Patterns About Transitioning from Quality Assurance to Agile Quality

Rebecca Wirfs-Brock, Wirfs-Brock Associates

The patterns in this collection focus on actions for improving software and system quality and integrating Quality Assurance concerns and roles early and often into your Agile development and delivery process. Although the actions you choose to take can vary depending on your team size, your organization and what you value, in general, most of these patterns can be applied to widely different contexts.

These patterns, written by Joe Yoder, Rebecca Wirfs-Brock, Ademar Aquiar, and Hironori Washikazi, were presented at various Patterns conferences (see references).

This paper describes the entire collection of patterns in pattern gist form. A *pattern gist* is a brief version of a fuller pattern description. The gist motivates the pattern, poses a question about the problem the pattern addresses, and then sketches the pattern solution.

Full pattern descriptions—typically 2 or 3 pages—can be found in patterns papers presented at various PLoP conferences.

The patterns in this collection are organized into these categories: core, identifying qualities, making qualities visible, and being agile at quality.

Category: Core Patterns

These patterns establish overarching goals for embedding quality-related activities and instilling a quality focus into Agile teams.

Pattern: Integrate Quality

Generally, Quality Assurance (QA) is not done until after many sprints or way late in the development process. Delaying QA testing until after many sprints have been completed can cause a lot of problems with work items that were thought to be good enough but weren't. System quality attributes, such as performance or security that are not addressed until way late in the process can cause upheaval in the architecture. If important system qualities had been recognized and considered during earlier sprints, some of them could have been incorporated at this earlier time resulting in less rework.

How can you incorporate examining important system qualities into your agile process and where does QA fit into the process?

As part of your agile process, create ways to understand, describe, develop and test for system qualities. This can be done through getting a high level understanding of what system qualities are important to your project and providing a means for describing them. The most important idea is to make QA part of the whole team and to integrate quality thinking into your agile mindset.

For example, if you are practicing Scrum, you would make sure this attention to system quality is part of your normal sprint including planning and testing. During the envisioning phase, important quality attributes should be considered and understood. Then, these can be prioritized into the backlog for consideration during sprints. During a sprint, any relevant quality tasks will be included. In addition to the normal functional and acceptance testing, the Scrum team will also develop tests to validate the system qualities or ways to monitor them through a dashboard.

Pattern: Break Down Barriers

Most agile processes do a good job of focusing on functional requirements, how to prioritize them, and on a collaborative environment that fully engages product owners, scrum masters and the development team. However, barriers can exist between people who are not "inside" the development team or feel tangential to it. This is especially important for contributors who may not be engaged full time on the project. Groups or individuals that contribute to an agile project need be engaged and feel like valued contributors to the software's success.

How can agile teams remove the barriers and become more agile at quality?

Tear down those barriers or walls that impede communication through various actions. Have QA fully participate in the team's estimation sessions. Have QA specialists move participate as part of the team bringing their expertise to the group. Have the Product Owner (PO), development team, and QA all be part of planning prior to the upcoming sprint.

An important principle in most agile practices is the "Whole Team" concept, where people work together to produce a high quality product. It isn't just testers who need to care about quality. Everyone on the team needs to care about quality, even though they bring different strengths and experiences to their work. Having QA as part of team from the start helps instill a quality mindset into the team and makes quality concerns an integral part of a more streamlined process.

Category: Identifying Qualities (8 patterns)

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.

Pattern: Find Essential Qualities

Quite often essential system qualities are overlooked or simplified until late in the development process. This can cause delays due to extensive refactoring and rework of the software design in order to correct quality flaws. To avoid extensive rework it is important that agile teams identify these fundamental qualities and make those qualities visible to the team in a timely manner.

How can agile teams understand essential qualities for an evolving system?

At the start of a project it is important to identify essential qualities critical to the success of the project. This can be done via an agile quality attribute workshop where you agree on essential qualities, and make sure they are visible to team. These workshops should include key members such as the product owner, developers, architects, quality assurance, and the customer. Whenever there are major changes to the roadmap or new system qualities become apparent, the team can choose to hold another quality workshop.

During a quality workshop, which might last an hour or two, simple collaborative techniques can be used to identify and characterize system qualities. People can identify a concern and write it on a sticky note that is associated with a specific system quality (such as performance or reliability). The team can vote on what they consider most important and urgent and then write Agile Quality Scenarios for those.

