Seeing Software Quality through an Agile Lens

Rhea Stadick
rhea.d.stadick@intel.com

Abstract

For a software quality engineer entering an Agile team for the first time, the transition can be extreme. The traditional software quality practices and activities change in many ways but, looking closely, the core principles are still there. In fact, Agile development can be a software quality engineer's dream! Agile naturally combines all factors that impact quality - from people to transparency - all in a cohesive system.

This paper focuses on the experience of one software quality engineer who went from a waterfall background and jumped head first into an Agile development environment. It details how Agile and specifically the Scrum process benefits quality. Additionally, it explains key concepts anyone starting Scrum should know and what they don't always tell you in the books. These are common "gotchas" when adopting Agile which will inevitably derail teams from achieving the quality promises associated with Agile product development if they're not understood. These topics will range from focus on technical workmanship and practices to accepting transformational change.

This paper will help quality professionals understand how Agile supports creating a quality organization and what to be aware of when embarking on an Agile journey.

Biography

Rhea Stadick is a software quality program manager for product engineering teams in the Business Client group at Intel. She is focused on developing a sustained, strong capability in engineering organizations to deliver high quality products. Over the last several years she has become an advocate for agile development and is an agile coach as well as a director of the Intel Agility Catalyst team promoting the adoption of agile culture and methodologies throughout the company. Within the Portland software community, she runs the Rose City Software Process Improvement Network (SPIN) which brings together software professionals in the area for networking and learning. Rhea graduated from Oregon State University with a Bachelor of Science Honors degree in Computer Science and holds an MBA from Willamette University.

1 Introduction

I began my software quality journey in a group that had years of experience in developing quality methodologies centered on a waterfall life cycle. This was due to our product life cycle being a staged-gate waterfall methodology. This had benefited our core hardware platforms which are expensive to change the further you move along the development timeline. However, we were developing software which is much cheaper to change. It is also higher in the stack which requires it to be at the mercy of any changes on dependent components in the system. In the complex environments that we operate in, not only the size of platforms but the complexity of the technology itself, confinement in a stage-gate life cycle that assumes you can cleanly exit one stage and not return simply did not work. I'd like to say I realized that the life cycle was a core issue in this process but when it surrounds and drives entire platforms, it's difficult to see what is really causing quality issues in the system. It's even more opaque when, as a quality engineer, you are focused on process improvement within the system, rather than looking at the system as a whole.

It wasn't until I joined a new software development team that was trying out something new (at least new to me) called "Agile" and specifically starting with an Agile project management process called Scrum. Immediately, I could see the benefits to quality in this environment: greater transparency into the quality, ability to change processes quickly, and focus on discipline within development. Compared to my experience in a waterfall life cycle, this process allowed me as a software quality engineer (SQE) to easily enforce quality while allowing rapid improvements when things weren't working. However, there were many things no one told me about what it took to actually be Agile and even how to really achieve a Scrum project methodology successfully. This paper details both the key benefits of adopting Scrum and transitioning to Agile as well as the key lessons I wish someone had told me from the very beginning. The key realization I've come upon in this seven year journey is that to achieve quality software in complex, emerging high-tech products, simply looking at life cycles or processes is not enough. It requires a full quality system to achieve, from the structure of organizations to the people and fundamentally to the culture of the development team. After having the benefit of working in two different approaches to product development, I've found Agile to be superior in enabling a system-focused view of quality that truly supports the fundamental goals of software quality professionals and ultimately of the business organization.

2 How Scrum Helps the SQE

2.1 Process Improvement

One of the biggest issues I faced in my role as a platform SQE was the very long feedback and improvement cycle. Only after we had left the requirements phase and got into development did we start to realize that there were problems in the requirements process. This could be months past the activity. Yet, due to life cycle and length of the programs, we couldn't improve the requirements process until the next platform. Months (if not over a year) on feedback loops is much too long for the fast moving high tech industry. It was also painful as an SQE to not be able to do much to fix an issue discovered in one phase that was root caused to an issue in a previous phase.

When I moved to the new team working with Scrum, the feedback loops become as long as the sprint; no more than four weeks. Our sprints happened to be two weeks at the time and eventually grew to three weeks. This single change was a complete transformation for me and the development team. We could identify issues in one sprint and change them the next and receive almost instantaneous feedback. Rather than wait entire programs for improvements, we waited a couple of weeks. Furthermore, Scrum as an empirical process builds in the required inspect/adapt activities needed for continuous improvement. Pausing to see how the program was going and what needed to change to be successful wasn't an afterthought we performed in a post-mortem following the end of a project. It was something we did every three weeks as part of a key activity in the process – the sprint retrospective. Then, we could adjust and include any changes as we planned out the next sprint. This had a profound impact on quality by identifying any quality issues or impediments very quickly and having built-in regulation to fix them before they derailed the project.

