QA to AQ Part Five

Being Agile at Quality
"Growing Quality Awareness and Expertise"

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 evolve. The whole team focuses on quality as they incrementally deliver working software. Incremental delivery provides an opportunity to engage in QA activities much earlier, ensuring that both functionality and system qualities are focused on just in time, rather than too late. Knowing what specific qualities need to be paid attention to and checking that they have been appropriately addressed is important. The patterns in this paper extend our previous work with three patterns aimed at increasing quality awareness and expertise: "Quality Checklists", the "Product Quality Champion", and "Shadow the Quality Expert".*

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, Quality Checklists, Quality Champion, Whole Team

Introduction

As organizations evolve to being more agile, it is important that the "Whole Team" focuses on quality. Having QA be an integral part of the team from the start enhances efforts to build quality into the system. 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. QA can assist with the definition of done which often needs to incorporate many important system qualities in addition to system functionality.

Quality is still easy to overlook. People are not always good at remembering to do things they should do to ensure quality. Having reminders can help make sure that critical qualities are not ignored. Sometimes the impact a decision can have on quality may not be obvious. Getting into the mindset of a quality expert takes time, especially with understanding what qualities need to be addressed before a system can be released.

Previously in [YWA, YW, YWW14, YWW15] we presented an overview of patterns to become more agile at quality (see appendix). In this paper we expand on ways for growing quality awareness and expertise by writing three additional patterns: *Quality Checklists*, the *Product Quality Champion*, and *Shadow the Quality Expert*.

Using *Quality Checklists* not only makes important qualities visible, they are helpful reminders of quality items that might otherwise be ignored or overlooked. A *Product Quality Champion* promotes awareness of qualities, ensuring they are understood and appreciated by the Product Owner and team. *Shadow the Quality Exper*t is an opportunity for an expert to actively mentor others as they perform quality-related tasks.

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. Although there is an agile focus, these patterns are for anyone who wants to instill a quality focus and introduce quality practices earlier into their process. These patterns need not just be for agile teams.

Quality Checklists

"Checklists seem able to defend anyone, even the experienced, against failure in many more tasks than we realized."—Atul Gawande, The Checklist Manifesto: How to Get Things Right

© Can Stock Photo Inc. /bolsunova.

A common challenge is how best to handle non-functional requirements or system qualities. 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.

Agile teams need to consider many different system qualities as they implement the system. Quality-related acceptance criteria can be added to a specific user story (*Fold-out System Qualities*) to define quality-related acceptance criteria that apply to that user story. In addition, some system qualities need to be consistent across many user stories, such as how to consistently handle security. Some quality requirements, such as transaction throughput, or the number of concurrent users, represent aggregate behaviors that cut across many different user stories and broadly affect how the software should be designed and structured.

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?

❖ ❖ ❖

User stories aren't complete until their acceptance criteria are satisfied. Stakeholders often forget to include quality-related requirements in their acceptance criteria. Stakeholders may assume the developers know what to do and they will include work on qualities as they implement the user stories. Even if stakeholders are familiar with specific quality requirements and specify those for a user story, they may naively assume that other system qualities will be elsewhere, because they were present in already implemented functionality.

Understanding what qualities are important can be difficult if the team only relies upon tacit knowledge and assumptions about system qualities. Specific qualities may be different for new functionality but the team needs to know what is expected of "standard" functionality in order to keep quality in mind.

If the team is skilled at agile development and has a good sense of what needs to be done about system quality, they can consistently implement system qualities as they implement new functionality. Knowing what system quality concerns should consistently be addressed can be a big problem, especially as the size of the team grows.

During the evolution of a system, even if you know what is expected, you can still be surprised. System performance may degrade for something that was previously tested and released. Because there initially weren't any performance problems, you are lulled into a false sense that your architecture will continue to be adequate. Then, as new features are added and released something that you thought no longer needed attention causes problems.

Developers who come from more traditional development backgrounds, often assume that they only need to implement what is specified by the requirements. Often user stories leave out technical and non-functional qualities. There can be a big gap between what is specified in a user story and what needs to be consistently implemented for a system to be usable, scalable, or maintainable.

