoring, running automated development tests and automated acceptance tests, holding focused reviews, and enhancing communication by having the product owner and customer on the team.

The following figure shows an Agile V-model for verification and validation, as compared to the traditional V-model (fig. 1).

Independent testing teams don't do white-box testing.

Independent testing teams traditionally focus on black-box testing, possibly shrugging off any responsibility related to low-level testing. However, in Agile projects, testers play a significant role in automated development and acceptance tests. Agile testing is continuous and seldom staged. Agile testers need to understand the design and code-level aspects in order to effectively perform testing for a sprint. While the developers take the lead in unit testing, the Agile testing team shadows the low-level testing efforts and leads the automation aspect.

The value of testing is realized only when defects are logged.

While the value of testing lies in early detection of defects and ensuring that the shippable product is of good quality, Agile teams need to unlearn the defect numbers-game mind-set. Teams may perceive that more detected defects indicates better performance of the testing team. As a result, many cosmetic defects are logged.

The Agile testing team directly contributes to the "done" state of the product backlog item, which essentially means that a backlog item cannot be considered done unless it passes testing. Agile testing teams must make use of the radiator boards to effectively radiate the information on the status of the backlog items.

Automation is optional and is required only when regression testing is needed.

Automation is not optional; it's an essential aspect, especially when the business is trying to improve the time to market for its products. Agile teams working at peak velocity adopt such practices as continuous integration, automated development tests, and automated acceptance tests. Without automation, the team cannot achieve the desired agility.

Testing nonfunctional aspects, such as performance of the system, is not possible in a sprint.

Sometimes it may not be possible to perform testing of nonfunctional aspects, such as system performance, within a sprint. However, this can be addressed by having a separate release sprint during release planning. The release sprint can address the required nonfunctional testing and also perform a cycle of acceptance testing to ensure that the system works after any defect fixes. Rigorous integration testing may not be required if the system was continuously integrated and tested by leveraging automation.

Testing must follow planning, specification, execution, and completion sequentially.

The aspects of planning, specification, execution, and completion are highly relevant in Agile testing. However, we need to understand that Agile testing is continuous, not staged. While one backlog item may be marked "done," another item could be in its specification stages. Some teams follow the practice of updating a backlog item as done only when the test cases are automated for the backlog item.

We don't have to write new test cases for detected defects.

Traditionally, it hasn't been a practice for a test team to go back to specify a test case for a detected defect, especially for defects detected during exploratory testing. One of the key pain points for not doing so is the process of re-baselining the test case document and running around for signatures, since this is a change from the planned baseline. However, adapting to change is one of the Agile framework's foundational aspects. In Agile testing, new test cases are specified for detected defects that don't already have an associated test case, and the test case is subsequently included in the automation test cases suite.

Poorly written code is not the testing team's focus, as long as the code addresses the required functionality.

This is related to the point above ("Independent testing teams don't do white-box testing"). Independent test teams traditionally focus only on black-box testing and may be unconcerned with the quality of the code as long the code performs the required functionality. But the value add from the testing team can be significant if it can provide early feedback and also identify technical debt by focusing on the code-level aspects during verification and during validation or testing. This is one of the key Agile-testing mind-set changes required for a new Agile tester.

Test-process improvement models do not address aspects of Agile testing.

In fact, we do have the ability to measure and improve Agile testing - using standard industry models. Test-process improvement methods such as TPI NEXT advocate business -driven test process improvement in an Agile environment by prioritizing the key areas of focus. This facilitates an Agile testing mind-set by mapping Agile principles with specific, prioritized areas. TPI NEXT also provides specific "improvement suggestions" for the checkpoints in priority areas of Agile testing.

Conclusion

Although the task of performing testing is not very different *in principle* in waterfall, iterative, or Agile, the Agile mind-set and its testing practices provide effective new means to achieve the desired results. The agility lies in the Agile practices, rather than in the overarching process itself. ■

This article was originally published on the Scrum Alliance website (http://www.scrumalliance.org).

About Madhu

Madhu Expedith has 12 years of combined experience in IT process and quality consulting, project management, quality management and software development. Madhu is currently an IT manager and principal consultant in the process and quality consulting practice, Enterprise Quality Solutions (EQS). He has executed projects involving end-to-end process definition, implementation, quality management, driving and managing organizational process changes, anchored process improvement programs including training & facilitation. His expertise includes models and frameworks such as Agile, CMMI, ITIL, TPI, TPINEXT, RUP and regulations such as GCP and 21 CFR Part11.

