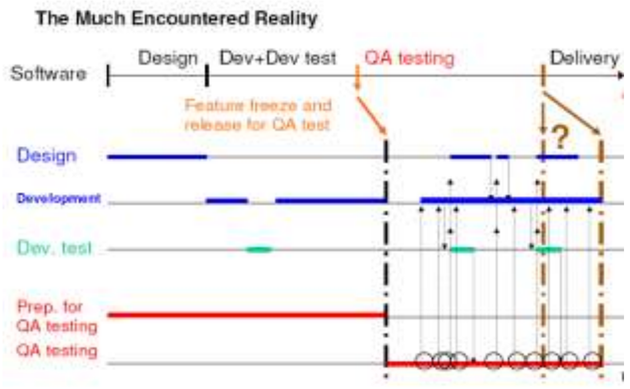
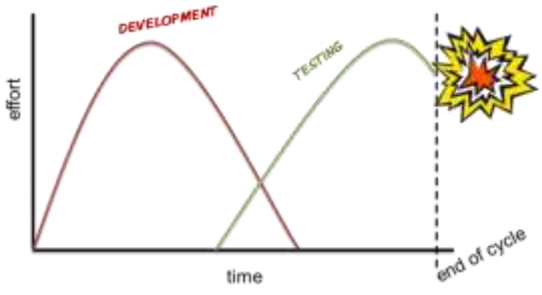
# A Software Tester's Travels from the Land of the Waterfall to the Land of Agile and Kanban

In my work, I have come across many software testing organizations/groups. Some use the waterfall method and may be in the first stages of exposure to Agile-based methods. Some are on the journey to switch between methods, and some have already been using Agile and are looking for ways to do this more effectively. In this article, I will try to describe the experience of a typical software tester when his organization decides to move to Agile.



First, let's look at a typical day of a software tester who uses the waterfall method. A version (or a project) is designed in stages. During the early stages, software testers are involved very little. They plan the stages of the tests, write test documents and prepare themselves for the day when the version will be ready for testing. During these stages, software testers may still be occupied with previous versions.

As the day for the tests approaches, tension mounts. Development teams begin to test the integration of the version in order to see if things work together. Usually, the version is not ready to be transferred for tests on the scheduled date. The integration still has problems. More time passes and the software tester is ready for the worst. He will have less time to complete the tests and, because the tests are always at the end, he already knows what the solution will be and who will be blamed since, in spite of the hours of overtime and the insane pressure, the version will be rejected. "Why do we always have a bottleneck at the software tests? They always delay our version! "

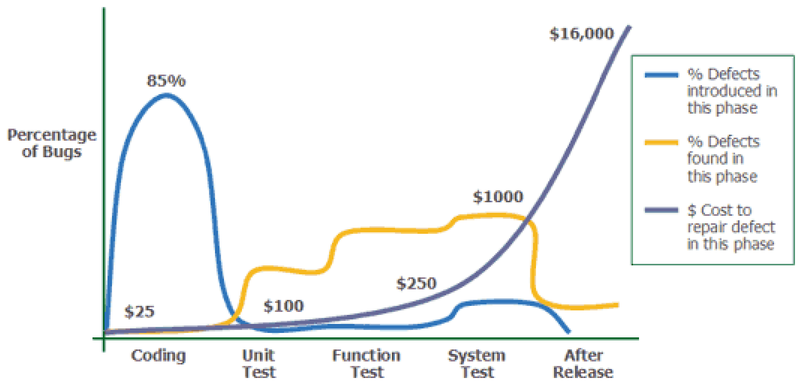


Given this dynamic, which almost always repeats itself, it is no wonder that the software tester sees himself as a "last defense", and when it's time to perform the tests, he fights valiantly to find bugs, make sure they are repaired, and finds he is frustrated when a lot of bugs that he found are not corrected in the version.

Why aren’t the bugs corrected? Because we are behind schedule, and if we fix them, we'll need to do another round of regression. We don’t have full automation so that a round of regression is very expensive. Why is there no automation? Because, if we get the version at the last minute, how can we manage to do automation? And the one who makes automation is sitting on the side, valiantly trying to complete a gap of years of writing code, and no one is trying to make his life easier.

Another fact that the software tester notices is that the longer the bug has been contained in the version, the longer it takes to find the source of the problem, and there is less chance that it will be fixed. Why?

Figure 1 Cost of Quality - Applied Software Measurement: Global Analysis of Productivity and Quality - Capers Jones



After examination of a large number of projects, it was found that most of the bugs were entered during the coding phase (surprise, right?). If we look at when they are discovered, we see that most bugs are typically found during functional testing and system testing. If much time elapses between stages - as in the waterfall method, where we finish **all** the coding of all the contents and **then** go through the stages of testing - the price of fixing a bug rises significantly. In other words, we have a problem here of Late Feedback.