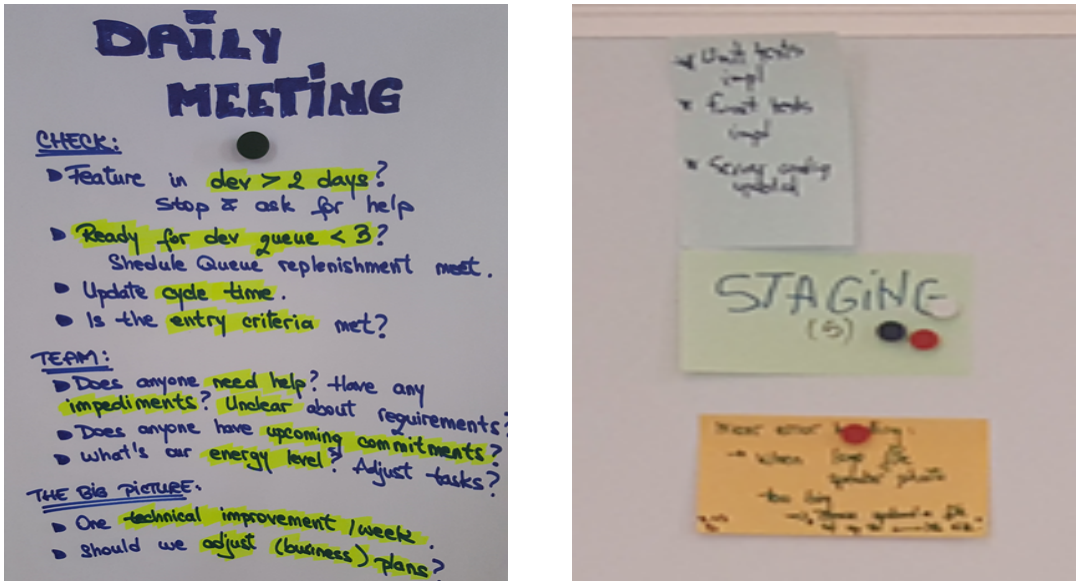
Expectations for quality goals can change as new technologies are adopted. So today's quality expectations might be greater than what was acceptable earlier in the life of the software. Being explicit about what qualities are consistently expected across an entire system allows you to adjust quality expectations as your system evolves.

Therefore, 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.

A checklist is a good way to keep quality-related expectations and actions in mind. 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 [Gaw]. 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.

A good place to use a checklist is where you have time to pause and reflect before proceeding. Finding these pause points isn't difficult for agile teams. There are natural places to run through a checklist, such as part of a sprint retrospective or release planning meeting, after integration testing, pre-deployment, post-build, or whenever an *Architecture Trigger* condition occurs. In fact, checklists can be run through daily as the teams perform their work. We know of one agile team that has a daily standup checklist which includes a checklist item for whether the team has worked on improving the system, reducing technical debt. They also have checklists for moving between states on their kanban board.



Daily Standup Checklist and a Checklist on the Kanban Board at Mozaic Works

Checklists can be proactively created based on past experience or perceived risks. For example, data migration has lots of potential risks such as loss of critical information. Based upon previous experience a checklist and migration process with rollback process can be created and verified before migration starts. The checklist will include places to pause and confirm during the migration process that things went well and, if there are any issues, actions that can be taken for recovery before proceeding with the next migration step.

A good time to update a checklist is when have a problem meeting your system qualities in spite of good intentions. You knew what to do, but even so, you didn't take the necessary actions to avoid slipping up. A reminder to check before you take action might make you pause, stop, and take appropriate action. For example, after costly downtime, one team added to their deployment checklist an item to check that a schema reversal script was written and verified before any production database schema changes were deployed.

You also might discover a problem through retrospectives or issues that arise while attempting to meet a specific quality objective. A checklist should be updated whenever it no longer fulfills its goal of ensuring system quality. If you find items that are being ignored or overlooked, consider revising your checklist. Team members need to develop, own and update their checklists as they evolve how they work. It is important to not create a speculative checklist trying to guess what might be important.