Pattern: Agile Quality Scenarios

Often backlog items are restricted to functional requirements. From these backlog items scenarios and user stories are written to elaborate them so that concrete tasks can be identified and work effort estimated. This incrementally helps the project move forward developing functionality. As the system evolves, however, there can be many other important system qualities such as security, performance, reliability and other qualities that also need attention. Typically these requirements have not been identified if a product backlog only includes functional requirements.

How can we get a good understanding and a high level view of the important qualities that need to be addressed during the development of the system?

Early on in the process, use a lightweight methodology to create and describe high-level quality scenarios that address important non-functional requirements such as

performance, load, reliability, and security. If you know that certain qualities are an important consideration, they can be prioritized as part of the product roadmap and included during relevant sprints. As more qualities become apparent, you can create scenarios for them as needed. These scenarios can be used in two ways: to drive the design of core aspects of a system based on quality concerns, or to capture a concrete scenario to evaluate whether the system's architecture satisfies that particular quality.

Quality scenarios describe desired system qualities following a general form that has these parts: the *source* of stimulus (or what causes the quality to be exhibited), the *stimulus* (or a brief summary of an action or event), the *artifact* and *environment* (what parts of the system under what operating conditions), the *response* (what happens when the system reacts), and the *response measure* (some concrete, tangible result you expect).

Pattern: Quality Stories

While creating and implementing user stories for functional requirements, you identify performance, usability, internationalization, reliability or other non-functional qualities that broadly apply to several user stories or across a number of features. At other times you may have a specific system quality that needs to be improved on, perhaps to achieve a landing zone target.

How can you make these quality requirements visible to the team and prioritized?

While working on different parts of the system or when important qualities need to be prioritized, create a quality story to represent the system qualities, especially for those that span multiple user stories or features. Create separate quality stories and add these to your backlog. In contrast, an agile user story is a short, brief description of a desired feature, told from the perspective of the person who desires that capability. Attached to a single user story there may also be story-specific fold-out qualities. But they don't give you a picture of the overall quality you need.

Your backlog can contain both quality-specific and functional user stories. Adding quality-specific stories to your backlog makes these quality requirements visible. It also allows the Product Owner to prioritize quality-related concerns along with system functionality.

Pattern: Measurable System Qualities

To know whether a desired quality has been achieved it has to be measured. The description of the quality and the specific aspect you are trying to measure can't be vague or fuzzy.

How can you decide on what values you expect for a quality and how to measure them?

Define an appropriate way to measure a quality and to describe it with only as much accuracy and precision as you need. This involves defining or finding an appropriate way to measure (the meter) and describing accurately the values you expect (the scale). There are three types of scales of measure: natural, constructed, or proxy.

A natural scale is one that is obviously associated with a specific quality and is usually the easiest to agree upon. A constructed scale is built specifically to measure and quantify a quality, for example, a 7-point user satisfaction scale. A proxy scale is an indirect measure of quality. When it is difficult, too costly or too early to directly measure system qualities use a proxy scale to measure parts of the system to give a feel for a certain system quality and then extrapolate expected values. You may need to construct a proxy scale when parts of the system are not yet completed or integrated. You may start by measuring using a proxy, then transition to a natural scale if you continue monitoring the quality in production.

Pattern: Fold-out Qualities

A user story or feature is considered shippable when it meets the expectations of a product owner and has the agreed qualities. Typically a Product owner's expectations are phrased as acceptance test criteria that is technology neutral, and at a high level. Some user stories have explicit system quality-related criteria that are part of accepting it as complete. In order for a story to be acceptable it must meet specific performance, usability, internationalization, reliability or other non-functional requirements.

How can you define and describe agreed upon system qualities that should be exhibited by an implemented story?

Create and attach specific quality acceptance criteria to the user story. These are called fold-out qualities because they are integral to accepting a user story, but they are not necessarily the first acceptance criteria you may identify. They unfold as you have deeper conversations about how your system should behave and what qualities it should exhibit. While your initial concern is correctly implementing that functionality, satisfying a fold-out quality can strongly influence your design and implementation choices. So, they are important to discuss and reach agreement about and more completely describe the definition of done.

