Moving to an Agile Testing Environment: What Went Right, What Went Wrong

Ray Arell

ray.arell@intel.com

Abstract

About two years ago, Ray Arell called his software staff together and declared, “Hey! We are going Agile!” Ray read an Agile project management book on a long flight to India, and, like all good reactionary development managers, he was sold! Now, two years later, their Agile/Scrum process has taken shape; however, its adoption was not without strain on development, test, and other QA practices. Join Ray as he takes you on a retrospective of what went right and, more importantly, what went wrong as they evolved to a new development/test process. He introduces the software validation strategies developed and adapted for Scrum, explains what makes up a flexible validation plan, and discusses their iterative test method. Learn how they use customer personas to help test teams understand expectations for quality in each sprint and employ exploratory testing in the Scrum development flow. If you see Agile in your development future, come discover what you’re in for, traps to avoid, and how to be successful. If you’re not ready for Agile, you’ll learn some new approaches that can be applied to traditional processes.

Biography

Ray Arell is a senior engineering manager and agilist at Intel, he has more than 20 years of hardware and software development, validation and management experience. During his tenure at Intel, he has worked on a variety of teams focused on CPU, chipsets and graphics system-level testing. Today, Ray manages an Agile software engineering team in Intel’s Digital Office Platform Division and is a leading force in the agilization of Intel. He is co-author of Change-Based Test Management: Improving the Software Validation Process (ISBN: 0971786127), and he has spoken at a number of conferences worldwide.
Introduction

Right off the get go, I have to admit that Agile project management three years ago ran contrary to almost every development process fiber in my body. After all, I participated and drove part of the creation of the waterfall product life cycle (PLC) for Intel platform software. I also used the Intel PLC within my own development teams, and I linked to other teams that followed the methods as well. So what would make a person so vested in the waterfall lifecycle rebel the way I have? Simple, the Waterfall model is flawed.

In the Waterfall model, Requirements are expected to be locked down 6+ months prior to the completion of the project. Doing so, as you can see from the figure above, creates an environment that is very resistant to change once the project is under development. This smothers innovation and makes it very hard to quickly respond to customer needs, and it creates a negatively charged work environment and high customer frustration.

Delivery cadence is also a major flaw. Requirements are never perfect no matter how well you think they are documented. Not giving customers a chance to have frequent review means you have a strong chance of missing the customer expectations due to miscommunication. That could mean five months of rework because of how far the product has been built out. This leads to cost overruns, low employee satisfaction, and customer frustration.

I spent years trying to overcome these problems, but I got to the point that I knew the model was flawed and we needed to try something new. That is when I started to dig into Agile methods and decided to use the Scrum project management methodology.

It is important to note that the scope of this paper only represents a handful of teams at Intel. Our corporation has hundreds of teams using a variety of different development methodologies, and this paper is not to imply that Intel as a whole is moving to an Agile development model. Though, like the rest
of the industry, Intel is starting to have a significant groundswell of new teams adopting Scrum and other Agile methods within the corporation.

**What Sold Me on Agile**

The primary selling point for Agile was its core focus on delivering high customer value with every release. Plus, I liked that it put together a work culture that was focused on teamwork, just enough process to get stuff done, frequent customer feedback, and, most of all, accepting the reality that requirements can never be fully determined at the start of the program.

**Picking the Project Framework**

Selecting the project framework was probably one of the most critical parts of going to the Agile development model. The Scrum model seemed to work the best. The framework was well structured and allowed modifications, it addressed building customer value and the work flow, and it would drive a culture of transparency and openness around the project. I also liked that it was not prescriptive on how the work got done. It was flexible enough to drop in any appropriate methods or tools.

**Moving from the Waterfall to Scrum**

Moving from waterfall to Scrum is not an easy task. Mostly, because it is a major paradigm shift for the people involved. A number of people thought the best way was to evolve over time to the Scrum methodology, but it is hard to see an effective path to doing this when you're dealing with a small co-located team. So we switched to a Scrum framework as soon as we had the team trained and a Scrum Master certified.

**Effect on the Team**

Within the first several product iterations, the effect on the team was clear. We had issues with old school behaviors around the role of management, people using job titles to avoid taking on tasks, problems with self-management, and the perception that the standup meetings were a form of micromanagement.

The role of the manager in the Scrum process required a fair amount of un-learning for me. I needed to shift from somebody who assigned tasks to a person coaching and removing roadblocks. I needed to be the one who oiled the gears between the development team and the customer, not the guy asking, “Did you get that done?”
People’s job titles also drove a destructive behavior in the team. For example, people with the title of architect would not pick up any task they felt was not their responsibility even if it was in their skill set, and this broke down team communications and trust. This led me to make the decision to abolish job titles, and I reinforced that people needed to embrace the concept that an Agile team member will take on any task to insure we meet our high value release. I knew making that decision ran the risk of losing a few people, and, in fact, a few people left to join a waterfall team. It was unfortunate to lose their skills, but, in retrospect, it enabled our adoption to move faster and teams to work with a greater degree of teamwork.