2.2 Knowing Your Quality

The second key issue I faced when working on a waterfall life cycle was the lack of visibility into the quality of the software. Our process was set up so that the Alpha milestone meant "code complete" but not feature complete. This meant that while the team thought the software was done, there hadn't been enough validation to actually verify this. This meant that we wouldn't really have a good picture of where we were at until Beta which was near the end of the program! As an SQE, I would have to track what I could and cross my fingers that all the activities not yet performed to vet out the software wouldn't find a catastrophic defect that pushed out our schedule. This is no way to live! It was painful for SQEs, but even more painful for teams who had to suffer the long nights of fixing major issues found at Beta when there was little to no time left. And by the way, this was the first time customers were really seeing the full software so new requirements (which according to our stage-gate process should have been complete) were rolling in requiring more development. I won't go into the massive integrations that also would occur at these times which led to even more excitement and discovery of new and interesting issues.

Scrum is based on the Agile value of transparency. This is designed on one level by the empirical process but really shines with the requirement that user stories are "done done" by the end of every sprint. This doesn't mean "code complete" or "done with 5 critical defects." This means for all intents and purposes we feel the user story is complete and we could ship it to customers without biting our fingernails. From a quality perspective, that means at the end of each sprint I know the exact quality of the software that had been implemented so far. I would like to throw in a giant "but…" here and I would challenge many teams who claim they're following Scrum to say they have perfectly achieved this. It takes a lot of work and time to get to this level in a 3-week iteration. However, you can make significant strides towards this that get you to a state where you are confident in your released software quality and eliminate any last minute surprise defects. More on this later in the Lessons Learned section. Aside from the "but" sandwich, this was an incredible relief from a quality perspective. I was no longer waiting anxiously on these giant integration milestones where I was certain to find many new, hairy defects and show a sea of red in the quality criteria indicators. We had visible, working software every sprint and a clear understanding of any defects in the code.

2.3 Managing Complexity

We can't kid ourselves on how complex our projects are - from the technology, to the scope, to the human elements, and even the surrounding environment. We cannot plan everything and in fact when we start a project we know very little of the full requirements and how we'll implement the solution. Therefore, we must use a life cycle and more importantly create a system that enables us to work with uncertainty, adjust to changes at any point in the program, and give us mechanisms that allow us to see how the system is doing and quickly fix any broken elements. Ralph D. Stacey demonstrates in his book *Strategic Management and Organizational Dynamics, the Challenge of Complexity*, that as the understanding of technological implementation becomes more uncertain and requirements are far from agreement, the complexity of the project significantly increases. Waterfall life cycles are good for projects with lower complexity where technology is closer to certainty and there is a higher understanding of requirements. As we move into projects with high complexity, it requires life cycles such as Scrum that are empirical rather than prescriptive and furthermore a larger systems-level approach to navigate the complexity.

At an even higher level, Agile focuses on the system level view of product development. It addresses the other complexities mentioned above such as organizational structure and team health. All quality professionals should know that product development does not just live in the space that we define by the life cycle and processes we put into place within that life cycle. In order to truly manage quality, we must have a way of addressing product development at a system level or we will inevitably be sidelined by something else in the system that we weren't addressing.

2.4 Team Ownership of Quality

The Agile values and 12 principles as described in the Agile Manifesto put a considerable amount of emphasis on high quality, working software delivered frequently. This focuses the entire team on owning quality and ensuring they are developing in a way that enables them to do this sustainably over time. On the Agile team, I was a ScrumMaster and by helping the team to follow the Agile principles and adhere to the Scrum process, I was effecting the same quality results as I was ultimately trying to achieve as an SQE. With the team owning quality, much better design, development, and testing practices emerged than I could have hoped to define on my own in the traditional SQE role. This was because the team was applying their expertise and defining ways to work better on a constant basis which resulted in much more advanced software development.

2.5 Quality is in the Eye of the Beholder

Last, but certainly not least, we get to the age old question of what is the definition of quality. Truth be told, we can define quality criteria all day long but we will never truly know unless we get it in front of a customer. Customer input and evaluation of software is just another feedback loop that is significantly shortened through the Scrum process. In waterfall we would develop all of the features and then throw them at the customer to get feedback. However, by definition of the life cycle, change was extremely hard so the likelihood of actually accepting customer feedback at that point was very low. Scrum allows and encourages you to get feedback from customers very early and frequently in the development life cycle. Because you have "done done" software in each sprint, you can now show them software that they can even evaluate. Furthermore, the process allows you to accept feedback at any point in the life cycle. At the end, you are much more assured that you have delivered a high quality product.