This leads to two possibilities - either we "sanctify" the quality and miss the deadline of the project, or we "sanctify" the due date and find ways to compromise quality. We “waive“ more and more bugs or hide them under the carpet. Anyone who has written Release Notes and sat in a "Bug Waivers" meeting for a Revision knows what I'm talking about. I confess that I have dealt with these disgraceful activities and often I have had to explain to software testers why this compromise is necessary, and why they still need to continue to test, even though not all the bugs are fixed.

Whoever has participated long enough and often enough in these projects, starts to look for another way. The Agile world provides an interesting alternative. In Agile, instead of moving the entire contents together through the stages of work (specification, development, testing, etc.), we choose small batches and each time move them through all the work stages. For example, if I have a project with 20 features, instead of coding all of them and then testing them, I will first choose the feature with the highest priority, specify code and then test it. Only when I feel it is ready for release, will I go on to the next feature.

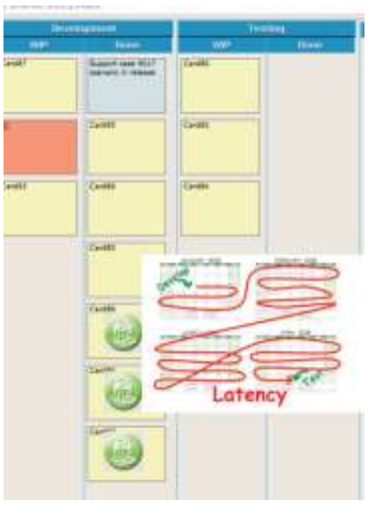
Moreover, Agile recommends working in teams in which these activities are carried out jointly. A team contains representatives of all the specialties required to take a need/idea and transform it from “demand” to “working and tested.”

How does the work of a typical software tester who works in agile look? The tester is part of the team of developers with various specialties. The team works in short iterations (sometimes called sprints). Every two - three weeks, the team will meet to plan what content/story they will focus on in the next iteration. The team decides what will be developed in the iteration, how the work will be divided among them and start the iteration. During the iteration, the team meets every day and each member shares with the rest of the team his status and what inhibits him, so that team members can help each other in their group assignment. They use Visibility tools such as Task Board, Burn down, Burn up, so that everyone will understand where they stand: are we moving in the right direction, and where to focus effort to succeed. The goal is that there will be a challenging, but doable, target, which the team can achieve without "killing themselves".

If you look at the load on the software tester, the goal is to go from insane loads at the end of a version, together with relatively “dead" periods during development of the version, to a reasonable workload throughout the version.

So far, the theory.

What is the typical experience of a software tester in an organization that works in Agile? It really depends on what the starting point is. Organizations moving to Agile from a good balance between testers and developers, with good automation coverage and a good infrastructure for developing automation, will reach the desired state relatively quickly.



But, most organizations start from a different point. Usually we see a lack of balance, with insufficient number of testers relative to developers, and minimal automation coverage. In this situation, the transition to the Agile approach of focusing on the content step by step, quickly surfaces a number of problems.

For example, we typically see developers outpacing the testers. The Agile approach says that everyone should pretty much run together – A Whole Team Approach. Gaps that are created lead to feedback problems later. Agile emphasizes the need and advantage of feedback as soon as possible - Early Feedback. Actually Agile, and, more extensively, Lean, guide us to examine any decision process that we adopt to see if it promotes earlier feedback and reduces the amount of work that we started and haven't finished - in other words "Inventory" we have in the process.

Typical software testers have seen all kinds of methods to deal with the above problem of rate differences, for example:

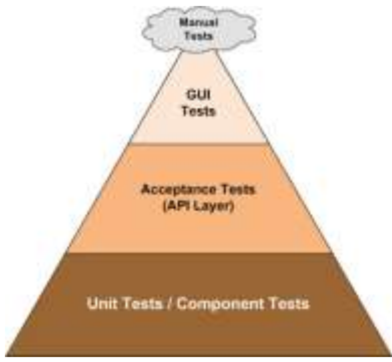
- Removing content from the version after it is developed. Developers have already developed something, and now they are waiting for it to be tested. At this stage, managers come and ask them to remove this content from the version because there is no testing time. The typical developer will be frustrated. Some of them direct the frustration toward the test group, which all the time delays them and causes them to waste time.
- Release low-quality content. It is tested late in the process because of the long queue at the entrance to tests, when it is difficult to fix all the bugs and decide to release so we can say "we made it to the finish line", but everyone knows that it is a short-lived success. The software tester feels frustrated because in the field they will now say that the version is of low quality and blame him.
- At the end of the version, when the queue at the entrance to tests has become very long the developers are given part of content to test. The developers are frustrated because it is not their role... The software testers are troubled because they don't rely so much on the developers to perform tests.

I don’t know about you, but none of the above options appeals very much to me...

The Agile approach says something else. At first, it might be more difficult and more painful, but it will pay off later. Agile says not to open a gap. Do what is necessary at all times to minimize gaps and move at the same pace. How do you make this happen?