Glossary: Agile Testing

Agile - Characterized by quickness, lightness, and ease of movement; nimble. Not necessarily characterized by fast speed.

Agile software development is a software development practice based on iterative and incremental development where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change.

Lean - Lean software development is a translation of lean manufacturing and lean IT principles and practices to the software development domain. Adapted from the Toyota production system, a pro-lean subculture is emerging from within the Agile community. When lean documentation is employed, it will result in reduced waste of time, money, and resources.

Scrum - Scrum is an agile software development model based on multiple small teams working in an intensive and interdependent manner. Scrum employs real-time decision-making processes based on actual events and information. This requires well-trained and specialized teams capable of self-management, communication and decision-making. The teams in the organization work together while constantly focusing on their common interests.

Scrumbut

1. A person engaged in only partial Scrum project management or development methodologies.
2. One who engages in either semi-Agile or quasi-waterfall development methodologies.
3. One who adopts only SOME tenants of the Scrum methodology.
4. In general, one who uses the word "but" when describing their Scrum practices.

Roles - The core roles in Scrum teams are those committed to the project in the Scrum process—they are the ones producing the product (objective of the project). These core roles are: Product Owner, Development Team, Scrum Master (Tester, Test Engineer, QA do not exist in Scrum).

Artifacts:

Product backlog- is an ordered list of "requirements" that is maintained for a product.

Sprint backlog- is the list of work the Development Team must address during the next sprint.

Increment- is the sum of all the Product Backlog Items completed during a sprint and all previous sprints.

Burn down- is a graphical representation of work left to do versus time.

eXP - XP which is also known as extreme programming is an Agile software development approach which consists of several practices and values. It is focused on software development rather than project management.

Kanban - Kanban is a method for developing software products & processes with an emphasis on just-in-time delivery while not overloading the software developers. It emphasizes that developers pull work from a queue, and the process, from definition of a task to its delivery to the customer, is displayed for participants to see.

Kanban can be divided into two parts:

1) Kanban - A visual process management system that tells what to produce, when to produce it, and how much to produce.
2) The Kanban method – an approach to incremental, evolutionary process change for organizations.

Ceremonies:

Sprint planning meeting- At this meeting, your product owner works with your team to determine which stories it will complete in the sprint.

Daily Standup– A daily team meeting held to provide a status update to the team members. The 'semi-real-time' status allows participants to know about potential challenges as well as coordinate efforts to resolve difficult and/or time-consuming issues.

Sprint Review- During this meeting the Scrum team shows what they accomplished during the sprint. Typically this takes the form of a demo of the new features.

Sprint Retrospective– A brief, dedicated period at the end of each sprint to deliberately reflect no how they are doing and to find ways to improve.

Sprint - A sprint is the basic unit of development in Scrum. Sprints last between one week and one month, and are a "timeboxed" (i.e. restricted to a specific duration) effort of a constant length.

Waterfall, or Traditional Development:

Traditional approaches, also known as “engineering” approaches, are defined at the very beginning of the software sciences. These are disciplined approaches where the stages of design and builds are predictable. Detailed stages of analysis and design precede the stage of building the software. These methodologies are well documented and thus are quite complex to apply. One of the main disadvantages is that these traditional methodologies are very bureaucratic. In practice, this highly detailed methodology leads to a high level of complexity. Often times, the work of managing the methodology is more than that of actually producing the product. Phases of the traditional approach are: requirements phase, architecture, design, code, test, and release.

Sources: Wikipedia, dictionary.com, Microsoft, Mountain Goat Software

Feature

Agile Methods and Software Testing

Agile methods were developed as a response to the issues that waterfall and V-model methodologies had with defining requirements and delivering a product that turned out to be not what the end user actually wanted and needed.

