QA to AQ Part Six

Being Agile at Quality

"Enabling and Infusing Quality"

Joseph W. Yoder, The Refactory, Inc. – USA
Rebecca Wirfs-Brock, Wirfs-Brock Associates, Inc. – USA
Hironori Washizaki, Waseda University – Japan

In order to pay appropriate attention to system qualities, it is important that quality is enabled and infused into the prioritized work throughout the whole process. This paper presents three patterns for Enabling and Infusing Quality: System Quality Specialist, *Spread the Quality Workload*, and *Automate First. System Quality Specialists can help to define, test, and implement complex quality items that are complex and require a lot of expertise to get right. Spreading the Quality Workload throughout the development process can help keeps the team from being overburdened with quality tasks. Automating First what you can to streamline your build and testing processes enables you to eliminate tedious or mundane tasks allowing more time for team members to focus on implementation and testing of important qualities.*

Categories and Subject Descriptors

• **Software and its engineering ~ Agile software development • Social and professional topics ~ Quality assurance**
• *Software and its engineering ~ Acceptance testing* • Software and its engineering ~ Software testing and debugging

General Terms

Agile, Quality Assurance, Patterns, Testing

Additional Keywords and Phrases

Agile Quality, Quality Assurance, Software Quality, System Qualities, Patterns, Agile Software Development, Scrum, Quality Radiator, Quality Roadmap, Quality Backlog

ACM Reference Format:

Yoder, J.W., Wirfs-Brock, R., and Washizaki, H. 2016. QA to AQ Part Six: Being Agile at Quality "Enabling and Infusing Quality". HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 23 (October 2016), 12 pages.

Author's email address: joe@refactory.com, rebecca@wirfs-brock.com, washizaki@waseda.jp

Introduction

As organizations transition to agile software practices and processes, it is important that they continue to pay attention to system quality. Typically, QA groups have worked independently from the software team. However, on agile teams, QA often works more closely with the whole team engaging more deeply with developers and the Product Owner. While incrementally delivering functionality, system qualities need to be visible and included as part of the prioritized work. It is also important to not lose focus on Quality Assurance for the "ilities" of a system and to spread quality expertise among the team.

This paper is a continuation of patterns for shifting from Quality Assurance (QA) to Agile Quality (AQ). The complete set of patterns includes ways of incorporating quality assurance into the agile process as well as techniques for describing, measuring, adjusting, and validating important system qualities (Agile Quality). Previously we presented many patterns for becoming more agile at quality [YWA, YW, YWW14, YWW15, YWW16]. See appendix for a summary of these patterns. Our collection of patterns focus on actions for improving system quality and integrating quality assurance concerns, values, and roles into the whole team. This paper expands on ways for ***Being Agile at Quality*** by presenting three patterns: *System Quality Specialist*, *Spread the Quality Workload*, and *Automate First*.

QA activities are often focused on functional testing and verification. In order to specify, test, and performance, scalability or reliability of a system, specific technical expertise and skills are often required. This is where a *System Quality Specialist* who knows both technology and has a quality mindset can contribute.

While we recognize that some quality assurance activities require specific expertise and a unique quality perspective, we also find that when quality-related concerns and activities are shared among team members by *Spreading the Quality Workload,* that a commonly held set of values develop and quality-related practices become pervasive and powerful forces for change and improvement.

As a system grows, it is essential to validate that the system continues to meet system quality requirements. The more you can automate build, test, and deployment tasks the more time you have to focus on your work rather than performing tedious mundane and error prone jobs. Automation can also provide useful feedback as to whether quality goals are being met as well as mitigating risks. Putting off automation can make it difficult to automate later. Therefore, *Automate First* whenever possible.

Our patterns are intended for agile teams wanting to focus on important qualities for their systems and better integrating QA into their agile process. These Agile Quality patterns are also of interest to anyone who wants to instill a quality mindset and introduce quality practices throughout their process.

