QA to AQ Part Four

Shifting from Quality Assurance to Agile Quality

"Prioritizing Qualities and Making them Visible"

Joseph W. Yoder [1], Rebecca Wirfs-Brock[2], Hironori Washizaki[3]

[1] The Refactory, Inc.,

[2]Wirfs-Brock Associates, Inc.

[3]Waseda University

joe@refactory.com, rebecca@wirfs-brock.com, washizaki@waseda.jp

Abstract. *As organizations transition to agile processes, Quality Assurance (QA) activities and roles need to evolve. Unfortunately, QA activities typically occur late in the process, after the software is fully functioning. As a consequence, QA departments have been "quality gatekeepers" rather than actively engaged in the ongoing development and delivery of quality software. Agile teams incrementally deliver working software. Incremental delivery provides an opportunity to engage in QA activities much earlier, ensuring that both functionality and important system qualities are addressed just in time, rather than too late. Agile teams embrace a "whole team" approach. Even though special skills may be required to perform certain development and Quality Assurance tasks, everyone on the team is focused on the delivery of quality software. The patterns in this paper are focused on identifying core qualities and prioritizing them on the project roadmap. Additionally during the process it is important to keep these qualities visible to the team and organization.*

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

Keywords

Agile Quality, Quality Assurance, Software Quality, System Qualities, Testing, Patterns, Agile Software Development, Scrum, Quality Related Acceptance Criteria, Agile Quality Scenario, Whole Team

Introduction

Nothing prevents QA from being involved throughout the development process, but generally this has not been the case. Unfortunately for many software projects, QA only becomes involved late in the development process, just before it is necessary to test and release the final product.

As organizations move to being more agile, it is important that this transition also includes Quality Assurance (QA). An important principle in most agile practices is the "Whole Team" concept. It isn't just testers who care about quality, the whole team does. Agile developers write unit tests to exercise system functionality, but there is much more to quality than testing system functionality. Ideally, agile quality involves a cross-functional agile team, with special expertise contributed by testers and quality engineers [CG]. Therefore having QA be a part of the team from the start can enhance efforts to build quality into the system and make attention to quality an integral activity of a more streamlined process. The whole team focuses on quality while delivering functionality.

One benefit of including QA throughout the development process is that they can help the team understand and validate both functional and nonfunctional (or system quality) requirements. QA can help the product owner understand what system quality requirements should be considered and when. And QA can assist the product owner with the definition of done which often needs to incorporate many important system qualities in addition to system functionality.

Previously in [YWA, YW, YWW] we presented an overview of patterns on ways to become more agile at quality as well as written thirteen patterns. We have written patlets for all the patterns we have identified so far which are listed in the appendix. A patlet is a brief description of a pattern, usually 1-2 sentences outlining the problem and solution. We are working on writing all of these patlets as full-fledged patterns that can ultimately help guide organizations as they become more agile at quality.

Central to successfully using these QA patterns is knowing where quality concerns can fit into your agile process. This includes ways for Breaking Down Barriers and Integrating Quality into your Agile Process. From here we classified our patterns into these categories: Identifying Qualities, Making Qualities Visible, and Being Agile at Quality. This paper expands on the Making Qualities Visible category by writing the patterns "Qualify the Roadmap", "System Quality Radiator", and "Qualify the Backlog.

Our patterns are written in the spirit of Edward Deming's fourteen principles for business transformation and improvement [De]. Consequently, our patterns focus on actions for improving software quality and integrating QA concerns and roles into the whole team. Our focus is not on technical software programming practices. We recognize that programming and development practices are vital and can significantly contribute to or detract from software quality. But many others have written about programming, design and architectural practices while ignoring organizational and QA related actions that can also improve software quality.

These patterns are intended for any agile team that wants to focus on important qualities for their systems and better integrating QA into their agile process. These patterns are for anyone who wants to instill a quality focus and introduce quality practices earlier into their process, too. These patterns need not just be for agile teams.

Qualify the Roadmap

"All you need is the plan, the roadmap, and the courage to press on to your destination."
— Earl Nightingale

Many agile teams include a product roadmap as part of their planning. This roadmap typically shows a rough plan for delivering features over time. This plan is useful for sharing a common understanding to the teams involved in the project and to help communicate stakeholders' expectations and overall project plans and goals across the organization. The roadmap includes a timeline with expected milestones and targets for when key features are desired.