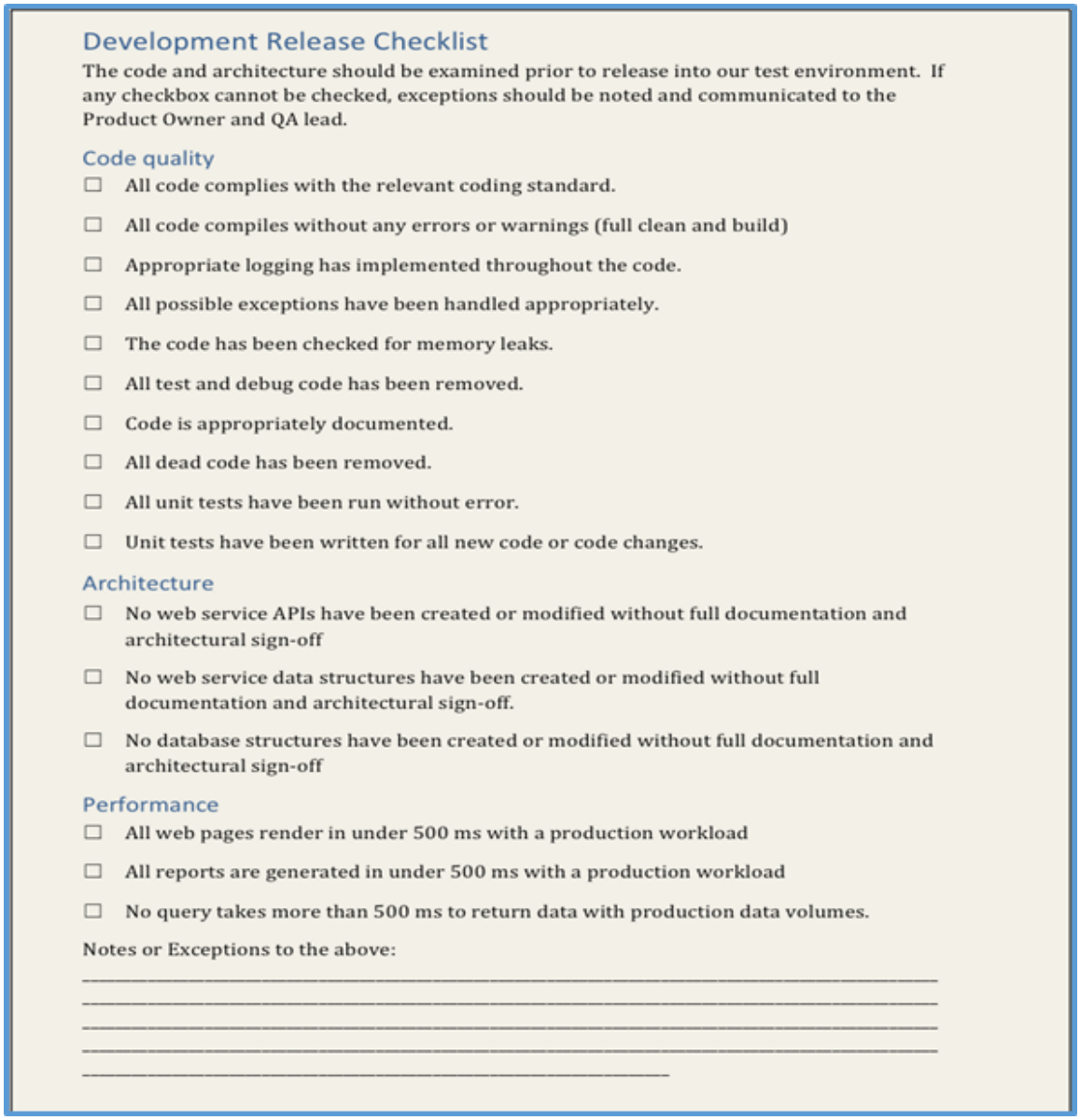
Checklists may initially be perceived as being burdensome or overly specific. Developers, feeling constrained by an unwieldy checklist may resist using them. A well-designed

checklist can relieve developers and QA of having to keep in mind every quality-related requirement as they dive deep into implementation details. To be accepted by the team, a quality checklist should be mutually agreed upon, and considered as a reasonable guide during development and a final check by the team before features are deployed.