System Quality Specialist

"Asking the right questions takes as much skill as giving the right answers." — Robert Half

Quality assurance on agile projects primarily focus on validating and verifying user stories which express requirements in terms of system functionality. QA may be more comfortable and familiar with functional testing. Nonetheless, production software needs to exhibit certain system qualities such as being scalable, usable, secure, and reliable in order to satisfy its end users. Individual user stories as well as the overall system qualities must be verified to meet objectives.

How can agile teams obtain and realize the best experience and practices for specifying, testing and validating system qualities?

❖ ❖ ❖

It is not common for user stories to contain acceptance criteria for specific system qualities. There are benefits of focusing on features first, however they may be over-emphasized because it is "easy" to appreciate the values they offer to end-users. Systems are not completely finished until requirements including system qualities are adequately addressed. There are always tradeoffs, and perfection is the enemy of good enough. Good enough needs to address both business features and adequate system qualities.

System design involves making tradeoffs between implementing functionality that is good enough to meet the important business requirements while adequately addressing system qualities. When making design tradeoffs, there is a temptation to overdesign or get into too many details about system qualities. On the other hand, trying to address important system qualities after basic functionality has been implemented can result in major rework.

Not focusing on important qualities early enough can cause significant problems, delays and rework. Remedying performance or scalability deficiencies can require significant changes and modifications to the system's architecture. However, focusing too early on system qualities in the development cycle can lead to overdesign and premature optimization [Knuth].

Some system qualities require significant expertise to make sure they are specified, designed, implemented, and verified appropriately. Sometimes this expertise is not available to the agile team.

❖ ❖ ❖

Therefore, when your team is lacking specific skills, include a *System Quality Specialist* at various times (possibly full time) to assist your team with describing, validating, and testing system qualities.

A *System Quality Specialist* is a QA role distinct from a developer or architect who has deep technical skills related specific system qualities. Those roles are needed too. For example, if a product needs to be secure, it is important that security is designed and built into the system from the beginning. Security doesn't magically emerge. A team might need the additional expertise of security architects or developers as well as security specialists from QA. This role can be temporary until the team acquires the necessary skills, or the specialists could become a full-time team member if the need is ongoing.

The term specialist sometimes has a bad connotation, implying that knowledge is unnecessarily held too closely or poorly communicated to others. For example, some agile teams avoid hiring specialists. However it is wishful thinking to believe you will *only* have "t-shaped"[1] people working on a team. Not everyone necessarily is able to easily acquire the deep skills necessary to perform certain quality-related tasks. Sometimes you need specialists and the specialists is not necessarily a t-shaped person.

A *System Quality Specialist* usually comes from the Quality Assurance group if the organization has one. If not, it is possible to bring in a quality expert from another part of your organization or bring in an outside expert to assist with this specialized role. Some quality specialists may have different areas of expertise such as usability, performance or security.

QA may provide specialists to agile teams to help with outlining and creating specific test strategies for validating and monitoring system qualities. The main requirement for a *System Quality Specialist* is that they understand system qualities and know how to describe, test and verify them.

The *System Quality Specialist* can raise awareness of system qualities to the team. They can work individually or collaboratively on system quality-related tasks. For example, they can help *Find Essential Qualities* or write or review *Quality Scenarios* and *Quality Stories*. They can ensure that quality-related acceptance criteria are adequately specified in *Fold-out Qualities.* And they can create useful *Quality Radiators* and *Quality Dashboards*.

There are many tasks that a *System Quality Specialist* can contribute to or lead. It is important that they don't become overloaded or are the only source of quality-related expertise. A *System Quality Specialist* can help *Spread the Quality Workload* through the *Pairing with a Quality Advocate* and *Shadowing the Quality Expert*.

[1] T-shaped people have skills and knowledge that are both deep and broad [Brown].