As systems qualities are a key factor in the success of any product, how can agile teams include these qualities as part of the roadmap and overall timeline?

❖ ❖ ❖

Features that are delivered to end users are tangible and of obvious value to end users, so they are easy to focus on and prioritized in a roadmap. While the delivery of features may also depend on system qualities, it can be unclear how they are related and thus often not included in the roadmap.

Agile teams tend to do a good job of prioritizing and implementing end-user related features and including user stories for them on the backlog. Often, these user stories do not mention any system qualities. Consequently, understanding system qualities and when they should be considered can sometimes be difficult.

System design involves making tradeoffs between implementing functionality that is good enough to meet the important business requirements and adequately addressing system qualities. Often, when making design tradeoffs, there is a temptation to overdesign or get into too many details about technical qualities.

❖ ❖ ❖

Therefore, while developing and evolving the product feature roadmap, also plan for when system qualities should be addressed. Be sure to "*Plan for Responsible Moments*" to know when important system qualities should be implemented.

Typically a product roadmap contains high-level features which are implemented by many user stories. Order processing and fulfillment, for example, may be a high-level feature that is expected to be delivered, and thus it is represented on the roadmap. To implement order processing you may need to develop one or more services, install web servers, implement security processes, and access a transactional database. These might be represented by technical items on your roadmap. However, if performance or security are also important

qualities, you may also want to add specific items for these concerns to your roadmap, and appropriate system quality-related items to your product backlog. This helps to more clearly identify when certain performance targets or security mechanisms are expected. There are various practices that can help you *Find Essential Qualities*.

Product roadmaps include a timeline for when major features are desired. Sometimes they can also include architectural features. Alternatively, teams may create a separate technology roadmap that outlines the expected delivery of architecture components and technology. Regardless of whether you have a separate technology roadmap or identify architecture features on your features roadmap, important system qualities should be noted. It is important to make visible when important system qualities should be considered and worked on. There are ways to make these qualities more visible such as *Quality the Backlog*, *Quality Charts*, and *System Quality Radiators*.

If you don't make the delivery of these qualities explicit, then they might not be recognized as being needed by a certain time. Waiting too long to implement certain system qualities can cause rework to the architecture. If certain qualities are addressed at more responsible moments, such as when core pieces of the system are implemented, it can be much easier to deliver on important system qualities, limit technical risks and increase your chances of timely completing your project. Quite often these system qualities directly contribute to meeting your definition of done.

To uniformly consider and specify necessary system qualities in product roadmaps, standards for software and system quality models such as ISO/IEC 25010:2011 [ISO25010] can be considered. They classify typical quality characteristics and provide an extensive framework for systematically considering quality concerns. Agile teams might focus on a few important quality characteristics, such as reliability and security, as important at the beginning of the project. When considering additional qualities, such as usability and maintainability, you might be forced to make tradeoffs between reworking your design to support qualities originally considered or re-adjusting your expectations. Although it is not necessary nor desirable to specify and define requirements for all qualities from the very beginning, it is important to define essential qualities and identity where on the product roadmap they are expected to be delivered.