Atul Gawande in *The Checklist Manifesto*, explores how checklists have been adopted in the medical profession [Gaw]. He states that one big problem of accepting checklists is overcoming the stigma associated with following a procedure or a checklist: "It somehow feels beneath us to use a checklist, an embarrassment. It runs counter to deeply held beliefs about how the truly great among us—those we aspire to be—handle situations of high stakes and complexity. The truly great are daring. They improvise. They do not have protocols and checklists. Maybe our idea of heroism needs updating."

If medical doctors can learn to use checklists and improve the quality of their work, agile teams can too. Rather than stifling creativity, checklists for system qualities can act as a safety net, making sure developers keep what's important about specific system quality requirements in mind, and that QA knows what is expected of the system in general, in addition to detailed or specific user-related system qualities.

Here is an example of a checklist from our PLoP 2015 workshop colleague, James Thorpe:



Development Release Checklist

The code and architecture should be examined prior to release into our test environment. If any checkbox cannot be checked, exceptions should be noted and communicated to the Product Owner and QA lead.

Code quality

- ☐ All code complies with the relevant coding standard.
- ☐ All code compiles without any errors or warnings (full clean and build)
- ☐ Appropriate logging has implemented throughout the code.
- ☐ All possible exceptions have been handled appropriately.
- ☐ The code has been checked for memory leaks.
- ☐ All test and debug code has been removed.
- ☐ Code is appropriately documented.
- ☐ All dead code has been removed.
- ☐ All unit tests have been run without error.
- ☐ Unit tests have been written for all new code or code changes.

Architecture

- ☐ No web service APIs have been created or modified without full documentation and architectural sign-off
- ☐ No web service data structures have been created or modified without full documentation and architectural sign-off.
- ☐ No database structures have been created or modified without full documentation and architectural sign-off

Performance

- ☐ All web pages render in under 500 ms with a production workload
- ☐ All reports are generated in under 500 ms with a production workload
- ☐ No query takes more than 500 ms to return data with production data volumes.

Notes or Exceptions to the above:

_____
_____
_____
_____
_____

Development Release Checklist from James Thorpe

It is important that this checklist fits on a single page or less. The above example includes what is important to James' company for their projects. In this checklist, items for code quality and specific performance measures are specified. There is the ability to waive a quality requirement if it isn't met, but the exception to the general requirement needs to be briefly explained.

What makes a good checklist?

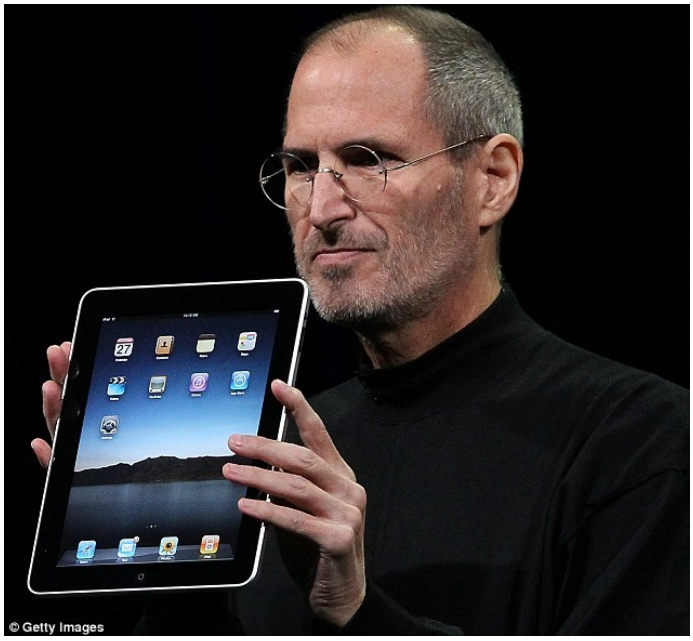
- Not too many items to check
- Organized into system quality categories or, alternatively, by component
- Specifies measurable values for quality attributes which cut across the architecture and/or affect many different stories
- It specifies baseline values for important qualities
- It is tailored to your specific project or product

Some checklists, such as those for code quality, should be evaluated before declaring that you are "done" for each sprint. However, other checklists such as performance and security can only be verified when the whole system is integrated. You might also write or use tools to assist with some automatic checks for the checklists but they are not a replacement for explicitly inspecting that checklist items are verified. For example, another agile team added a checklist item for the team to review any changes to threading code on their web application before deployment, after a hasty last minute "fix" that seemed obvious to a developer, broke their production system.