From www.agiletesting.com.au

A software tester's role in traditional software development methodology, a.k.a waterfall & the V-model can be generally summarized as:

- Finding defects in development products, such as requirements and design documents
- Proving that the software meets these requirements
- Finding where the software under test breaks (whether that is through verification of requirements or validation that it is fit for purpose)

So, what are we really talking about when we use the term "Agile" and what are the implications for a software tester? A tester's life in a waterfall or V-model based software project world is, for most traditionally trained testers, the basic process they steps they perform are similar to the following:

- You receive a requirements document which you proceed to review
- You eventually get a requirements document that is considered baseline or signed-off
- You analyze these requirements to create test conditions and test cases
- You write your test procedures
- You then wait for a piece of software to miraculously appear in your test environment
- You now start executing your tests
- Oh and now you begin re-executing some of these tests as you now start iterating through new builds which are released to fix bugs or they may even include new functionality
- You then reach the acceptable risk = enough testing point (or the fixed immovable deadline) and the software is released

Now, while all the above sounds logical and "easy" to do, the real world we live in makes it not quite so straight forward! Requirements are never complete and there are always ambiguities to deal with. The worst case is the software meets its specifications but doesn't meet the user needs.

Wouldn't it be better to build smaller parts of the system, have the business work with the developers and testers to confirm that what's being built is indeed what they want and need? So let's build the system in small increments, increasing the systems functionality in each release, and deliver a working system at the end of each increment that actually meets the end user needs. Say hello to "Agile"!

Software development is currently having a "passionate affair" with the term Agile. Unless you have been trekking in the Andes for the past 5 years, you will no doubt have heard somebody in your organization talking about Agile software development or read about some aspect of Agile on any number of software development and technology related web sites.

The trend in adoption of an Agile based methods has increased significantly and Forrester Research [2005] reported:

> *Agile software development processes are in use at 14% of North American and European*

enterprises, and another 19% of enterprises are either interested in adopting Agile or already planning to do so.

Agile software development methodologies appeared in the early 1990's and since then a variety of Agile methodologies such as XP, SCRUM, DSDM, FDD and Crystal, to name but a few, have been developed.

The creators of many of these processes came together in 2001 and created the "Agile Manifesto" which summarized their views on a better way of building software.

Testing from the beginning of the start of the project and continually testing throughout the project lifecycle, is the foundation on which Agile testing is built. Every practice, technique or method is focused on this one clear goal. So what does testing now need to know and do to work effectively within a team to deliver a system using an Agile method?

The concept of "the team being responsible for quality" i.e. "the whole team concept" and not just the testing team, is a key value of Agile methods.

Agile methods need the development team to write unit tests and/or following Test First Design (TDD) practices (don't confuse TDD as a test activity as in fact it is a mechanism to help with designing the code). The goal here is to get as much feedback on code and build quality as early as possible.

The desire for information earlier in the development phase requires monitoring the current code and build quality of the latest checked code. This requirement leads to the use of continuous build and integration practices which provide feedback every time code is checked into the code repository and the system is built, usually on a daily basis.

Brett Pettichord defined the role of testing within Agile projects as:

- Testing is the headlights of the project – where are you now? Where are you headed?
- Testing provides information to the team – allowing the team to make informed decisions
- A "bug" is anything that could bug a user – testers don't make the final call
- Testing does not assure quality – the team does (or doesn't)
- Testing is not a game of "gotcha" – find ways to set goals, rather than focusing on mistakes

The article, "Agile testing – changing the role of testers" looks at the difference between a tester on an Agile project, versus a tester on a traditional V-model project. The key challenges for a tester on an Agile project are:

- No traditional style business requirements or functional specification documents. We have small documents (story cards developed from the 4×4 inch cards) which only detail one feature. Any additional details about the feature are captured via collaborative meetings and discussions.
- You will be testing as early as practical and continuously throughout the lifecycle so expect that the code won't be complete and is probably still being written
- Your acceptance test cases are part of the requirements analysis process as you are developing them before the software is developed
- The development team has a responsibility to create automated unit tests which can be run against the code every time a build is performed
- With multiple code deliveries during the iteration, your regression testing requirements have now significantly increased and without test automation support, your ability to maintain a consistent level of regression coverage will significantly decrease