3 Embarking on the Journey: 5 Things You Need to Know

Based on the previous pages, you may be thinking that I've just drunk the kool-aid and believe that Scrum and Agile are a journey filled with rainbows and unicorns. Either that or *you've* drunk the kool-aid and want to jump right in and implement it tomorrow. Regardless of which side of the fence you're on, the truth is that actually applying Scrum is hard and you must implement other Agile methodologies to actually get to a state that resembles Scrum. Remember that Scrum is just a project management process. Many consultants don't talk about this but there's a lot more to do in order to get it right. So, for the skeptics and those that are about to jump in, here are the things I learned from adopting Scrum and eventually understanding what it means to be Agile.

3.1 Lesson 1: You can't get to Done (or Scrum) Overnight

This is very important to know when first starting out: actually implementing Scrum the way it is meant to be run can take months or years depending on the changes needed in the organization and becoming Agile is a never ending journey. This means two things:

1) If you really want to do it right, you need to be in it for the long haul.
2) Don't beat yourself up because you can't achieve exactly what Scrum tells you to do in the first few sprints.

Had someone told me this, I would have saved myself years of frustration as to why we couldn't implement this "simple" process. I believed the books and consultants that talked about this simple, light weight process which led me to think that it must be easy, right? WRONG. Scrum is only a light weight process because it relies on the fact that you are living in an Agile organization. That means that first and foremost you have an Agile culture that is the real glue, checks and balance system, driver, quality controller, continuous improver, and conscience of your product development. When you have a culture and a system that enables all of this, you can get away with having a super simple process like Scrum to manage your complex software development projects. I have seen scores of teams start their Agile journey with only Scrum thinking this will magically transform them into an Agile team. Scrum provides you with some visibility into what's truly wrong with your team and organization, but it doesn't tell you what the root cause is, how to fix it, or especially how to deal with the fact that a lot of organizations don't want to admit they're wrong or invest in making the big changes that will fix the core problems.

As an SQE, I implore you to start with the Agile values and repeat them over and over. Internalize them. Make everyone in the group internalize them. These will help guide you and enable the right decisions as you start to implement Agile methodologies like Scrum. Scrum and other Agile methodologies are very adaptable and can be tailored easily. It's critical that you have the Agile values and principles act as your compass to ensure you are always heading towards your ultimate goal. At the end of the day, this is all about business results and fundamentally providing value. While it may take a while to get there, focus on this will reap incredible rewards for your organization. Just make sure everyone understands it won't happen overnight. It's clear why your superiors must understand this (to prevent them from yanking the chain too quickly), but sometimes it's forgotten that not being able to achieve goals quickly can be demoralizing to teams. Help your team understand that this is a journey and celebrate each step you take towards being Agile or even getting the Scrum process right. This is your first sprint where you actually got a user story implemented, unit level tested, functionally tested, tested with the rest of the system AND it has zero critical defects? That's huge! Congratulate yourselves! Now figure out how you get more done next sprint and keep continuously improving. This is not to say that you shouldn't question why you're not actually doing Scrum. The team should understand why there are gaps and what they are going to do in order to fix them.

Eventually, when you get to a more Agile state and the team is really performing, you will see great execution and high quality every sprint. Our teams that were kept stable started executing like clockwork after four or five sprints. However, the rest of the organization was still in crisis fighting mode so these high performing teams weren't always recognized for the amazing accomplishments they were making each sprint. Angela Druckman, in her *Agile Transformation Strategy* white paper, notes that not recognizing these sprint over sprint efforts is a common failure in Agile organizations. As an SQE, you can help to highlight the great quality achievements being made each and every sprint on your teams.