Spread the Quality Workload

"Individual commitment to a group effort - that is what makes a team work, a company work, a society work, a civilization work."— Vince Lombardi

Agile teams spend most of their time specifying, implementing, and verifying that functionality is complete. It is also necessary to implement and validate system qualities before a system is ready to release. There are many quality-related tasks that need to be performed. If they aren't addressed in a timely fashion QA can become the bottleneck for getting things done.

How can teams balance quality efforts with feature delivery to ensure that all tasks are addressed at responsible moments?

❖ ❖ ❖

QA may be reluctant to verify system qualities until all functionality is completed, believing that testing won't be useful on partially implemented functionality. Not verifying important qualities early enough can cause significant problems, delays and rework. Remedying performance or scalability deficiencies can require significant changes and modifications to the system's architecture. Focusing too early on system qualities can lead to overdesign or premature optimization.

It requires technical skills and effort to specify, configure an environment for testing, and verify system qualities. QA may lack experience in system quality specification and verification tasks. This might lead teams to perform a lot repetitive and inefficient manual tasks. Although there are benefits to getting quick feedback, as the project grows, having to perform a growing number of manual tasks to verify system qualities will slow the team down.

It can be difficult to overcome cultural barriers. For example, many developers want to focus on writing code and not want to take on QA tasks or the role of tester. Or they might see verifying system qualities as just another testing task.

QA is often understaffed, overworked, and underappreciated. This can lead to poor morale and not enough QA resources or experience to address system qualities as often as they would like. This leads to QA being in a reactive rather than a proactive mode, identifying fires rather than preventing them.

Product Owners often focus early in a project only on functional requirements. While understanding functionality is important, this can lead to quality-related tasks getting piled on at the end.

Therefore, rebalance quality efforts by involving more than just those who are in QA to work on quality-related tasks. Spread the quality workload over time by including quality-related tasks throughout the project, not just at the end.

The goal is to take a balanced approach to tackling quality work including the definition, implementation, and validation of system qualities. Developers already have a responsibility and ownership for code quality and helping make sure it meets the core business requirements through functional testing (i.e. Unit Tests). However, a developer can also assist with validating system qualities as well. For example, a developer can work on writing a test-fixture to validate a specific system quality with guidance and verification from the QA expert. Or a developer can pair with QA to build some infrastructure for validating and monitoring critical system qualities [Sav]. Or if developers get trained on the basics of exploratory testing, they can provide fresh testing perspectives on new system functionality and help balance the load.

This all comes down to everyone working together to make the project successful, pitching in when needed, not only when being told to. All teams members including developers can help with QA tasks. It is a way to "load balance" quality efforts. Not everyone has the same expertise but they can still learn to do some quality-related tasks. As the team is growing, there are times that individuals from the team can move slightly outside of their comfort zone, which is normal for growth. For example it can become a developer's responsibility to run system quality tests insuring they all pass before checking in their code. QA is still responsible for overall system quality, however some of their responsibilities or tasks can be shared.

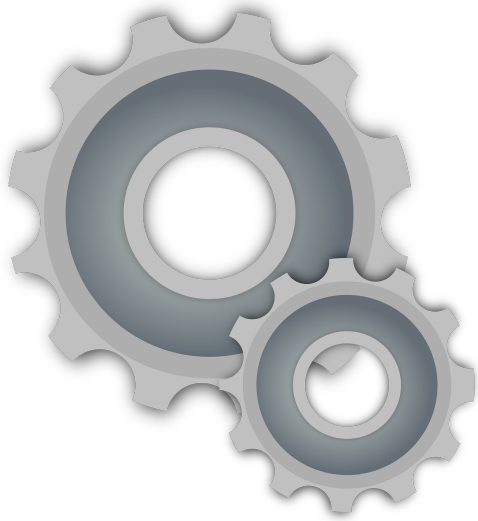
It is important to spreading the quality workload over time as well as distribute it within the team. Trying to address complex system qualities at the end of the project can cause many problems and rework. One way to make sure important items are addressed at appropriate times is to *Qualify the Roadmap* and to *Qualify the Backlog*.