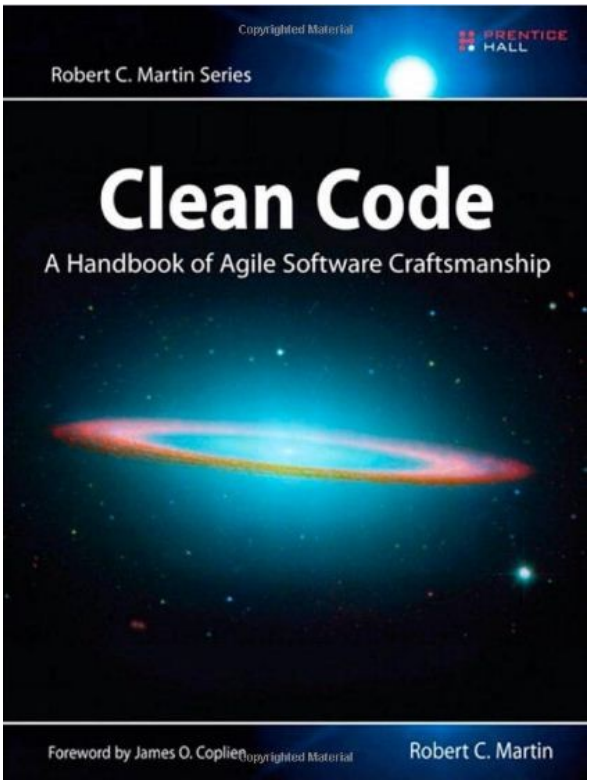
The role of a tester in an Agile project requires a wider variety of skills:

- Domain knowledge about the system under test
- The ability to understanding the technology be used
- A level of technical competency to be able to interact effective with the development team ■

Book Review

Clean Code: A Handbook of Agile Software Craftsmanship

By Gunnar Peipman



Writing code that is easy to read and easy to test is difficult to achieve. The fact that poorly written code can function often leads to coding practices that are effective but not necessarily efficient. Too often, many programmers fresh out of school write code in the manner that was effective for passing their courses, but contains a lot of spaghetti.

Experience and continual learning will eventually improve coding skills, but if you want to improve quickly I recommend a book that helped me raise my coding skill to a new level. *Clean Code: A Handbook of Agile Software Craftsmanship* by Robert C. Martin is an excellent resource on the art of writing easily readable code. The book challenges the reader to think about what's right about code, and what's wrong with it.

Readers will learn:

- How to tell the difference between good and bad code
- How to write good code and how to transform bad code into good code
- How to create good names, good functions, good objects, and good classes
- How to format code for maximum readability
- How to implement complete error handling without obscuring code logic
- How to unit test and practice test-driven development

What makes this book stand out from the others is that the information is presented in a manner that allows you to adapt and improve upon what you already know. Other books I've read introduced great methods, but required a lot of time to learn. It's fine if you have the extra time, but most of us don't. After reading the book I was able to start applying what I learned on a current project.

My experience

I selected a project that allowed me to slow down a bit so I could apply several of the practices outlined in the book. The important part for me was starting with small steps and repeating them. Pretty soon I found a new coding rhythm, enabling me to incorporate even more improvements. Putting what I learned into practice every day and not trying to do too much at once was key for me.

I started by writing unit tests the way they were outlined in the book. At the same time I also identified and was able to modify several code smells. Next, I modified the members of my classes so they had better names. I also moved some logic to subclasses and made several refactoring modifications. The first results weren't ground breaking, but they were a big step in improving my ability to write consistently clean and testable code.

I especially liked that I didn't need to spend time creating practice applications. I was able to take the project I was working on and make it better! If you follow my example, you can improve your coding ability and have fun doing it while saving yourself a lot of frustration. The important thing is don't expect to achieve super results the first time. It's a process that requires practice, but because it builds on what you already know, it doesn't require extra time. I just took small steps like the book suggested and I was able to progress quickly.

Threading guide

Besides writing clean code, the author also introduces testing problems. There is a very good section about testing threaded code, the pitfalls, and how to deal with them. Given my level of understanding of the subject, I consider this one of the best essays on threading I have read. It is not a hard-to-read technical piece, but beginners may need to research some of the definitions to understand it thoroughly.

My special thanks to Robert C. Martin

I want to give special thanks to Robert C. Martin for writing this book. Many books on coding have a steep learning curve before you start getting results, and others point you in the direction to go then leave you alone on the path not knowing what to do.