The perception of micromanagement was something that I did not expect to come up. I thought that people meeting every morning would promote an open work environment and a higher level of cooperation between team members. It did, mostly, but a few in the team felt that being asked daily about what they did, what they were setting out to do, and if they had any blocking issues was micromanaging them. Now the interesting thing was this was not really the issue. Digging in deeper, this had to do with the fact that what they did yesterday was very transparent to the rest of their peers. You now had daily accountability to your team. The people were mad at the process because it did not allow them to slack off.

**Specific Effect on Testing Professionals**

Intel has built up a very strong validation capability built around the Waterfall PLC. In fact, a fair number of validation teams inside of Intel have developed their own waterfall model that runs parallel to the primary development PLC. For a test professional to transition into an Agile environment is difficult. Existing test strategies, plans, test development, automation, and other processes do not take into account the potential product requirement changes. Furthermore, testing teams inside of Intel tended to be segregated from the product developers. So working integrated with the developers in the Scrum process was very awkward and stressful at the start.

**Scrum Software Validation Strategies**

The most important document that is created for any project is the test/validation strategy document. This holds true if you're using Scrum or the Waterfall lifecycle for your product development. For people who are coming from the waterfall environment, it is very important to note that you need to approach the strategy with a clean slate. This is primarily due to the fact that the Scrum project framework puts two major constraints on testing: 1) Products are delivered on a fixed cadence, and cannot be pushed out. 2) All features need to be working and meet the acceptance criteria. In other words, no features are shipped to the customer if it is not tested, repaired, and retested. Let's look at one possible strategy.

The illustration below shows the degree and type of testing deployed throughout the development lifecycle. You’ll note a more aggressive posture on requirements inspection is done early on in the project when the application is in its early stages of development. Once the software is built out, a heavier emphasis is put on regression and customer testing. Atomic level feature testing is performed with each of the sprint story deliverables, and this occurs consistently throughout the life of the development. In the early stage of product development, a usability study is conducted in order to build a pseudo-representation of the end-user. This persona, which we will explore later in this paper, enables the development team to focus on the high value needs of the customer.
Strategy: Test Scheduling

Within the Scrum framework, user stories are the mechanism for capturing product requirements. They describe the functionality with respect to the value to the customer and they contain a high-level description of the feature, evolving details, and the acceptance criteria that will be used to judge whether or not development has been completed.

One of the most profound realizations in the Scrum methodology is that putting too much detail into an ambiguous definition of a feature is a waste of time. Instead, details are deferred until the maximum amount of information is available. And, in most cases, the product development itself helps to fill in the gaps. So, as nerve-racking as it seems, test planning needs to be distributed within each development sprint.

At the beginning of each sprint, the test engineer will establish a test strategy on how to test each of the atomic features being delivered in the iteration, and will define what level of regression testing will be performed on the release candidate prior to distribution to the customer. All necessary infrastructure changes, like validation tool modifications, become a part of the development tasks that are tracked and performed within the sprint. Test case development for each of the features is also created within the sprint and ran within the sprint.

Strategy: Requirement Inspections and Regression Testing

You’ll be happy to hear that requirement inspection and regression testing don’t change that much from what you would use in the waterfall model, but it is important to understand how the key constraints, as talked about above, factor into what flavor of methodology that you employ.
Requirements inspection of the user stories takes place at the beginning of each sprint, and is primarily focused on answering one critical question, “Is the user story testable?” If it does not pass that scrutiny, then it should not be accepted into the development cycle unless there's a clear path to get that information. After all, if you can't test it, then how are you going to build it?

Regression strategies should be based on a change-based/risk-based methodology to ensure that your product quality has not regressed. Furthermore, your regression needs to be fast. If you break down a two-week development cycle, then you will see that a continuous integration and regression methodology will be needed to address the very short amount of time that you have to perform testing prior to shipping the product. The nightly regressions should be focused more on a smoke test of the product, and the targeted regression used in the days prior to shipping needs to cover a wide breadth and selected depth of testing to ensure high product quality. One key note on test automation, automate only what is necessary. Stick with automating the highest value test cases, and structure your automation infrastructure to allow for easy modifications and changes.

**Strategy: Focusing your Effort with Customer Personas**

One of the most daunting parts of establishing a good test strategy on a feature rich product is learning how the customer is going to interact with your application. In the Agile world of building high customer value with each release, you need to set a test strategy that will target the behaviors, skill, work environment, and expectations of your users. If you have only several customers, this is not too difficult to understand. But what if you have 30,000+ users of your product? Well that is where the customer persona comes into play.