Also, it is productive to write and run system quality tests as soon as there is enough implementation to be tested, even before the end of a sprint. Test results, while still preliminary, provide valuable feedback to the development team. This also helps the team to know when important qualities should be worked on and improved. QA should post feedback of what they were able to test and their results on an ongoing basis. But this isn't enough. Everyone on the team should feel comfortable raising quality issues when they find them.

Quality Checklists and *System Quality Dashboards* can help ensure quality items are not being forgotten or overlooked. Experience can be shared by *Shadowing the Quality Expert* and by *Pairing with a Quality Advocate.* When you *Spread the Quality Workload*, you definitely *Break down the Barriers* as you work more as a *Whole Team*.

Automate First

"The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency."— Bill Gates



At the start of agile projects there are many pressures to get something out to the end-user and to get initial reactions and feedback. It is also important to establish a frequent delivery cadence and tools to make that possible. In creating this environment, quality-related items need to be considered as well. As a system evolves, it is essential to regularly evaluate the system to make sure that key qualities are being met.

How can agile teams create tooling and an environment to assist with quick feedback about important qualities of the system and make their current status accessible and visible to the team?

❖ ❖ ❖

Agile teams primarily focus early in a project implementing functional requirements. There is often a priority to doing the minimal necessary to getting something working so as to get customer's reaction to the system's functionality as soon as possible. This can lead to taking shortcuts or a lot of quick and dirty manual tasks such as testing to quickly get the product out. Although there are benefits to getting fast feedback, as the project grows, a growing number of manual tasks slows the team down, making it harder to safely validate and evolve the system.

There is often a temptation to use the latest and greatest tool that has been recently hyped. However, you have limited time and resources and there is a lot of pressure to get something out as soon as possible. Automation takes time and energy to set up and sometimes the payoff is not immediately apparent. Some tasks, specifically quality testing and validation, can be hard to automate if the architecture was never designed to be testable.

Setting up environments and automating the testing of system qualities can require specialized skills and expertise. Also, you may not know what to measure about qualities until you have sufficiently defined the functionality. You want to avoid automating or putting effort into tasks that may not add value later. Being agile, you want to do things just in time.

❖ ❖ ❖

Therefore, create an environment and use tools to automate fundamental things that add value as soon as you can. Do not put off automation tasks until late in development.

Automate First means to automate whatever adds value as soon as you can. Some automations are important to do from the start. Early on the most essential things to automate are the build, integration and test environment configuration. Then automate functional tests and system quality tests. But that's only the start. There are many things you can automate such as functional tests, integration, build, deploy, quality tests, performance metrics, code smell detection, application security checks, and architectural conformance. Also if you have repetitive, tedious or error prone tasks, and if you can automate those, do so as soon as possible. As you automate tasks, they become part of the cadence of your project.

The more you automate repetitive manual tasks, the more time it frees up to you to do the work. It also allows more time to spend exploratory testing. Automation also allow you to more safely evolve the system. Automation lets you do work in smaller batches, making fewer mistakes and getting quicker feedback. With automated tests, you will know when something goes wrong with those items you are testing. You can run automated tasks more often making sure that important qualities or assumptions are still being met.

If you need to validate performance under load before you release, you might need to spin up and create a specific environment to test the system performance. This could require setup of databases and networks, etc. Doing this by hand every time can be error prone and take time. By creating a virtual environment with scripts that automate this setup, you can make performing this task much easier, especially if you have to repeat this task. As you see you are having to repeat manual tasks and that automation can help, it is time to add an automation task to your backlog.

When making a decision to automate, it is helpful to think about how long a particular automated task takes to execute and how frequently it is performed. You may automate infrequently performed tasks, especially if they are error prone or involve a lot of steps. If this task is expensive to automate and you do it very infrequently, a manual checklist script might be the better alternative. For example, if you have a manual task that you perform once a year that takes one day but can be automated in five days, your return on time spent to automate will come only after five years.