First, let's make the testing phase as efficient as possible. If testing is identified as a bottleneck, let’s concentrate our efforts and resources on improving it, with the understanding that it will contribute to improving the bottom line for us (inspired by the Theory of Constraints by Eli Goldratt). For example, we'll seriously invest in automation to reduce the manual regression effort that takes a sizable chunk of the testing time especially as we desire more and more frequent regression testing in order to get earlier feedback. Another example - avoid a situation where the testers compete for resources in the test laboratory. Start by asking the software tester how can we save him time or what most wastes his time.

By the way, since we have already mentioned automation, if we rely on the testers alone to write automation tests for the system we will probably increase the gap between development and testing because developing automation takes more time than running manual tests. . The solution here is the "whole team approach". Automation and rate differences are a challenge to all the team and the entire group, and it is probably better that developers will help write automation to decrease the gap. If developers invest time in developing automation, most probably their own pace will be lower, and if we have automation that allows reducing regression cycle time, it is likely that the testing pace will go up. And if we combine automation with "continuous integration", meaning continually executing integration-compilation, installation and testing every day or even every check-in, we reach a situation where when the tester takes code for testing, there is a much higher probability that it is ready for testing and will not "break" right at the start and waste everyone’s time on ping pongs of fix & test.



Involving the developers in Automation is basically an example of subordination of the other team activities to help streamline testing and make it more efficient. Another example is performing more tests at the unit level (Unit Testing) by developers to reduce the number of bugs and cycles that require involvement of the testers. Beyond that, when planning work, you should think of the automation pyramid. In Agile we want to rely as much as possible on Unit Tests, a little less on Acceptance Tests, and much less on the level of the user interface (GUI), with the aim that only 5-10 % of coverage tests will be through the GUI. This assumes that the business logic and functionality of the system can be checked without going through the user interface. A transition to the agile test automation pyramid is one of the keys to success in automation, which reaches high coverage and also provides viable maintenance over time. The more we can push coverage to the base of the pyramid, the less it will cost us to develop and maintain the automation.

One of the most common concerns of organizations that need to balance rates as part of the Whole Team Approach is that balancing rates will pull the entire organization down to the lowest common denominator. This concern is valid. If, for that matter, we specify that the team or organization must not create a gap and should "march to the beat" of the bottleneck, there is a good chance that the faster specialties will “forget" how to run fast. The recommended solution is to define clearly the set of subordination tasks and expect the fastest teams to utilize their excess capacity to perform as many subordination tasks as possible (e.g., automation), so that, in the end, they do not slow down.

Another possibility is to integrate the risk management approach with testing (Risk Based Testing). You can define that content of less dangerous areas will be tested by developers or will be tested only once a bigger set of functionality (e.g. a minimum marketable feature) is ready or even just as part of the stabilization cycle (yes, late feedback… this is why we call this Risk Based Testing…), while areas at risk will be tested as soon as possible by software testers (early feedback). That could produce a bypass route that deals with the pace gap problem and prevents slowing down the entire organization.

The bottom line is that, when entering the world of Agile as a software tester, we must remember a few things. First of all, it is a journey; it does not solve all the problems in the first month and not everything will be like it is in the books. It is important to remember the principles and try to make all decisions based on these principles. How can we advance toward Early Feedback in the most dangerous areas? How can we reduce the amount of inventory in process - the amount of code developed, but not yet tested? Let us assume that there is trust and a common desire to succeed, and we agree to get help that enables us to meet our challenges and to streamline.

Software testers have a significant role in the agile environment - they help to represent the client and his expectations from the team. Even if some of the work being done by testers today is not necessarily done by them as part of an agile team - it allows them to move from reactive defense to proactive attack. What does this allow? Participation in defining functional content involved in the process earlier - e.g., by a method such as Acceptance Test Driven Development, freeing time for Exploratory Testing earlier and identification of more advanced bugs.

Software testers whose companies have adopted agile fall into two main groups: those which leverage this move to advance the level of testing in their organization and thus become really big supporters of the move, and those who are still afraid of the move and its implications for their roles. The beginning of the changeover is a deeper understanding of the method and tools. I hope that in this article I have provided a glimpse into the fascinating world of testing in an agile environment.

## Credits/References

http://blog.mountaingoatsoftware.com/the-forgotten-layer-of-the-test-automation-pyramid

http://www.slideshare.net/ehendrickson/agile-testing-u

http://www.amazon.com/Agile-Testing-Practical-Guide-Testers/dp/0321534468

## About Yuval Yeret

Yuval is a senior consultant at AgileSparks, who focuses on the subjects of agile tests, Kanban and management of large projects and programs. Yuval comes from the world of development management, with more than 17 years' experience in software development and IT. Yuval writes a blog on software development and Agile at www.yuvalyeret.com

## About AgileSparks

AgileSparks provides complete solutions in agile /Scrum/Kanban and helps companies improve their project management capabilities, with a focus on software development and integrations. AgileSparks' services include training and implementation at the client's site as well as public and private courses on Agile Testing, Scrum, Kanban, Project Management, and Product Management in the agile environment, and more. http://www.agilesparks.com