A persona is a fictional person that is created to represent an amalgam of the demographics of the target users of your product. It captures all the critical characteristics of your users. Typically the persona is synthesized from a collection of interviews of real target users. Depending on the sample size of people interviewed, you may end up with more than one persona, which can be used to further focus your test effort. The theory is that if you meet the expectations of the personas, then you meet the needs of most of your users.

**Example Persona**

To help illustrate this, let me talk about the fictional character who we will call Bill Johnson.

Bill is an information technology professional with more than 15 years experience. He spends his day troubleshooting end user problems on a help line in a company of 8,000 employees, and he is also responsible for making recommendations on new technologies that help optimize cost. He is a proficient user of most office and server tools. He is a very detailed reviewer of products, and he tends to focus on compatibility of applications within his existing computer network. His pet peeves with any product are complex, hard to learn interfaces; simple errors in documentation and help files; basic features working intermittently under high network loads; and basic features that do not work the way they were described on sales brochures. Any of these conditions will prompt him to give a negative recommendation on a product. Because his recommendations have corporate-wide impact, he will vet all features first in his
lab, progress to a limited pilot, and finally do a full deployment on his corporate network. He seldom uses advanced features of the product. If he does, he will use samples provided by the vendor....

The persona continues to go on talking about the list of applications that he uses by name, what he likes and dislikes about competitive products, types of defects that are unacceptable, etc.

At this point you should be seeing the value of the persona. The fictional character, Bill, puts a face to a group of users of a product. If we focus on satisfying him, then we are on our way to meeting the expectations of a large group of people.

Applying the Persona to Your Test Strategy

The main addition to your strategy will be a new section that gives a brief overview of your user personas, the location of the detailed documents, and how the personas were used to set your approach. This level sets the customer-relevant information that will be interwoven throughout the remainder of your document.

Now let’s use our fictional character, Bill Johnson, to help us to choose the testing methods we might want to use. We know, based on Bill’s profile, that his testing is very exploratory in nature. He starts with new features, and later he will check a new product’s ability to work alongside other currently deployed products. Based on this information, it may be a good idea to add Session-Based testing to our strategy and to use his persona in the creation and priority setting of each of the testing sessions. As you can see, the benefit we have gained is that part of our testing is going to be similar to Bill’s. This is a vital step to finding defects in your lab versus his. It is one example of focusing the methods used, and you should continue to pick other methods and processes that will continue to drive a positive usage experience.

Bill also helps us to analyze gaps and limitations to our strategy, to convey the possible impact to our customer if a defect escapes, and to form a mitigation plan if we run the risk of a major test hole. We know from the profile that Bill is going to test the product in three environments. The largest is a network with 8,000 nodes. With no mitigation, we are at risk of Bill finding defects when he enters the third stage of his evaluation. Also, we know that those issues are not going to be found right away. The timing of Bill’s testing is not called out in the persona, but we can make a fair assumption that those issues may come weeks to months later after the product has been deployed adding to the risk. One other possible risk could be that you have more than one user persona, and you may not be able to test to more than Bill’s expectations. With the personas of those additional users, you are able to articulate possible market impact and risk of not being able to test to that demographic. In both cases you are making data driven tradeoffs and decisions based on real customer impact.

At this point you should have a good overview of what a customer persona is and how it fits into your test strategy. Using this method will make having to create a well formed test strategy less daunting, and it will focus your work on protecting a positive user experience. Done right, you can expect a big “thank you” from all the Bill Johnson’s in the world.

Conclusion

As talked about earlier in this paper, I want to make sure that no one gets the impression that Intel has converted to a 100% Scrum process. We still have a lot of waterfall teams working within the company, and those teams deliver products with respectable quality. But, from my perspective, I do believe that the Scrum process and other Agile methods are a far more effective way of meeting customer needs because they are centered on delivering high value to the customer with every release.

I hope that this guidance on how to structure a testing strategy for Scrum gives you some insight into what you might be in for when you transition your team to the Agile Scrum development model.
References

Pettichord, Kaner, Bach, Lessons Learned in Software Testing, on-line
Various Authors, Exploratory Testing, Wikipedia
Various Authors, Test Strategy, Wikipedia
Various Authors, Scrum (development), Wikipedia
Various Authors, Session-based testing, Wikipedia
Ray Arell, Change-Based Test Management, (ISBN: 0971786127)
James Bach, Heuristic Risk-Based Testing, STQE 11/99
James Bach, Risk and Requirements-Based Testing, Computer, June 1999
Ingrid Ottevanger, A Risk-Based Test Strategy, StarEast 2000
Bret Pettichord, The role of information in Risk Based testing, StarEast 2001
Erik Petersen, Smarter Testing with the 80:20 Rule, StarWest 2002
Anne Campbell, Using Risk Analysis in Testing, StarEast 2000
Paul Gerrard, Risk-Based E-Business Testing, System Evolutif
Gregory T Daich, Defining a Software Testing Strategy
Jim Highsmith, Agile Project Management
Ruku Tekchandani, Building a Effective Test Strategy, Intel SQE