The time it takes to perform any automated task figures into your consideration of whether to install it into your normal build and deploy process or to take it off of that workflow. If you can get early, quick feedback using less time-consuming automated tasks, those automations may prove as valuable as more thorough, longer running tests or automations. For example, a security scan that runs penetration testing against a deployed app can be automated, but might be too slow to be part of an automated build process. A static code analysis that includes looking for some security defects might not be as comprehensive as the penetration scan, but if it can be done much faster, it can be pushed earlier in the process.

There are different testing cycles, especially for system qualities. Some tests will be run often (maybe hourly) such as unit tests. When you check-in code, there will be some simple integration tests that will run. But other tests, if run at that time, might slow down the check-in process too much. So these quality or regression tests might be run nightly or even less frequently.

There are a broader set of activities, beyond simply building and running tests continuously that need to be part of a continuous integration pipeline, such as deployment and IDE

integration [Duv]. One of these activities that is valuable during continuous integration is *Continuous Inspection*. *Continuous Inspection* includes ways of running automated code analysis to find common problems before integration. *Continuous Inspection* also describes many additional automated tasks that can help insure that certain qualities and architecture constraints are being met [MYGA].

Automation tools are highly dependent upon the architecture and platform that is being used. For example if you are developing with Java, you might consider SonarQube, PMD, Checkstyle, FindBugs, JaCoCo, Jenkins, Maven and Eclipse with various plugins such as JUnit and code analysis tools.

When selecting tools, it is also useful to evaluate and select one or more tools that can perform static analysis on your code base. Tool evaluation criteria should include the programming and scripting language(s) used in your software projects versus the ones supported by the tool; whether the tool provides an API for developing customized verifications; integration with your IDE; integration with your continuous integration server; and the appropriateness to your project of the built-in verifications provided by the tool. Specialists can assist with tool selection.

Automation considerations will influence your architecture style, your choice of frameworks, interface design, and many design details. For example if you decide that you will have many automated tests, you will want to design your system so these can be easily added. And while you will want to make parts of you system testable in isolation, you also need to consider how to perform meaningful integration tests.

There is a risk that focusing too much on automation could cause the team to get caught up in tooling and spend too much effort and time on automation. Another risk is that by automating too many tasks you consequently slow down your build and deploy pipeline by testing too frequently or at the wrong time in the pipeline. Knowing what automation is necessary or not, and when automated tests and tasks should be run, is important. *System Quality Specialists* can provide expertise to assist with automation. Some quality tests are hard to set up and take a lot of time to run. Sometimes you have to be clever to set these up correctly and decide when good times are to run them.

Summary

This paper is a continuation of patterns for shifting from Quality Assurance (QA) to Agile Quality (AQ). The complete set of patterns includes ways of incorporating QA into the agile process as well as agile techniques for describing, measuring, adjusting, and validating important system qualities. This paper focused on three patterns for prioritizing and making quality visible. Ultimately it is the authors' plan to write all of the patlets listed in the appendix as patterns and weave them into a 3.0 pattern language [Iba] for evolving from Quality Assurance to an Agile Quality mindset.

Acknowledgements

We thank our shepherd David Kane for his valuable comments and feedback during the PLoP 2016 shepherding process. We also thank our 2016 PLoP Writers Workshop Group, AAA, BBB, and CCC, for their valuable comments and suggestions for additional patterns to add to this collection.

References

[Bar] Barbacci M., Ellison R., Lattanze A., Stafford J., Weinstock C., Wood W., "Quality Attribute Workshops (QAWs), Third Edition," Technical Report, CMU/SEI-2003-TR-016, 2003.

[Bass] Bass L., Clements P., Kazman R., S*oftware Architecture in Practice (3rd Edition)*, Addison-Wesley, 2012.

[Brown] Brown T., *The hunt is on for the Renaissance Man of computing,* in The Independent, September 17, 1991.

[Coc] Cockburn, A., *Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams*, Addison-Wesley, 2004.

