# 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 attention to system quality need to evolve along with the evolution of development practices. Agile quality teams incrementally deliver working software while ensuring that important system qualities are also addressed. In order to pay appropriate attention to system qualities, they need to be visible and included as part of the prioritized work. This paper presents patterns for identifying system qualities and including them on the project roadmap, adding quality-related work items to the project backlog, and creating a quality radiator that communicates the status and goals for delivering system qualities.*

## Categories and Subject Descriptors

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

## General Terms

Agile, Quality Assurance, Patterns, Testing

## Keywords

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

## Introduction

As organizations evolve to being more agile, it is important that they also pay appropriate attention to quality while delivering functionality. System qualities need to be visible and included as part of the prioritized work. Also, it is important to know where quality concerns can fit into your agile process and how to break down barriers between quality assurance and the rest of the organization to better integrate quality into your agile process. Previously in [YWA, YW, YWW] we presented an overview of patterns to become more agile at quality as well as wrote thirteen patterns (see appendix).

In this paper we expand on ways for ***Making Qualities Visible*** by writing three additional patterns: identifying system qualities and including them on the project roadmap (*Qualify the Roadmap*), adding quality-related work items to the project backlog (*Qualify the Backlog*), and creating a quality radiator that communicates the status and goals for delivering system qualities (*System Quality Radiator*).

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 significantly contribute to or detract from software quality. Since many others have written about programming, design and architectural practices, we focus our patterns on QA related actions and increased visibility of system quality requirements in order to help improve overall software quality.

Our patterns are intended for any agile team wanting to focus on important qualities for their systems and better integrating QA into their agile process. These Agile Quality patterns are for anyone who wants to instill a quality mindset 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 include in a roadmap. While the delivery of features may also depend on system qualities, it can be unclear how they are related and thus system qualities and architectural capabilities needed to deliver them are 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 while adequately addressing system qualities. Sometimes when making design tradeoffs, there is a temptation to overdesign or get into too many details about technical qualities. On the other hand, trying to address important system qualities after basic functionality has been implemented can result in major rework. There might be a more appropriate time to address these qualities that would cause less disruption.

❖ ❖ ❖

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

The product roadmap is a high-level view showing epics about how the product is likely to grow across several major releases. 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 support by additional technical items on your roadmap. However, if performance or security is also important, you may also want to add specific items for these concerns to your roadmap.

Quality-related roadmap items should either be placed just before or along with any functionality that depends on them. Note, 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?

During planning, corresponding system quality-related items should be added to your product backlog or ToDo List. Adding quality related work to your backlog helps to more clearly identify when certain performance targets or security mechanisms are expected and kept help you understand priorities of quality-related items when you *Qualify the Backlog*.

Product roadmaps include a timeline for when major features are desired. Sometimes they can also include architectural features to achieve desired system qualities. 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. There are additional ways to make these qualities more visible such as *Qualify the Backlog*, *Quality Charts*, and *System Quality Radiators*.

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

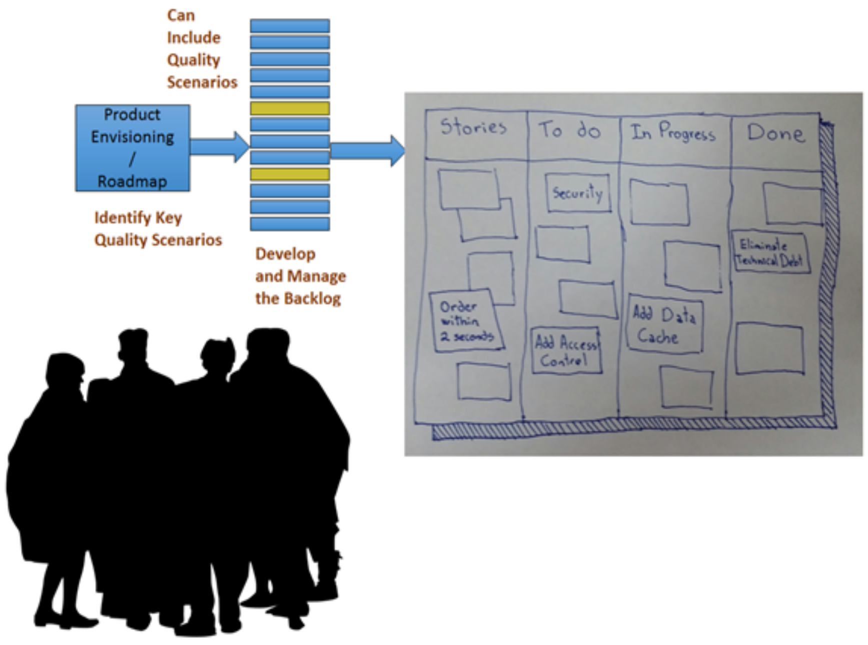
To uniformly consider and specify necessary system qualities in product roadmaps, standards for software and system quality models such as ISO/IEC 25010:2011 [ISO] can be considered. They classify typical quality characteristics and provide an extensive framework for systematically considering quality 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 neither 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 continue addressing 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 remain stable 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 simply "when" but also address "which components" or "which changes." Instead of just saying, "Check privacy of personal information at the end of third iteration," a plan should also include "and whenever this happens, check for…".