Pattern: Agile Landing Zones

On a complex project or product, you need to be aware of those system qualities that contribute to your project's success. You don't want these essential success criteria to get lost in with the myriad of other requirements. You also need to make design tradeoffs as you implement your system. Almost always these tradeoffs have architectural implications, so your definition of success needs to be somewhat flexible—you may have to compromise on one design goal in order to achieve another.

How can you understand and monitor those system qualities that need to be addressed in a way that allows you to make thoughtful design tradeoffs?

Define and use an agile landing zone. Tom Gilb originally defined a landing zone as a set of criteria used to monitor and characterize the "releasability" of a product. An *agile* landing zone is one where all the criteria and acceptable values are not fixed or known at the beginning. For some system qualities, there isn't one specific number you are aiming for, but you know what is minimally acceptable. For other qualities, you may have specific targets, but you are willing to compromise on them in order to achieve other system quality objectives. You want flexibility in achieving some quality requirements and overall accountability.

The criteria and values of an agile landing zone take shape over the lifetime of a project. Landing zone criteria are similar to release criteria, except they provide for tolerances in acceptable values. There isn't one number you are aiming for; you have a range of values for each system quality attribute you are targeting. This gives you some flexibility in defining what's "good enough." There are three possible values for any landing zone criteria: minimum, target, and outstanding. A minimum value is something you are willing to live with, although you may aspire for a higher value. A target value is what you think you can achieve with reasonable cost and effort. An outstanding value is something that you believe might be achievable but not without significant effort.

Pattern: Recalibrate the Landing Zone

Initially, you defined a set of landing zone criteria that you expected to achieve over a few iterations. You left the rest of your landing zone purposefully sketchy. As you've implemented new functionality, you have continued to add new landing zone criteria while monitoring the values of existing ones.

How can you continue to evolve your landing zone and keep it up to date?

Revisit your landing zone criteria from time to time and reset expectations. Some initial values may not be appropriate, given what you know now. Because you are implementing your system incrementally and learning more about your system's capabilities and limitations, it is natural for the criteria in an Agile Landing Zone and their values to shift and be adjusted over time.

What initially appeared to be reasonable targets may change in light of new facts or market changes. You might recalibrate/readjust landing zone criteria based on new information/system capabilities/technologies. Previous implementation decisions may either positively affect or limit your ability to achieve newly identified criteria. Landing zones, like release criteria can and do change. In fact, changing acceptable values for your landing criteria is not always a bad thing to do, especially if you are reacting to the current situation and making thoughtful design tradeoffs.

Pattern: Agree on Quality Targets

There are several areas where you need to define specific quality-related targets. You may have targets for performance, usability, internationalization, reliability or other non- functional qualities that broadly apply to several user stories or across a number features. If you've done something similar in the past, the quality criteria to choose and their acceptable values may be obvious. At other times it can more of a challenge to reach agreement. How much improvement to strive for may be open to debate.

How can you reach consensus when defining quality acceptance criteria?

Work towards informed consensus on quality-related targets. Ideally a small group of informed individuals should agree upon target values. If you have diverse stakeholders with varying opinions, you may decide to give each stakeholder group a voice in identifying several qualities that are particularly relevant to them. However, you might want to choose someone, such as a business architect, product owner, or lead engineer who knows about the product to take a first cut at establishing reasonable values for quality criteria and values that are questioned, challenged, and then reviewed and agreed upon by a small group.

It is important to recognize that technical considerations impact quality targets. Any assumptions about how these values can be achieved should be noted. To reach consensus on specific quality targets, you may need someone to play the role of facilitator. The facilitator should know enough about the program or product to be constructive, but they need not be the "authority" or "expert." That person should be good at gaining consensus and get the best from individuals who may have strongly held opinions and disagreements. Ideally, a facilitator knows enough about the product to offer constructive observations and has the ability to lead a small group forward in defining acceptable criteria and values.

Category: Making Qualities Visible (5 patterns)

It is important for team members to be aware of important system qualities and have them readily available.

Pattern: System Quality Dashboards

As your system evolves the team begins to better understand what system qualities are important and how to better measure them. As time goes on, and more and more qualities are built into the system, keeping track of these qualities becomes increasingly important. Some are important to keep a watch on while others, once validated and made testable, are good enough. Although originally the system might meet quality constraints, qualities can degrade if they aren't monitored and maintained.