3.2 Lesson 2: Champion Discipline

As mentioned above, when joining the Agile team I started as a ScrumMaster. I highly recommend SQEs joining this role if their team is moving to Agile. There are many similarities between the role, specifically the responsibility of championing discipline, adhering to core values and principles, facilitating, and influencing without direct authority.   The need to be an advocate for high quality software, discipline in development, and continuous improvement is critical in an Agile team especially when first starting out. Adopting Agile is a significant shift for the team and they need a strong rallier to push through the difficult parts. It is all too easy to fall back to what is comfortable (validating in the next sprint, individuals owning whole sections of the code on their own, piling up defects, etc.).   We also know that we can't get to Scrum (done) overnight. This is where the SQE expertise can be applied. Knowing that we weren't doing full testing (scalability, performance, stress, etc.) every sprint, there was a false sense that we were really done done with stories.   Someday, we may be able to perform full testing in a sprint – I've known teams that have accomplished this. But, in the meantime, as an SQE you can help by making the true state of the software transparent in other ways. Not only did I help the team define strict done criteria for each of their stories, I also defined sprint criteria that would help show all the testing that was done (and wasn't) in that sprint to stakeholders. That way, if the risk of not performing certain testing concerned the stakeholders, they could call it out and the team could incorporate it in the next sprint. Let's be clear, there was full functional and regression done every sprint. As an SQE, you have to help the team balance out what is cutting corners and what simply doesn't have any ROI to do every sprint. As much as possible, the amount not done each sprint should be a very low percentage of overall validation and be decreasing over time.

3.3 Lesson 3: Let Go – Resistance is Futile

Change is fundamentally difficult for humans and transformational change (such as moving from a traditional software environment to an Agile one) is even more difficult. The important part is to recognize that you will be undergoing a transformational change and learn some of the techniques for working through this change. Note that these types of changes can take years. For one of the small engineering teams I worked on, it took about four years to get to a high performing state. You'll gain a lot of benefit even early on, however. For SQEs, use this paper and start to understand how it will benefit your quality mission. I learned that everything I wanted I now had: a way to continuously improve, have a clear idea of the software quality, and a team that was carrying the quality torch with me.

During this transformation, it will be easy for the team to not follow all of the rules. For example, when I walked into the first Agile team I was on, they were validating their user stories in the following sprint from when they were developed.   Obviously, this is not Scrum. However, at the time the team had accepted it and claimed doing both development and validation in the same sprint was impossible. It wasn't impossible however trying to accomplish this using the same development and validation approach we had used in the past *was* impossible. We had to change the way we worked and how we approached product development. That meant as a team we had to build up new skills and learn new techniques to accomplish this much higher state of software quality.

3.4 Lesson 4: Meet the Team

As Agile proclaims, individuals and interactions are more highly valued than tools and processes. The people doing product development are the most important part of getting high quality software out the door to customers. When moving to Agile, I've seen many teams try to squeeze people into roles where they lack the skill or ability to deliver on the responsibilities of the role. Everyone should take these roles and responsibilities very seriously because if anyone is half delivering on a role, you will not achieve your goals. Here is a description of the typical roles in an Agile team following Scrum and the insight I've gained as to what they really need to be.

Development Team: This is a team of self-organizing, highly disciplined professionals. They support and challenge each other, they are highly focused on getting things done correctly, and they are constantly monitoring the health of their team. At the end of the day, the team shares responsibility for delivery of the product and there is no longer an excuse that any one person failed to deliver. Some examples of when you start to see this are the team swarming on stories, team members picking up tasks of others when they need help, and challenging each other to do things right from the beginning rather than banking on re-designing or fixing issues later.

Product Owner (a.k.a "Customer Whisperer"): This is one of the roles I see most abused in Scrum. Often a project or program manager is thrown into this position despite it requiring different skills. In Scrum, the team is managing their own work and executing, there no longer is a need to have someone tracking a schedule. However, there is a huge need for the Product Owner to balance out a large set of stakeholders needs, understand ROI, and see beyond what people are asking for directly. I call this the customer whisperer role because it simply isn't enough anymore to just take what customers say and write it down into a user story. The Product Owner has to look and see what the customer is ultimately trying to achieve and communicate this to the team. Why? Because as humans we aren't that great at saying what we really need and we typically phrase things in terms of what we know. Sure, I want a portable CD player that doesn't skip. However, what I ultimately want and am unable to voice is that I want a way to listen to music anywhere at any time without having to do any work. By communicating needs in this way to the team, the Product Owner allows the team of really smart engineers to innovate and create incredible solutions that delight customers (such as an MP3 player in this example).

ScrumMaster (a.k.a "Zen Master"): This is a very difficult role to become proficient in and in Scrum it is a critical role to get right. Teams may scramble around and get to a decent state without having a good ScrumMaster but they won't get to great. The ScrumMaster has to live and breathe the values and principles Agile espouses and be a leader for the team in this area. When teams are transitioning to Agile and everything is changing and many want to revert to a previous comfortable state, the ScrumMaster is the rock pushing the team forward. This person has to understand how to push when needed and how to stand back and let the team make mistakes in order to learn and ultimately become self-organizing. The most advanced teams I've seen put some of their most senior leaders in these positions because it ultimately does warrant this level of capability. Additionally, the ScrumMaster has to be able to continuously encourage the team to improve and get better. They are a zero-gravity thinker and must be able to step back and be an objective voice to the team.