Clean Code gives you a lot more than technique. It also talks about mental tools to use to really sharpen your skills. Clean Code is top-level book. Get a copy and read it. You will never regret your decision. I promise. ■

About Gunnar

Gunnar is the CEO of Developers Team Ltd, an Estonian software development company. He has been using Microsoft platforms and technologies since primary school. The maniacal interest and enthusiasm of developing systems and playing with bleeding edge technologies is still with him. His career working with .NET and related technologies started in 2003. Gunnar specializes in ASP.NET web applications, SharePoint and SmartCard technologies. To contact Gunnar, email him at: gunnar@developers-team.com.

2010-2011 GLOBAL TESTING SURVEY RESULTS

Manager's Survey

Michael Hackett *looks at questions posed to managers in the final installment of our 2010-2011 Global Survey results.*

When I created this survey, I expected that the vast majority of respondents would be testers, QA analysts (staff who execute tests), and leads. It surprised me to learn that over a quarter of the respondents were managers - QA managers, test managers, etc. Fortunately, I had anticipated that enough managers would respond that I provided a few survey questions geared specifically toward that group. This is the final installment of our 2010-2011 Global Survey. The full survey results may be downloaded from: http://www.logigear.com/magazine/category/issue/survey/

1. What phase or aspect of development do you feel the test team at your company does not understand? (you may select multiple answers)

	Response Percent	Response Count
Schedule necessities and pressure	25%	6
That the test team is not uniquely responsible for assuring (guaranteeing) quality	58.3%	14
How to measure their test effort	41.7%	10
How to communicate their test effort	25%	6
How to choose good coverage metrics	25%	6
How to analyze and assign risk and priority	54.2%	13

Analysis: The fact that the assertion, "the test team is not uniquely responsible for assuring (guaranteeing) quality", is deemed by respondents to be the least understood by the test team is not surprising. I hear this very often in my consulting work. This reveals that many people incorrectly think the test team (often the least technically trained, the least paid, and whose work most often occurs at too late a stage to make real change) is responsible for quality. This is compounded by the fact that many people do not know the difference between testing, QC/validation, V&V and QA. There is a long way to go to get the message across that test teams are just one part of releasing a quality product.

It is also important to note *risk analysis* is the number 2 answer. Risk and risk analysis are being talked about more and more in software projects these days. What is not evident is that "talk of risk" has any meaning to real project change.

2. How do you define success in training?

	Response Percent	Response Count
Generally higher job performance	47.8%	11
Greater employee satisfaction	0%	0
Skill development, absorption of new state-of-the-practice	34.8%	8
More efficient/shorter test cycles	17.4%	4
New/more task execution for specific learned skill	0%	0

3. How do you and/or your staff normally get trained for the job?

	Response Percent	Response Count
The company provides classes with outside instructors	28%	7
The company reimburses for outside classes	20%	5
The company provides internal staff training	40%	10
The company provides for no training of testers	12%	3

4. Does your group have a test team training budget?

	Response Percent	Response Count
Yes	64%	16
No	36%	9

Analysis: It is remarkable that such a high number of companies have no test team training budget. Training, especially continuous training, is crucial to every aspect of work, from effective task execution to job satisfaction.

5. What types of training have you and/or your group taken within the last two years? (you may select multiple answers)

	Response Percent	Response Count
Programming language	20.8%	5
Test productivity/management tool	41.7%	10
Automation tool	54.2%	13
Test methodology	79.2%	19
Test or project process (Agile, RUP, IT governance, etc.)	41.7%	10

Analysis: It's encouraging to see the emphasis on methodology and not just on tool training.

6. Do you get all the training you need? If no, why not? (Duplicate responses removed)

	Response Percent	Response Count
Yes	32%	8
No	68%	17

Representative sample of responses:

- "Time. We have a small test team that is always busy!"
- "Time to train means taking time from doing work."
- "Budget"
- "Low budget"

Analysis: That close to 70% of teams do not get the training they need is surprising since there are so many methods of training available!

Also, 15 out of 17 responses to the question, "Why not?", said they were lacking in time and/or budget. Management needs to realize that the investment of time and money in training increases efficiency, and effectiveness, ultimately saving more time and money down the road.

7. Does your group share intelligence with other test groups on testing methods, techniques and the effective use of testing tools?

	Response Percent	Response Count
Yes	75%	18
No	25%	6

Analysis: One of the easiest forms of training is teams sharing methods, tools and practices across their own organization. This response ought to be 100% "Yes".