How can agile teams provide a means to make this information accessible and visible to the team?

The first step is to outline the critical items that need to be measured and monitored on an ongoing basis. Some of these can start off with simple measures, exercising and measuring only partially implemented functionality. Although initially you might be making simple measures, unless you incorporate them into a dashboard, they won't be readily visible.

So create a dashboard to monitor important qualities to provide ongoing, timely feedback. As important system qualities are outlined and included in the backlog, note which ones should be monitored and where tools can be created to measure the system as it evolves. If an existing tool provides all that is needed and is relatively easy to use, use the tool to create your dashboard. Some tools that provide powerful means of measurement can be costly or hard to use. So you may need to determine whether it is better to purchase a powerful tool or use an open source dashboard that may not be as powerful.

Pattern: System Quality Radiator

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

Post visual 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. System Quality Radiators can have many forms ranging from posters or displays to colored sticky notes on a Kanban board, to colorized backlog items. What is important is that the quality radiator is visible and easily understood. It is also important to keep the quality radiator up to date.

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, third-party tools can be used to present a live display of key system qualities and values—resulting in a Quality Dashboard.

Pattern: Quality Checklists

Some system qualities, such as security and usability, can be observed by executing system functionality. Other qualities, such as maintainability and extensibility are embodied in how the software is constructed. Even though you might be paying attention to quality, as your system evolves, problems will arise and there will be issues sustaining qualities. Quality requirements will inevitably change as you learn and parts of the system that were good enough at one time, might no longer meet today's requirements.

How can you ensure system-wide quality requirements are being considered and not overlooked as your system evolves?

Create checklists that include expectations for desired system qualities, which are common across the system and should be consistently met. Checklists can be reviewed by the team to ensure that qualities are met before features are released and verified by the team as part of quality assurance. Explicitly stating what qualities need to be delivered consistently across many different user stories, and what important qualities should be considered as new functionality is added can help the team keep quality requirements in mind.

There are two kinds of checklists: read/review and do/confirm. A read/review checklist is one where team members may perform tasks separately beforehand, then come together to affirm that these items have been successfully completed. A do/confirm checklist is one where each checklist item is performed on the spot and then verified by the team before proceeding to the next step.

Pattern: Qualify the Roadmap

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?

Product roadmaps. Typically a product roadmap includes a timeline for when high-level features, which are implemented by many user stories, are desired. While

developing and evolving the product feature roadmap, also plan for when system qualities and the architecture features to support them should be addressed.

Quality-related roadmap items should either be placed just before or along with any functionality that depends on them. This may seem contrary to that well-known agile mantra, “Make it work, make it right, make it fast.” However, if you have a risky architecture feature, you might want to work on that feature a bit before implementing functionality that depends on it. Isn’t this similar to using a spike solution and then refining that solution? 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 product roadmap, it is important to make visible when important system qualities should be considered and worked on.

Pattern: Qualify the Backlog

Agile backlogs include an ordered list of important features and technical tasks necessary to complete a project or a release. This backlog prioritizes the order that work is done. 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?

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, then this overall quality is more visible if you create a separate Quality Story and add it to your backlog to represent that quality requirement. Sometimes certain qualities are related to 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.

Category: Being Agile at Quality (8 patterns)

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 as an integral part of a whole team approach to quality.

Pattern: Whole Team

Traditionally Quality Assurance teams belong to a separate group. Typically, QA in most organizations has not had good access to business stakeholders. As a consequence, they generally prefer a lot of documentation and prefer to specify their tests based on detailed written specifications. Although QA likes a lot of documentation, the quality of that documentation can be inconsistent or outdated. And since testing takes so much effort, QA has traditionally preferred to test a fully functioning system in order to minimize re-testing and rework. Since QA typically has not been engaged until late in the process, serious time-to-market pressures can cause compromises to quality. Problems can arise when QA is not part of the development team (creating an us vs. them syndrome).

How can you better incorporate QA into an agile team?

Include QA as part of the team from the start. When QA is included as part of the agile team from the beginning, QA can help everyone on the team understand and validate requirements. QA is also able to assist with the definition of done and help product owners understand what quality attributes should be considered and when they should be addressed.