Validation Engineer (a.k.a. "Developer in Disguise"): One of the biggest shifts for our team was realizing how much we could improve by having validation be competent in software development. If you're trying to complete stories, have them fully validated, and have low defects within a 2-3 week period all while keeping a sustainable pace, you no longer have time for validation to write out test cases and perform manual testing. You have to trim out activities that don't actually provide value. The validation team learned that they could write automated tests, comment in-line and auto-generate documentation of the test case while building up a regression suite that could be run on a nightly basis. This required them to understand how to write automated tests, how to work with the developers to design the software in a way that could be automatable, and even start to incorporate software development techniques like code reviews on their own tests.

Developer (a.k.a. "Code Artist): As we started to remove all the unnecessary activities that developers have to deal with in a typical product development environment, we were able to allow the team to spend more time on writing really good software. I understand this is a novel concept but in the typical development I had seen there was simply too much thrash, unrealistic schedules, and lack of discipline to have time to think about writing really great code. As we started to remove impediments from the team and create an Agile culture, developers were able to spend time on design and developing a flexible architecture that would support changes late in the project. They also wrote code in a way that was clean and understandable because anyone on the team should be able to work on any part of the code when needed. Furthermore, they focused on designing in a way that would limit technical debt and enable automated testing. The Agile team is not the disparate group of programmers slapping together code and throwing it over the wall. This was a team focused on creating value on a continuous basis and writing software that would enable them to be agile regardless of changing requirements at any stage in the life cycle.

Manager (a.k.a. "Servant Leader): With the team being self-organizing and the ScrumMaster focused on developing the team and creating a culture of continuous improvement, where does the manager fit into this picture? The role of the manager is different on an Agile team and may or may not be necessary depending on the organization. What we found is that the manager is now a career coach focused on helping people to develop. Other than that, most of their time is spent on removing impediments for the team. They clear a path to let the team execute. Do they need to know about the day to day activities of the team? No. Fortunately, this allows managers time to focus on the bigger changes need in the organization to make their team successful.

To close this discussion of how roles and responsibilities change on an Agile team, it's important to note some of the roles that are often no longer needed. We will need to say our goodbyes to the extra roles we've built over time to manage a process that doesn't fundamentally work. Goodbye to the Project Managers, the "Leads" on teams, and even to the Architect. The responsibilities of these roles are absorbed in the roles mentioned above. The team manages their own schedule and execution and the development team is now a group of leaders and great software designers.

3.5 Lesson 5: Let's get Technical

When starting Scrum, we had this false impression that by just trying to adhere to Agile and follow the Scrum process we would magically be able to achieve Scrum. Don't let anyone tell you this! We struggled for a long time with Scrum until we had people join our group who had experience in Extreme Programming (XP) and were able to bring XP techniques to the team. XP practices give the development team a way to get to clean code and increase discipline in reducing or ultimately avoiding technical debt. To repeat again, Scrum is just a project management process and you need to develop an entire system in order to deliver high quality software. That means the development team has to employ Agile technical practices in order to deliver to this goal.

Conclusion

The road to Agile can be bumpy and is far from a straight path. It requires critical thinking every step of the way, closely monitoring key metrics (delivery of working software, team health, technical debt, etc.) to understand where impediments lie and how to remove them, and the maturity to let go of old, comfortable ways of working in favor of transformative change that will lead to creation of a system that delivers the highest business value possible.   The rewards and lessons learned in this paper are a few of the key things I wish someone would have made brutally clear to my team as we started our Agile journey. There are many other complexities and activities that are part of being Agile and even achieving a Scrum process so this is by no means a complete set. For those that still have the gumption to adopt Agile, I wish you all the best and hope these lessons help! For us, it has been an extremely positive transformation to the point that no one on the team can imagine developing software in any other environment. The day you see that you can count total software defects on two hands, the whole group is focused on delivering customer value, schedule slips are a thing of the past, and the team is continually engaged and having fun you'll look back and understand the power behind creating a whole quality system.

References

Druckman, Angela. “Agile Transformation Strategy.” Collabnet, Inc., 2011.

Beck, Kent and others. “Agile Manifesto,” http://agilemanifesto.org. 2001.

Stacey, Ralph. D. *Strategic Management and Organisational Dynamics: The Challenge of Complexity*. Prentice Hall, 2011.