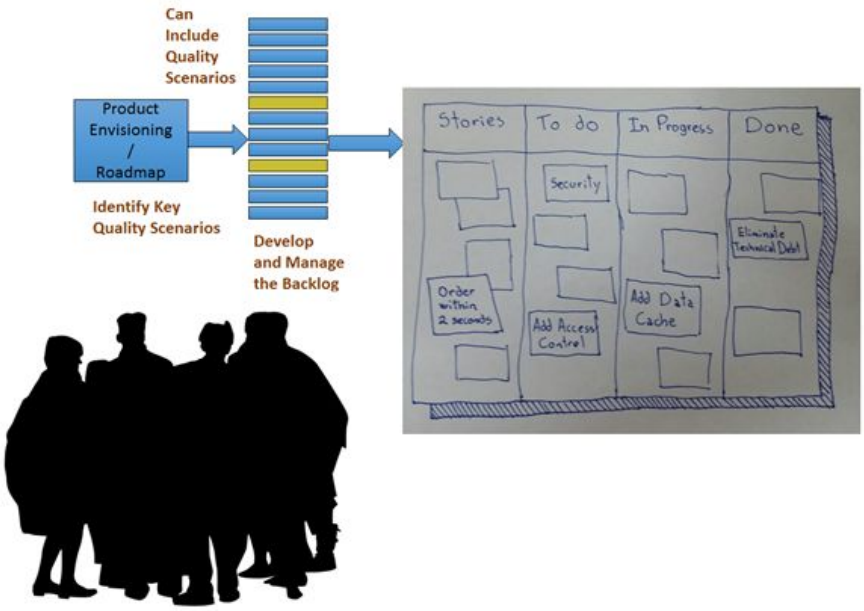
To ensure that quality assurance concerns are adequately addressed, standards for quality assurance processes and activities such as IEEE 730-2014 [IEEE730] and IEEE 1012-2012 [IEEE1012] define some activities that can be useful for agile teams to consider as they adjust or review their product roadmap. For example, IEEE 730-2014 specifies 16 software quality assurance activities in terms of purpose, outcomes and tasks. By referring to these activities, QA and other team members may develop an effective and consistent QA process tailored to the specific requirements and/or environmental constraints for their project.

It is important to address system qualities as the system evolves and more functionality is delivered. Checking that some qualities have been adequately addressed at any point in time does not guarantee that these qualities will continue to perform as measured after subsequent changes. In general, it is impossible to foresee when to ultimately check for a quality (however, it might be possible in a few specific cases). So a plan to check qualities should not be related to "when" but to "which components" or "which changes." Instead of just saying, "Check privacy of personal information at the end of third iteration," it should also include "and whenever this happens, check for...". One way to address this is to *Monitor Qualities*.

Qualify the Backlog

"Things come to me pretty regularly. There is never a shortage or a backlog."

— Duncan Sheik



Agile backlogs include a list of important features and technical tasks necessary to complete a project or a release. This backlog prioritizes the order that work should be done to help ensure success. The definition of done for each backlog item may also need to include important system quality requirements. However, certain system qualities cut across one or more user stories.

How can agile developers better understand the scope of the work that needs to be done, especially when it comes to understanding, implementing and testing system qualities?

❖ ❖ ❖

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.

Focusing too early on system qualities in the development cycle can lead to overdesign and premature optimization [Knuth1974]. However, ignoring these qualities makes it difficult to know what it will take to deliver them.

Product Owners are focused on system functionality and prioritize the backlog based upon the most important features that deliver value. Many of these product owners do not want to see these technical items cluttering the backlog. Or if the technical items are on the backlog, they may be given low priority.

When system quality requirements are buried in the acceptance criteria of specific user stories, development may overlook their importance or underestimate their effort. Desired system qualities do not just happen by magic or emerge appropriately along with the implementation. They take a commitment to quality as part of the ongoing work.

Some system qualities require a certain amount of infrastructure implemented before they are able to included and validated. A system isn't acceptable until it delivers functionality along with desired system qualities. It can be hard to know how much infrastructure is needed to deliver certain qualities.

Therefore, create and add specific quality items to your backlog. These items can include *Quality Scenarios*, *Quality Stories*, and *Fold-Out Qualities* for some user stories.