The role of QA shifts from being an outsider on a different team to being a team member on a unified "Agile Team." This transition from "outsider" to "team member" increases the team's overall knowledge about quality. By being part of the team throughout, QA assists the team by keeping those qualities are important visible and to help know when working on specific system qualities best fits into the process (when to do what for different qualities)

Pattern: Quality-Focused Sprint

Features don't make a viable system; rather a viable system is accomplished by focusing on features accompanied by paying attention to system qualities. If you have only concentrated on implementing functionality, you are delivering working software each sprint. But it may not meet the demands of a production environment, which has more demanding users, higher volumes of data, more transactions, and more of, well everything.

How can you incorporate these other non-functional requirements into your system as needed?

Take time to focus on your software's non-functional qualities and devote a sprint to measuring and improving one or more of your system's qualities. Set expectations that no new features will be delivered, focusing on a better system for the result.

Like any other sprint, you need to identify and prioritize work and create a backlog. However, the nature of the work in a quality-focused sprint will be different: instead of functional stories, you need to identify and prioritize stories about the qualities you are trying to improve.

Improving one quality can impact other system qualities. The definition of "done" for quality-focused sprint involves more than just implementing and verifying improvements. It can also involve measuring the impacts your quality improvements have on existing system functionality and potentially revising your quality acceptance criteria.

Pattern: Product Quality Champion

Many agile teams and product owners are focused on the delivery of important features for the system. However, it requires more than implementing the features before any system can be considered done. While delivering features is critical to the success of the project, the system is not "ready for release" until critical system qualities have also been addressed.

As the system's features are delivered, how can the team pay attention to systems qualities as well?

Include as part of your agile team a Product Quality Champion. This is someone who helps the team keep focused on important system qualities. This person is involved from the start of the project understanding the customer requirements and continues working throughout assisting the team with a quality focus. Typically a *Product Quality Champion* doesn't have "quality champion" in their job title. This is a role or task that they take on in addition to their other responsibilities. A product quality champion can come from QA. Business analysts, architects, or product managers may also be product quality champions.

A product quality champion doesn't let quality issues slide, and works to build consensus around system quality requirements and how they might be delivered. The quality champion or advocate collaborates closely with the Product Owner and other team members pointing out important qualities that can be included in the product backlog. They also work to make these qualities visible and explicit to all team members by leading *Quality Workshops*, setting up *Quality Radiators* or *Quality Dashboards* and generally promoting enthusiasm about product quality.

Pattern: System Quality Specialist

Quality assurance on agile projects primarily focuses on validating and verifying user stories that express requirements in terms of system functionality. QA may be more comfortable and familiar with functional testing. Nonetheless, production software needs to exhibit system qualities such as being scalable, usable, secure, and reliable 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?

When your team is lacking specific skills, include System Quality Specialists 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 with deep technical skills related to specific system qualities. The term specialist sometimes has a bad connotation, implying that knowledge is unnecessarily held too closely or poorly communicated to others. Some agile teams even go so far as to avoid hiring specialists. However it is wishful thinking to believe you will only have "t-shaped" 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 are not necessarily t-shaped.

The System Quality Specialist can be temporary until the team acquires the necessary skills, or the specialists could become full- time team members if the need is ongoing. The specialists work with the team by directly assisting them with the quality-related tasks. They are hands-on rather than merely advice givers. This specialist may not be familiar with agile practices or processes. Effectively incorporating them into your team may mean that you need to work with them to understand your agile values and preferred ways of working. And you may want to adapt your process based on their inputs and advice.

Pattern: Spread the Quality Workload

Agile teams spend most of their time specifying, implementing, and verifying functionality. 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?

Rebalance quality efforts by involving more than just those who are in QA or have QA roles to work on quality-related tasks. Spread the quality workload over time by including quality-related tasks throughout the project. The goal is to take a balanced approach to tackling quality work including the definition, implementation, and validation of system qualities. This all comes down to everyone working together to make the project successful, pitching in when needed, not only when being told to.

Developers already have a responsibility and ownership for code quality and helping make sure it meets the core business requirements including system capabilities and functionality. However, a developer can also assist with validating system qualities. 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. 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.

Pattern: Automate as You Go

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 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?

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. 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 other things you can automate such as acceptance tests, performance metrics, code smell detection, application security checks, and architectural conformance. As you automate tasks, they become part of the cadence of your project.