## Qualify the Backlog

*"Things come to me pretty regularly. There is never a shortage or a backlog."*
— Duncan Sheik


Can Include Quality Scenarios
Product Envisioning / Roadmap
Identify Key Quality Scenarios
Develop and Manage the Backlog
Stories
To do
In Progress
Done
Security
Eliminate Technical Debt
Order within 2 seconds
Add Access Control
Add Data Cache

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?**

❖ ❖ ❖

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. On the other hand, focusing too early on system qualities in the development cycle can lead to overdesign and premature optimization [Knuth].

Product Owners are focused on system functionality and prioritize the backlog based upon the most important features that deliver value. Many 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 due to lack of understanding of their impact on the overall system design.

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 to be 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 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.

If these qualities become cumbersome to manage on the product backlog, they can be put into 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. While some believe there should be only one backlog, experience has shown that there can also be benefits at times to having a separate technical backlog. It is important to do whatever adds the most value to the team.

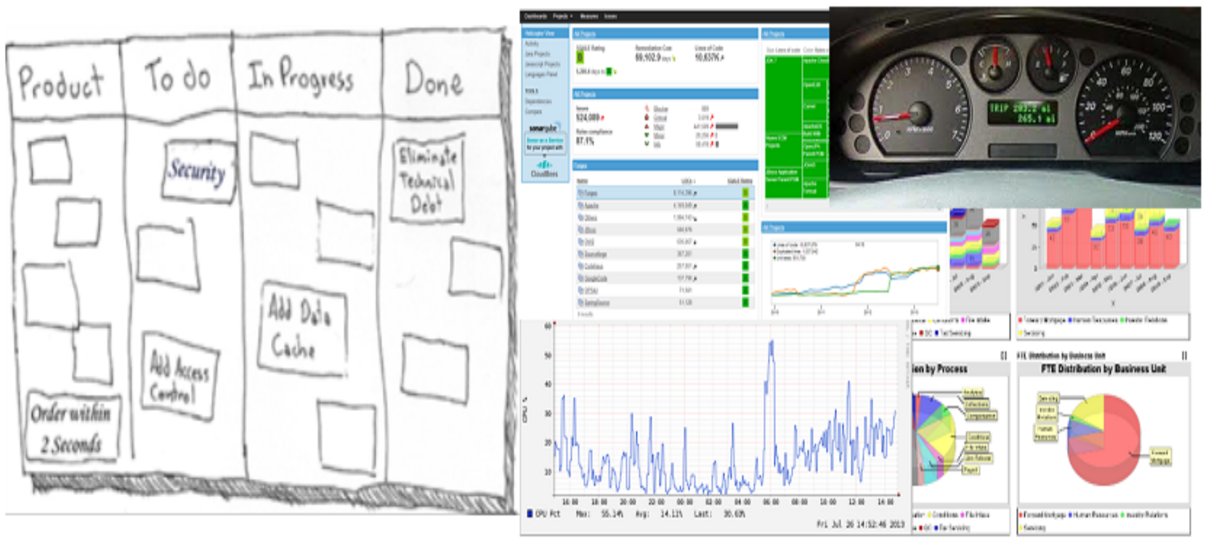
A separate team dedicated to working on system quality and architecture concerns can possibly work on this technical backlog. One issue that must be resolved when having separate backlogs is how to coordinate the work and manage dependencies between system qualities and features based upon user stories. 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) [Bar], Scenario-Based Architecture Analysis (SAAM) and Architecture Tradeoff Analysis Method (ATAM) proposed by CMU/SEI [Bass]. 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.

One of the consequences of having quality-related items on the backlog is they are visible. However, if they are not assigned a high enough priority, they will not get done. It is important that the team and the product owner understand the impacts of ignoring quality. Phillippe Kruchten suggests coloring backlog items which are invisible to users to make them stand out: architecture features including quality-related items are colored yellow and technical debt reduction items are colored black [Kru].