[De] Deming, W. Edwards, *Out of the Crisis*, MIT Press, 1986.x

[Duv] Duval, Paul. Continuous Integration: Patterns and Anti-Patterns. DZone, 2010. http://refcardz.dzone.com/refcardz/continuous-integration.

[Iba] Iba, T. 2011, "Pattern Language 3.0 Methodological Advances in Sharing Design Knowledge," International Conference on Collaborative Innovation Networks 2011 (COINs2011).

[IEEE730] IEEE Standard 730-2014 - IEEE Standard for Software Quality Assurance Processes, 2014.

[IEEE1012] IEEE Standard 1012-2012 - IEEE Standard for System and Software Verification and Validation, 2012.

[ISO] ISO/IEC 25010: 2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.

[Knuth] Knuth, D., "Structured Programming With Go To Statements," Computing Surveys, Vol 6, No 4, December 1974, pp. 261-301.

[Kru] Kruchten, P., Blog post - "The Missing Value of Software Architecture," http://philippe.kruchten.com/2013/12/11/
/the-missing-value-of-software-architecture, 2013.

[MYGA] Merson P., Yoder J., Guerra E., and Aguilar A., "Continuous Inspection: A Pattern for Keeping your Code Healthy and Aligned to the Architecture," 3rd Asian Conference on Patterns of Programming Languages (AsianPLoP), Tokyo, Japan, 2014.

[Sav] Savoia S., "Tearing Down the Walls: Embedding QA in a TDD/Pairing and Agile Environment," Agile 2014 Conference, Orlando, Florida, USA.

[YWA] Yoder J., Wirfs-Brock R., and Aguilar A., "QA to AQ: Patterns about transitioning from Quality Assurance to Agile Quality," 3rd Asian Conference on Patterns of Programming Languages (AsianPLoP), Tokyo, Japan, 2014.

[YW] Yoder J. and Wirfs-Brock R., "QA to AQ Part Two: Shifting from Quality Assurance to Agile Quality," 21st Conference on Patterns of Programming Language (PLoP 2014), Monticello, Illinois, USA, 2014.

[YWW] Yoder J., Wirfs-Brock R. and Washizaki H., "QA to AQ Part Three: Shifting from Quality Assurance to Agile Quality: Tearing Down the Walls," 10th Latin American Conference on Patterns of Programming Language (SugarLoafPLoP 2014), Ilha Bela, São Paulo, Brazil, 2014.

Appendix

We have published several papers that outline core patterns for evolving from more traditional quality assurance to being agile at quality [YWA, YW, YWW14, YWW15, YWW16]. We briefly describe the entire collection patterns using patlets in the tables below. A patlet briefly outlines the gist of a pattern, usually in one or two sentences. The patlet names in bold have been written up as patterns. We organize our software-related Agile Quality patterns into these areas: identifying system qualities, making qualities visible, fitting quality into your process, and being agile at quality assurance. Our ultimate goal is to turn all patlets into full-fledged patterns and make a pattern language for action and change useful to software teams who want to become more agile about system quality.

Core Patterns

Central to using these QA patterns is breaking down barriers and knowing where quality concerns fit into your agile process. The following patlets describes these considerations.

Patlet Name	Description
Break Down Barriers	Tear down the barriers between QA and the rest of the development team. Work towards engaging everyone in the quality process.
Integrate Quality	Incorporate QA into your process, including a lightweight means for describing and understanding system qualities.

From here we classified our patterns into these categories: Identifying Qualities, Making Qualities Visible, and Being Agile at Quality which we outline below.

Identifying Qualities

An important but difficult task for software development teams is to identify the important qualities (non-functional requirements) for a system. Quite often system qualities are overlooked or simplified until late in the development process, thus causing time delays due to extensive refactoring and rework of the software design to correct quality flaws. It is important that agile teams identify essential qualities and make those qualities visible to the team. The following patlets support identifying the qualities:

Patlet Name	Description
Find Essential Qualities	Brainstorm the important qualities that need to be considered.
Agile Quality Scenarios	Create high-level quality scenarios to examine and understand the important qualities of the system.
Quality Stories	Create stories that specifically focus on some measurable quality of the system that must be achieved.
Measurable System Qualities	Specify scale, meter, and values for specific system qualities.
Fold-out Qualities	Define specific quality criteria and attach it to a user story when specific, measurable qualities are required for that specific functionality.