It is fine to have multiple checklists. Any checklist should be specific to the task at hand. You use it to tick off items that need to be met before proceeding to the next step in your process. For example, you might have specific checklists for code quality, performance, and security. Checklists can be at different levels of granularity and quality-related items can be part any checklist. For example you might have a quality checklist for releasing your software into production which has detailed steps that must be taken for setting up the environment or validating rollback, etc. Or you might have a performance related checklist that verifies you haven't fallen below critical *Agile Landing Zone* minimum values before planning work for your next sprint.

Product Quality Champion

"Quality is never an accident. It is the result of intelligent effort."— John Ruskin



AP Photo/Paul Sakuma, (CC BY-NC 2.0) — https://goo.gl/wIztw7

When organizations transition to agile, they often do a good job at getting developers and other team members to focus on agile development practices and prioritizing items that are important to the delivery of the project. However, as they focus on delivery, quality can seem to get left aside and in fact, quality seems worse than it was before.

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 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?

❖ ❖ ❖

Product owners prioritize the backlog. Given all their conflicting advice and things that are competing for their attention, how can quality-related items be included on the backlog and made part of the roadmap?

Product Owners primarily focus on the highest priority features first, in order to get feedback as to their usefulness. However, a system is not ready for production until important system qualities have also been addressed.

A Product Owner can be pulled in different directions from stakeholders. Some stakeholders do not understand how functionality can be impacted by quality. Product Owners may think they are technical, but they may be out of touch, so they don't necessarily appreciate the effort required to meet specific quality targets.

Product Owners can worry about losing sight of the important goals of the project, especially when quality concerns are raised that compete for the team's attention. They may think that focusing on these system qualities is over-engineering a solution. Sometimes they are right, but at other times there are important architecture concerns that will affect quality and need to be addressed.

Often you need to make tradeoffs between various qualities that have significant architecture impacts. If these are not examined and prioritized at the responsible moments, they can lead to a lot of technical debt which might led to a lot of refactoring or a Big Ball of Mud [FY].

Therefore, include as part of your agile team a Product Quality Champion. This is someone who helps the team keep focused on important system qualities.

To deliver a system that is, someone needs to be an advocate for the overall qualities of the system too. This is achieved by having a *Product Quality Champion* be included as part of the *Whole Team* working as a quality advocate. 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. A product quality champion can come from QA. Business analysts, architects, or product managers may also be product quality champions. Sometimes you know of a person that has excitement and enthusiasm around product quality. As a manager or agile coach, you support and encourage them in this role. At other times, you may need to find a willing team member and help them grow into this role.

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. It is especially important that the quality champion and the Product Owner have a good working relationship. When the *Product Quality Champion* raises a quality concern, the Product Owner needs to pay attention. On the other hand, raising too many concerns can deflate their importance. Ultimately the Product Owner makes the final decision on the prioritization of the backlog based upon careful consideration from the *Product Quality Champion* along with input from stakeholders and the team.

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. This person can be part of the agile team or they might even come from outside of the team. If quality is a high priority, then it is important to make clear who is responsible for this role. 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. One product quality champion we know was a master at getting the team to come to agreement on *Measurable System Qualities* for *Agile Quality Scenarios*. Another quality champion persisted at bringing to the attention of the Product Owner whenever performance degraded, even when that news was unwelcome. Sometimes it may seem like a *Product Quality Champion* is only the bearer of bad news. However, if you are a product quality champion, it is equally important for you to make visible the successes the team has at improving system quality and make visible the small successes of the team.