## 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 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 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?**

❖ ❖ ❖

Agile developers are good at developing code based upon the requirements from user stories. Understanding what qualities are important in addition to system functionality is also important. Making them visible can help the team know what is key for a project's success.

Creating and maintaining meaningful visible indicators for some qualities is difficult. Unless there is activity or the values of the qualities change with some frequency, people will tune them out. Reminders are valuable, but even if relevant, they can be ignored.

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 often isn't time to carefully 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 of the system is built and qualities are implemented, certain ones are important to monitor. Some qualities, like performance, might degrade with the addition of new capabilities. Others, once validated and made testable, are not likely to change over time.

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 evolves, 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.**

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. As noted by Alistair Cockburn [Coc]:

> *"An Information radiator is a display posted in a place where people can see it as they work or walk by. It shows readers information they care about without having to ask anyone a question. This means more communication with fewer interruptions."*

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.

Other times, the team might need to create specific tools that help monitor measurable system make them visible (sometimes as a plugin to their development environment). When teams start monitoring the live system, their tools can evolve into a *System Quality Dashboard*. Some qualities are not directly measurable but still need attention. These qualities can still be put on radiators so the team is still reminded of them.

Quality radiators can include dashboards, with the main difference between a radiator and a dashboard being that a radiator is purposefully intended to communicate quality conditions and information at a glance. A dashboard, on the other hand might include very detailed information that isn't easily interpreted by a casual but interested observer. A dashboard isn't necessarily a quality radiator but can be included as part of a quality radiator.

You also may want to create a chart or listing of the important qualities of the system along with their objectives and also make that visible to the team; possibly on the agile board. This chart might contain reminders about specific 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…

A 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 any quality radiator with new information fairly frequently or it will get ignored. If too much information is being radiated, then it can be difficult to interpret what's really critical and important qualities and changing values might be lost or ignored.

## Summary

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

**Acknowledgements**

We thank our shepherd Antonio Maña for his valuable comments and feedback during the PLoP 2015 shepherding process. We also thank our 2015 PLoP Writers Workshop Group, Michael John, Eduardo Guerra, Wiebe Wiersema, Cees de Groot, and James Thorpe, for their valuable comments and suggestions for additional patterns to add to this collection.

## References

| | |
|---|---|
| [Bar] | Barbacci M., Ellison R., Lattanze A., Stafford J., Weinstock C., Wood W., “Quality Attribute Workshops (QAWs), Third Edition,” Technical Report, CMU/SEI-2003-TR-016, 2003. |
| [Bass] | Bass L., Clements P., Kazman R., S*oftware Architecture in Practice (3rd Edition)*, Addison-Wesley, 2012. |
| [Coc] | Cockburn, A., *Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams*, Addison-Wesley, 2004. |
| [De] | Deming, W. Edwards, *Out of the Crisis*, MIT Press, 1986.x |
| [Iba] | Iba, T. 2011, “Pattern Language 3.0 Methodological Advances in Sharing Design Knowledge,” International Conference on Collaborative Innovation Networks 2011 (COINs2011). |
| [IEEE730] | IEEE Standard 730-2014 - IEEE Standard for Software Quality Assurance Processes, 2014. |
| [IEEE1012] | IEEE Standard 1012-2012 - IEEE Standard for System and Software Verification and Validation, 2012. |
| [ISO] | ISO/IEC 25010: 2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011. |
| [Knuth] | Knuth, D., “Structured Programming With Go To Statements,” Computing Surveys, Vol 6, No 4, December 1974, pp. 261-301. |
| [Kru] | Kruchten, P., Blog post - “The Missing Value of Software Architecture,” http://philippe.kruchten.com/2013/12/11/ /the-missing-value-of-software-architecture, 2013. |
| [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. |

## Appendix

We have published several papers that outline core patterns for evolving from typical quality assurance to being agile at quality [YWA, YW, YWW]. We outlined the entire collection 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 to correct quality flaws. It is important that agile teams identify essential qualities and make those qualities visible to the team. The following patlets support identifying the qualities:

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

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

***Making Qualities Visible***

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

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

***Being Agile at Quality***

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

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

| | |
|---|---|
| Product Quality Champion | QA works from the start understanding the customer requirements. A QA person will collaborate closely with the Product owner pointing out important Qualities that can be included in the product backlog and also work to make these qualities visible and explicit to team members. |
| 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 many other QA activities such as code reviews, inspections, architecture prototyping or experimentation, which occur throughout development. It is important for iterative processes to include QA and evaluation activities throughout the whole development cycle. This will lead to other patterns which we have started to outline ideas for below.

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