If you have repetitive, tedious or error prone tasks, and it is feasible to do so, automate those as well. The more you automate repetitive manual tasks, the more time it frees you up to do more. It also allows time to spend on exploratory testing. Automation also allows you to more safely evolve the system and lets you do work in smaller batches, making fewer mistakes and getting quicker feedback.

Pattern: Shadow the Quality Expert

As an organization grows, it is important to also grow and evolve quality expertise along with the agile team. Often organizations do not have the resources or people to completely fulfill their Quality Assurance needs. Quality experts can have deep technical and product knowledge. A whole team philosophy leads you to want to not lock up or isolate expertise, but instead spread knowledge across the team and grow "T-shaped" skills within your organization. T-shaped people have skills and knowledge that are both deep and broad.

How can organizations grow quality expertise and spread knowledge and ideas about qualities across the teams?

Have various people follow or shadow a QA expert while they are doing their tasks. The QA expert works as a mentor teaching by actively involving the shadow as an apprentice in understanding what happens and what needs to be considered during important quality tasks.

Early on, a shadow can follow a QA expert around, observing and taking notes. It is important that the shadow asks relevant questions and has close interactions with the QA expert. The expert can explain what she is thinking as she is doing her work. As the shadow becomes more confident and learns new skills, they become more involved in performing quality tasks. Shadowing takes time and commitment. The best scenario for effective shadowing is when the QA expert has enough time and patience to actively involve the shadow with their daily tasks.

Pattern: Pair with a Quality Advocate

Quality Assurance is much more than just testing and validating the software. There are important quality considerations that if the team is made aware of can help with the success of the system.

How can the agile team build quality into the system, especially when it comes to understanding and testing for system qualities?

Pair developers and other agile team members with quality assurance to complete quality-related tasks and to spread QA knowledge and quality perspectives. The synergy achieved by various roles pairing with a quality advocate has mutual benefits. Pairing can help the team understand key qualities and how to validate and think about them. QA can pair with the Product Owner or Project Manager to inform them about how the software is expected to work and what kinds of testing is going on. A Product Owner may not otherwise be aware of how the system is tested and how testing for system qualities differs from the unit tests developers are writing. While pairing, QA can gain a deeper understanding of project priorities and how quality-related work needs to be made more visible.

During programming tasks, QA members can pair with developers. Testers typically focus on testing from the users' perspective. As they pair with developers, testers gain a deeper understanding of how the software works behind the scenes. This helps QA to identify potential areas that might require more testing as well as help them better isolate defects. Developers learn much about testing boundary conditions, good interfaces, and input validation from QA, as well as what conditions might lead to potential failure. Pairing QA with a DBA can be useful, especially when isolating data schema or data access issues. QA often has a deeper understanding of how the software is expected to function than the DBA. Both come to better appreciate each other's concerns and get ideas for improved testing and validation of the system.

References

The patterns in the QA to Agile Quality collection were presented and workshopped at the following Patterns conferences. Copies can be found in the ACM digital library or Rebecca Wirfs-Brock's website (See www.wirfs-brock.com/rebecca/collections/Agile-Quality-Patterns for downloadable copies of the full patterns papers).

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

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

[YWW14] Joseph Yoder, Rebecca Wirfs-Brock, and Hironori Washizaki. 2014. "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.

[YWW15] Joseph Yoder, Rebecca Wirfs-Brock, and Hironori Washizaki. 2015. "QA to AQ Part Four: Shifting from Quality Assurance to Agile Quality: Prioritizing Qualities and Making them Visible," 22nd Latin American Conference on Patterns of Programming Language (PLoP 2015), Pittsburgh PA, USA, 2015.

[YWW16a] Joseph Yoder, Rebecca Wirfs-Brock, and Hironori Washizaki. 2016. "QA to AQ Part Five: Being Agile at Quality: Growing Quality Awareness and Expertise," 5th Asian Conference on Patterns of Programming Language (AsianPLoP 2016), Taipei, Taiwan, 2016.

[YWW16b] Joseph Yoder, Rebecca Wirfs-Brock, and Hironori Washizaki. 2016. "QA to AQ Part 6: Being Agile at Quality: Enabling and Infusing Quality," 23rd Conference on Pattern Languages of Programs (PLoP 2016), Monticello, Illinois, USA, 2016.