The *Product Quality Champion* can *Break down Barriers* through regular interactions with all stakeholders. During envisioning, they will help to *Quality the Roadmap* and during sprints they can work with the Product Owner to *Qualify the Backlog* or lead workshops to *Find Essential Qualities.* They might speak up at retrospectives and planning meetings to make sure quality concerns are heard and get involved whenever quality checklists are updated.

When things are not going so well, and people are not listening to quality concerns, it is important for a *Product Quality Champion* to be more of a quality promoter rather than a complainer. A *Product Quality Champion* might need to employ some fearless change patterns [MR, MR2015] such as *Accentuate the Positive, Small Successes, Whisper in the General's Ear, Guru on your Side,* or *Elevator Pitch*; specifically when communicating important qualities to the team, trying to get buy-in about the importance of qualities and ensuring they receive the appropriate attention.

Shadow the Quality Expert

''Tell me and I forget. Teach me and I remember. Involve me and I learn.''

— Benjamin Franklin

Photo/acute_tomato, (CC BY-NC 2.0) — https://goo.gl/7GkEl6

As an organization grows, it is important to also grow and evolve quality expertise along with the agile team. Often organization do not have the resources or people to completely fulfill their QA needs. Quality experts can have deep technical and product knowledge. It is desirable to spread around this expertise to minimize the "bus factor[1]." 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 [Brown].

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

❖ ❖ ❖

Companies can reorganize or downsize QA teams. Companies can grow rapidly thus leading to a need for more QA. This can lead to QA being spread too thin and having too much work to do. This can sometimes lead to important qualities being overlooked until right before or right after the software is released. The challenge is how to use what existing experience you have to help grow an understanding of important quality practices within the organization.

QA may be asked to do more than they've done in the past (not just only manual tests, but exploratory testing and performance testing, too). QA expertise can become highly specialized--I do UX testing, you do automated tests, I do manual testing, but then the

[1] A project's bus factor (or alternatively truck factor) is the number of persons it would need to lose in order for the project to lose its institutional memory and halt its progress. The bus factor is a measurement of the concentration of information in individual team members [CH]. https://en.wikipedia.org/wiki/Bus_factor

workload shifts. Sometimes work needs to be evened out (not every developer is busy all the time with writing production code).

Sometimes teams cannot afford full time QA resource. What if you don't have enough quality people, how can you share that information to the team so that the team knows how to at least address the important system qualities? QA work is sometimes offshored. This can make it difficult for communication and shared understanding.

It is important for the Whole Team to understand quality, not necessarily be experts but at least get a better understanding. How can you grow the agile team to spread quality expertise around? Ultimately everyone needs to understand quality.

Therefore, 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.

Shadowing QA and getting a better understanding of their tasks can help prevent a "weakest link" in the QA team. Developers and other team members can shadow and learn some of the QA tasks and QA's ways of thinking. A shadow is analogous to an apprentice or job shadow (https://en.wikipedia.org/wiki/Job_shadow). There are side benefits from shared experiences: QA can learn more from the developers and team members' perspective and vice-versa.

By spreading expertise, the Whole Team improves its ability to think about system qualities or implement quality-related tests. This can help with quality-consciousness by having another person spend time working with someone who is highly skilled and knowledgeable about quality assurance on key tasks. Ultimately the "quality" mindset can spread throughout the team.

Early on, a shadow can follow a QA expert around, observing and taking notes. It is important that the shadow asks relevant questions and have 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. 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.

People with specialized expertise think more effectively and problems solve better than non-experts. Bransford, Brown and Cocking [BBC] point out:

> "Research shows that it is not simply general abilities, such as memory or intelligence, nor the use of general strategies that differentiate experts from novices. Instead, experts have acquired extensive knowledge that affects what they notice and how they organize, represent, and interpret information in their environment. This, in turn, affects their abilities to remember, reason, and solve problems."

Experts tend to think in terms of higher level goals and strategies when solving a problem within their area of expertise. Rather than following procedures, experts are quickly able to determine what to do given the current context, and adjust their tactics based on changing situations. Because experts notice features and patterns of information that novices do not, effective shadowing takes someone who is good enough to learn from an expert (not a rank

novice) paired with an expert who is able to verbalize all the things they may be thinking about out loud, relating the knowledge and higher order abstractions/bigger concepts/bigger picture that the expert has in her head as she is solving the problem. Not all experts are adept at articulating what they are thinking as they are problem solving.