Agile Landing Zone	Define a landing zone that defines acceptance criteria values for important system qualities. Unlike traditional landing zones, an agile landing zone is expected to evolve during product development.
Recalibrate the Landing Zone	Readjust landing zone values based on ongoing measurements and benchmarks.
Agree on Quality Targets	Define landing zone criteria for quality attributes that specify a range of acceptable values: minimally acceptable, target and outstanding. This range allows developers to make tradeoffs to meet overall system quality goals.

Making Qualities Visible

It is important for team members to know important qualities and have them presented so that the team is aware of them. The following patlets outline ways to make qualities visible:

Patlet Name	Description
System Quality Dashboard	Define a dashboard that visually integrates and organizes information about the current state of the system's qualities that are being monitored.
System Quality Radiator	Post a display that people can see as they work or walk by that shows information about system qualities and their current status without having to ask anyone a question. This display might show current landing zone values, quality stories on the current sprint or quality measures that the team is focused on.
Quality Checklists	Create a quality checklist to use to help ensure important system qualities are being met.
Qualify the Roadmap	Examine a product feature roadmap to plan for when system qualities should be delivered.
Qualify the Backlog	Create quality scenarios and architecture items that can be prioritized on a backlog for possible inclusion during sprints.

Being Agile at Quality

In any complex system, there are many different types of testing and monitoring, specifically when testing for system quality attributes. QA can play an important role in this effort. The role of QA in an Agile Quality team includes: 1) championing the product and the customer/user, 2) specializing in performance, load and other non-functional requirements, 3) focusing quality efforts (make them visible), and 4) assisting with testing and validation of quality attributes. The following patlets support being agile at quality:

Patlet Name	Description
Whole Team	Involve QA early on and make QA part of the whole team.
Quality Focused Sprints	Focus on your software's non-functional qualities by devoting a sprint to measuring and improving one or more of your system's qualities.

Product Quality Champion	QA works from the start understanding the customer requirements. A QA person will collaborate closely with the Product owner pointing out important Qualities that can be included in the product backlog and also work to make these qualities visible and explicit to team members.
System Quality Specialist	QA provides experience to agile teams by outlining and creating specific test strategies for validating and monitoring important system qualities.
Automate First	Some tasks specifically tests can be hard to automate later. As you go along automate any and all tasks (specifically tests) that you can. Do this as soon as possible.
Spread the Quality Workload	Rebalance quality efforts by involving more than just those who are in QA work on quality-related tasks. Another way to spread the work on quality is to include quality-related tasks throughout the project and not just at the end of the project.
Shadow the Quality Expert	Spread expertise about how to think about system qualities or implement quality-related tests and quality-conscious code by having another person spend time working with someone who is highly skilled and knowledgeable about quality assurance on key tasks.
Pair with a Quality Advocate	Have developers work directly with quality assurance to complete a quality related task that involves programming.

Other QA to AQ Patterns:

There are many other QA activities such as code reviews, inspections, architecture prototyping or experimentation, which occur throughout development. It is important for iterative processes to include QA and evaluation activities throughout the whole development cycle. This will lead to other patterns which we have started to outline ideas for below.

- Exploit Your Strengths
- Value Quality
- Everyone has QA responsibilities
- Grow the Team
- Architecture Runway
- Quality Debt related to Technical Debt
- Define Quality Acceptance Criteria
- Making Quality Debt Visible and How to Manage
- Getting the Agile Mindset
- Perform an Experiment to Learn
- Responsible Moments
- Continuous Inspection
- Quality Risk Assessment
- Quality Tests
- Share the Quality Load