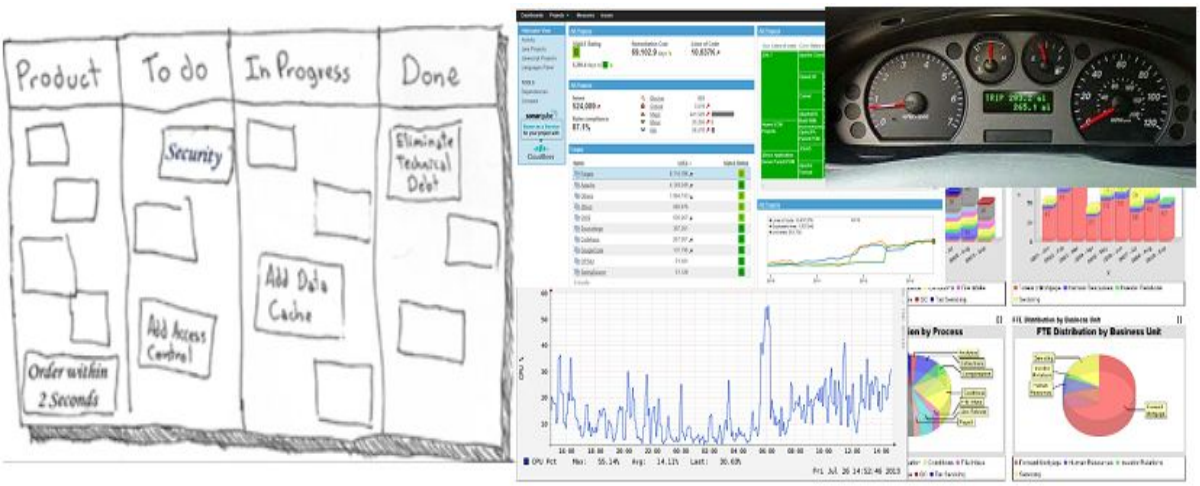
If you have identified *Quality Scenarios* in a Quality Workshop, these can be added to your backlog as individual work items. If a specific quality spans multiple user stories, for example, the aggregate performance of multiple business transactions, then this overall quality is more visible if you create a separate *Quality Story* and add it to your backlog to represent that overall requirement. Sometimes certain qualities will be noted when working on specific functional user stories. When then happens you can use a *Foldout Quality* instead. This ensures that the story isn't declared done until it is delivered along with its desired qualities.

If these qualities become cumbersome to manage on the product backlog, they can be put onto a separate technical backlog. The product owner may not want these items on the main product backlog as they want to primarily focus on the delivery of end-user features for the product. This technical backlog can possibly be worked on by a separate team dedicated to working on system quality and architecture concerns. One issue that must be resolved when having separate backlogs is how to coordinate the work and manage dependencies between system qualities and features based upon user stories. For example, reprioritizing a user story may cause you to reprioritize system qualities that are needed to support it. The backlogs need to be in alignment, or you won't deliver qualities in support of features.

There are several well-established approaches for conducting Quality Workshops and defining *Quality Scenarios* such as Quality and Architecture methods including Quality Attribute Workshops (QAWs) [Barbacci03], Scenario-Based Architecture Analysis (SAAM) and Architecture Tradeoff Analysis Method (ATAM) proposed by CMU/SEI [Bass12]. In the spirit of agile quality, quality workshops and reviews are conducted incrementally, as the system is developed. These meetings tend to be shorter and more focused on immediate architecture concerns. *Agile Quality Scenarios* and *Quality Stories* [YWA] can be written in these workshops to communicate important qualities.

System Quality Radiator

"The lavish presentation appeals to me, and I've got to convince others."— Freddie Mercury



Typically, agile software development focuses on features and functionality first before paying attention to other important system aspects such as architecture and critical qualities. On agile projects you hear statements like, "Make it work, make it right, then optimize it". Most agile practices push to develop important functional requirements as outlined by the product owner, which are prioritized on the work backlog.

As the system evolves the team begins to better understand what system qualities are important and how to better measure them. Keeping track of these qualities and what the current quality of the system becomes increasingly important. There are important qualities that are key to the success of the product.

How can agile teams provide a means to make important qualities of the system and their current status accessible and visible to the team?

❖ ❖ ❖

Understanding system qualities is important. While agile developers are good at developing based upon the requirements from user stories, understanding qualities and keeping them visible to the team can help the team know what is key for a project's success.

Creating and maintaining meaningful visible cue for some qualities is difficult. Unless there is activity or the values of the qualities are changing with some frequency, people will come to ignore them. But reminders are still important.

Creating a lot of tools and displays can seem like a pointless luxury compared to making sure the system is meeting the requirements well enough to ship. Creating displays takes time and often there are limited resources and people dedicated to building QA tools. There is often not enough time committed to consider important quality implications of one's design and what is important to measure and monitor.

It can be difficult to know what qualities are important to monitor. As more and more qualities are built into the system, certain ones are important to keep a watch on while others, once validated and made testable, are good enough.