When identifying experts and shadowing opportunities, consider whether the QA expert is able and willing to explain what she is thinking while performing some task. It is also important that the "shadow" have enough skills and knowledge to be able to ask questions and get into the mindset of the expert.

There are different approaches for shadowing the QA expertise. For example, you might start with a training workshop to build skills followed by practicing the quality-related tasks with the Quality Expert. Other shadowing approaches and situations will be determined by the Quality Expert based on the context and need.

Shadowing a Quality Expert can be part time or task specific, depending on your needs. A good time for shadowing is when a new member joins the QA team and has the opportunity to learn some important practices and values and how to think about quality concerns from the Quality Expert. Another opportune time for shadowing is when you want to grow skills and have someone other than the Quality Expert develop competency at performing a critical quality-related task.

It can be hard to effectively shadow if QA is spread too thin or is under strict time constraints. Although you can still shadow, it might slow down QA. Shadowing takes time and commitment. Shadowing might fail if there is too much pressure on producing immediate results or there is only half-hearted buy in from the shadow or the expert or management.

Shadowing can also be used to train a *Quality Product Champion*. Shadowing with a Quality Expert is similar to *Pairing with a Quality Advocate* but with a longer term goal of building deep expertise rather than assisting the developers with quality related tasks. If QA is too busy, you can bring in a *Mentor* [MR] to assist with skill building.

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 focused on three patterns for growing quality and expertise. The authors plan to write more QA to AQ patterns and weave them into a pattern language for evolving from Quality Assurance to an Agile Quality mindset.

Acknowledgements

We thank James Thorpe for sharing his development release checklist and Alex Balboaca for sharing his checklists used at MozaicWorks. We thank our shepherds Chien-Hung Liu and Yu-Chin Cheng for their comments and feedback during the AsianPLoP 2016 shepherding process. We also thank our 2016 AsianPLoP Writers Workshop Group participants G Priyalakshmi, Paulina Andrea Silva Ghio, Emiliano Tramontana, Jesper Lundgren, Chunwei Lin, and Yu-Cheng Shao for their valuable comments.

References

[BBC] Bransford T., Brown A., and Cocking, R., *How People Learn: Brain, Mind, Experience and School.* National Academy Press, 2000.

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

[CH] Coplien J. and Harrison, N., *Organizational patterns of agile software development*. Wiley, 2004.

[FY] Foote B., Yoder J., "Big Ball of Mud," Pattern Languages of Programs Design 4 Harrison N., Foote B., and Rohnert H., editors. Addison Wesley, 2000.

[Gaw] Gawande A., *The Checklist Manifesto.* Picador, 2009.

[MR] Manns, M. L., and Rising, L. *Fearless Change: Patterns for Introducing New Ideas*. Addison-Wesley Professional, 2004.

[MR2015] Manns, M. L., and Rising, L. *More Fearless Change: Strategies for Making your Ideas Happen*. Addison-Wesley Professional, 2015.

[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.

[YWW14] 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.

[YWW15] Yoder J., Wirfs-Brock R. and Washizaki H., "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 2014), Pittsburgh PA, USA, 2015.

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.

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.
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 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.

Product Quality Champion	A person with Quality experience (maybe from QA) works closely with the agile team from the start of the project keeping a quality focus during development.
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. Spread the work on quality by including quality-related tasks throughout 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 an apprentice work with someone highly skilled and knowledgeable about quality assurance.
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 other activities that contribute to quality. It is important for agile and iterative processes to include QA and evaluation activities throughout the whole development cycle. Exploring these tasks will lead to other patterns which we have started to outline ideas for below.

- Exploit Your Strengths
- Reward all, make all equal
- Quality For All
- Grow the Team
- Architecture Runway
- Quality Debt related to Technical Debt (related to risk and project management)
- System Quality Acceptance Criteria
- Agile Quality Requirements
- 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 Workshop
- Quality Tests
- What not to test
- Automate First (Automate, Automate, Automate)
- Share the Quality Load
- No QA for small groups (Everyone is QA)