8. Does your company have a documented career growth plan?

	Response Percent	Response Count
No, it's up to the individual	41.7%	10
Yes, the company has a career growth plan for testers	58.3%	14

Analysis: It's troubling that 41% of respondents answered "No". A lack of career plan hurts morale and retention.

9. Do you plan career development for your offshore/ outsourcing team?

	Response Percent	Response Count
We provide more training than for the team at home	20%	4
We provide the same amount of training as for the team at home	25%	5
We provide less training than for the team at home	10%	2
That team's career development is not our responsibility	45%	9

Analysis: That almost half the respondents answered #4 is a clear problem for retention, productivity and effectiveness of offshore/outsourcing teams. This is a problem for the home team. ■

VIET NAM SCOPE

Beaches Abound

Whether you're looking for a 5-star resort or a UNESCO World Heritage site, Vietnam's 2025 mile coastline has something for everyone.

By Brian Letwin

Vietnam is a country with a plethora of natural beauty which manifests in many different ways. And with 2025 of coastline, the country's beaches are perhaps the pinnacle of that beauty. The characteristics of these beaches vary widely—some feature pristine natural beauty, while others have been developed and cater to millions of resort-seeking tourists each year. But no matter how developed, these beaches share two universal qualities — rich history and great seafood.

Saigon, Vietnam's largest city, lies 1 hour from the East Sea. A quick hydrofoil boat ride will bring you to the coast in no time. But if you're looking for a really special beach, you can take a 5 hour bus ride up to Mui Ne, a quickly developing beach town that has been popular with Russian tourists for decades. And with good reason—not only does Mui Ne feature great beaches but also two types of sand dunes where you can rent sleds and slide down the sandy peaks, reminiscent of childhood snow days. The seaside is flanked by outdoor seafood restaurants with dozens of fish tanks displaying the day's catch. From crabs to eel to mussels, these places have it all and at a very reasonable price.

A Few more hours up the coast lies the booming city of Nha Trang. Like the rest of Vietnam's cities, Nha Trang has benefited greatly from a large influx of foreign investment that has transformed this once sleepy beachside town into a rapidly growing commercial and tourist center. Rich in history, Nha Trang not only sports one of Vietnam's premier resorts—Vinpearl—but also some amazing ruins from the former Cham empire.

Continuing farther north, you'll find Da Nang and Hoi An, cities particularly rich in natural beauty and history. Da Nang, once a major U.S. airbase, has shed its post-war cocoon and has evolved into one of the country's most developed resort towns. The World Heritage UNESCO town of Hoi An is situated forty-five minutes from Da Nang. The road between the two towns is lined with monolithic 5 star resorts and villas that are quickly being snatched up by foreigners and wealthy Vietnamese.

Upon arriving in Hoi An, the picture changes quickly. Starting in the 15th century, the town became the country's first truly international port. The Portuguese arrived in 1535 seeking to establish a major trade center. Later centuries witnessed the arrival of the Chinese and Japanese (who believed that the "heart of all Asia" lay in the ground beneath the town) merchants which put Hoi An in the fortuitous position as the main trade conduit between Europe, China, India and Japan. Along with the economic benefits of being a trading hub, wealthy merchants built beautiful houses, canals, and other infrastructure projects such as the famous Japanese Bridge.

Eventually, the town's economic status declined after a rebellion gave exclusive trade rights to Da Nang. While losing economic power, its beautiful architecture was preserved and when walking around the town today, it feels like you're in a fairy tale or, at the very least, that you've managed to get your hands on a time machine.

Vietnam's traditionally costal development has created some truly beautiful seaside towns and cities. Whether you're after great food, pristine beaches or amazing culture, Vietnam is the place to visit. ■

LOGIGEAR MAGAZINE
JULY 2012 | VOL VI | ISSUE 3

United States
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +1 650 572 1400
Fax +1 650 572 2822

Viet Nam, Da Nang
7th floor, Dana Book Building
76-78 Bach Dang
Hai Chau District
Tel: +84 511 3655 333
Fax: +84 511 3655 336

Viet Nam, Ho Chi Minh City
1A Phan Xich Long, Ward 2
Phu Nhuan District
Tel +84 8 3995 4072
Fax +84 8 3995 4076