Certain qualities such as performance and reliability, if not tracked regularly, can become very difficult to improve late in the development process. Although originally the system might meet quality constraints, as the system evolved sometimes qualities become invisible and as a consequence aren't maintained over time.

Therefore, post displays that people can see as they work or walk by that shows information about the system qualities you want to focus on and their current status without having to ask anyone a question.

A display might show current landing zone values, *quality stories* on the current sprint, reminders about quality-related activities, or quality measures that the team is actively working on. Sometimes a display will just show the results of certain system quality-related tests for the day. Sometimes, there are third-party tools that can be used to assist with monitoring thus giving a live display of key system qualities. Other times, the team might need to create specific tools that help monitor important qualities and make them visible (sometimes as plugin to their dev environment). When teams start monitoring the live system, their tools can evolve into a *System Quality Dashboard*.

You also may want to create a chart or listing of the important qualities of the system along with their objectives and also make them visible to the team; possibly on the agile board. This chart might contain reminders about the quality items to focus on for a particular sprint or set of sprints, instead of specific quality measure.

Following are examples of some potential quality-related reminders:

- Performance tune every service invocation….
- Make sure audit logs are generated….
- Focus on addressing security holes in….
- Keep working on caching…

Any quality radiator can be a blend of actual measures, targets, short term objectives and longer-term quality goals. There need not be just one quality-related display of information. It is important to update with new information or it will get ignored.

Summary

This paper is a continuation of patterns for shifting from Quality Assurance (QA) to Agile Quality (AQ). The complete set 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 focuses on three patterns for making quality visible. Ultimately it is the authors' plan to write all of the patlets as patterns and weave them into a 3.0 pattern language for evolving from Quality Assurance to an Agile Quality mindset.

Acknowledgements

We thank our shepherd Antonio Maña for his valuable comments and feedback during the PLoP 2015 shepherding process. We also thank our 2015 PLoP Writers Workshop Group, xxx, yyy, and zzz, for their valuable comments.

References

[Hil] Hile E., "Head On Collision: Agile QA Driving In A Waterfall World," Agile 2014 Conference, Orlando, Florida, USA.

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

[Kunth] Knuth, Donald: Structured Programming With Go To Satements, Computing Surveys, Vol 6, No 4, December 1974, 261-301.

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

[ISO25010] ISO/IEC 25010: 2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models

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

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

[Barbacci03] Mario R. Barbacci, Robert J. Ellison, Anthony J. Lattanze, Judith A. Stafford, Charles B. Weinstock, William G. Wood, "Quality Attribute Workshops (QAWs), Third Edition," Technical Report, CMU/SEI-2003-TR-016, 2003.

[Bass12] Len Bass, Paul Clements, Rick Kazman, "Software Architecture in Practice (3rd Edition)", Addison-Wesley Professional , 2012.

[Knuth74] Donald E. Knuth. Computer programming as an art. Commun. ACM 17, 12 (December 1974), 667-673.

Appendix

Previous papers on this topic have been published which outlines some core patterns we found when evolving from typical quality assurance to being agile at quality [YWA, YW, YWW]. We outlined the patterns using patlets in the tables below. A patlet is a brief description of a pattern, usually one or two sentences. The patlets in bold have been written up as patterns. We break 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 required to correct quality flaws. It is important in agile teams to 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.
Qualify the Roadmap	Examine a product feature roadmap to plan for when system qualities should be delivered.
Qualify the Backlog	Create quality scenarios 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 "Becoming 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.
QA Product 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.
Agile Quality Specialist	QA provides experience to agile teams by outlining and creating specific test strategies for validating and monitoring important system qualities.
Monitor Qualities	QA specifies ways to monitor and validate system qualities on an ongoing basis.
Agile QA Tester	QA works closely with developers to define acceptance criteria and tests that validate these, including defining quality scenarios and tests for validating these scenarios.
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 also many other QA activities such as code reviews, which naturally happening along the process. 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 possible patterns
- Reward all, make all equal
- Quality For All
- Grow the Team
- SAFE (Architecture Runway) “something related”
- Quality Debt related to Technical Debt (related to risk and project management)
- 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