QA to AQ

Patterns about transitioning from Quality Assurance to Agile Quality

Joseph W. Yoder[1], **Rebecca Wirfs-Brock**[2], **Ademar Aguiar**[3]

[1] The Refactory, Inc.,

[2]Wirfs-Brock Associates, Inc.

[3] FEUP, Departamento de Engenharia Informatica,
Faculdade de Engenharia, Universidade do Porto

joe@refactory.com, rebecca@wirfs-brock.com, ademar.aguiar@fe.up.pt

Abstract. *As organizations transition to agile processes, Quality Assurance (QA) activities and roles need to evolve. Traditionally, QA activities have occurred late in the process, after the software is fully functioning. As a consequence, QA departments have been "quality gatekeepers" rather than actively engaged in the ongoing development and delivery of quality software. Agile teams incrementally deliver working software. Incremental delivery provides an opportunity to engage in QA activities much earlier, ensuring that both functionality and important system qualities are addressed just in time, rather than too late. Agile teams embrace a "whole team" approach. Even though special skills may be required to perform certain development and Quality Assurance tasks, everyone on the team is focused on the delivery of quality software. This paper outlines 24 patterns for transitioning from a traditional QA practice to a more agile process. Six of the patterns are completely presented that focus on where quality is addressed earlier in the process and QA plays a more integral role.*

Categories and Subject Descriptors

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

General Terms

Agile, Quality Assurance, Patterns, Testing

Keywords

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

Introduction

As organizations transition to agile processes, the role of Quality Assurance (QA) needs to evolve. Nothing prevents QA from being involved throughout the development process, but often this does not happen. Unfortunately, many QA people only become involved late in the development process, just before it was necessary to test and release the final product. This has been so primarily because of a different mindset between QA in traditional software processes and Agile QA. Generally, QA's primary responsibility is to certify the functionality of the application based upon the contract and requirements; usually with black-box tests. Most QA groups work independently from the software team. However, in Agile, QA works closely with the team on an ongoing and daily basis.

Not focusing on testing early enough can cause significant problems, delays and rework. Correcting functional flaws can be time-consuming. But correcting performance or scalability deficiencies can require significant changes and modifications to the system's architecture. If important system qualities are considered and addressed during earlier sprints, significant architectural verification could be performed much earlier, preventing significant disruptions or delays as architectural flaws are corrected. Agile teams incrementally deliver working software. Incremental delivery provides an opportunity to engage in QA activities much earlier, ensuring that in addition to functionality, important system qualities can be addressed in a timely fashion, rather than at the end of development.

QA in agile groups can benefit by being more proactive, working to ensure quality at all levels of the development process. Consequently, they can and do work closely and coordinate between business, management and developers. To be effective, Agile QA teams require additional skills to those of a "more traditional" QA team. For example, they often need to know how to understand the code, know how to write their own automated suite cases, and be involved in all parts of the agile process.

An important principle in most agile practices is the "*Whole Team*" concept. It isn't just testers who care about quality. Ideally, agile testing involves a cross-functional agile team, with special expertise contributed by testers [CG]. Agile developers write unit tests to exercise system functionality. But there is more to quality than unit testing. Therefore having QA be a part of the team from the start can help build quality into system and make attention to quality part of a more streamlined process. This will help the team to know what system qualities are important and how they fit into the process (when to do what for different qualities). Another benefit of including QA is that they can help the team understand and validate requirements. QA also can help the product owner understand what quality attributes should be considered and when. And QA can assist the product owner with the definition of done which often needs to incorporate many important system qualities in addition to system functionality.

This paper presents patterns for transitioning from a traditional QA practice to a more agile one, where quality is addressed earlier in the process and QA plays a more integral role. 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.

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

We break our software-related Agile Quality patterns into these categories: fitting quality into your process, identifying system qualities, making qualities visible, and being agile at quality assurance. This paper will outline twenty-four patlets organized into four categories: knowing where quality concerns fit into your process, identifying system qualities, making quality visible, and being agile at quality assurance. We expect to evolve and extend these categories and patterns over time.

A patlet is a brief description of a pattern, usually one or two sentences. Additionally, we take six of these patlets and write them as patterns for this paper: *Integrate Quality, Agile Quality Scenarios, Quality Stories, Fold-Out Qualities, Whole Team and Quality Focused Sprint*. Our patterns are written using a modified version of Takashi Iba's Patterns 3.0 format [Iba]. While similar to the traditional Alexandrian pattern form, the most important differentiation of the Pattern Language 3.0 is their orientation towards pattern readers who explicitly use pattern to design their own actions in a collaborative environment. Our ultimate goal is to turn all patlets into full-fledged patterns.

Fitting Quality Into Your Process

Central to successfully using any QA pattern is knowing where quality concerns might fit into your process and the removal of any physical and organizational impediments that prevent you from taking action.

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.

While our patterns focus specifically on quality, others have written patterns about how to introduce new ideas and change into organizations [MR]. They are specifically focused on actions to garner buy-in to new ideas, to bring groups together, and develop a common shared purpose and vision. Unless you also tackle any significant organizational resistance to change, it may be difficult to achieve your quality-related goals.

We view organizational patterns as being complementary to those we write about quality. They are critical to consider when attempting any significant change in process and how people collaborate.

Integrate Quality

"Quality is never an accident; it is always the result of high intention, sincere effort, intelligent direction and skillful execution; it represents the wise choice of many alternatives." —William A. Foster

Generally, 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 we incorporate examining important system qualities into our agile process and where does QA fit into the process?

❖ ❖ ❖

Often, QA is overworked and is part of a separate team. QA is often slammed by the forces upstream from them and they are constantly in a response mode. Although they'd like to help more there just isn't enough time or people.

QA can be seen as the obstacle to getting the product out which can often lead to an "us and them" mentality between QA and the development team.

It is important for agile teams to focus on features and important functionalities. Certain system qualities might not seem important, as they don't give the instant gratification of showing something useful to the end-user.

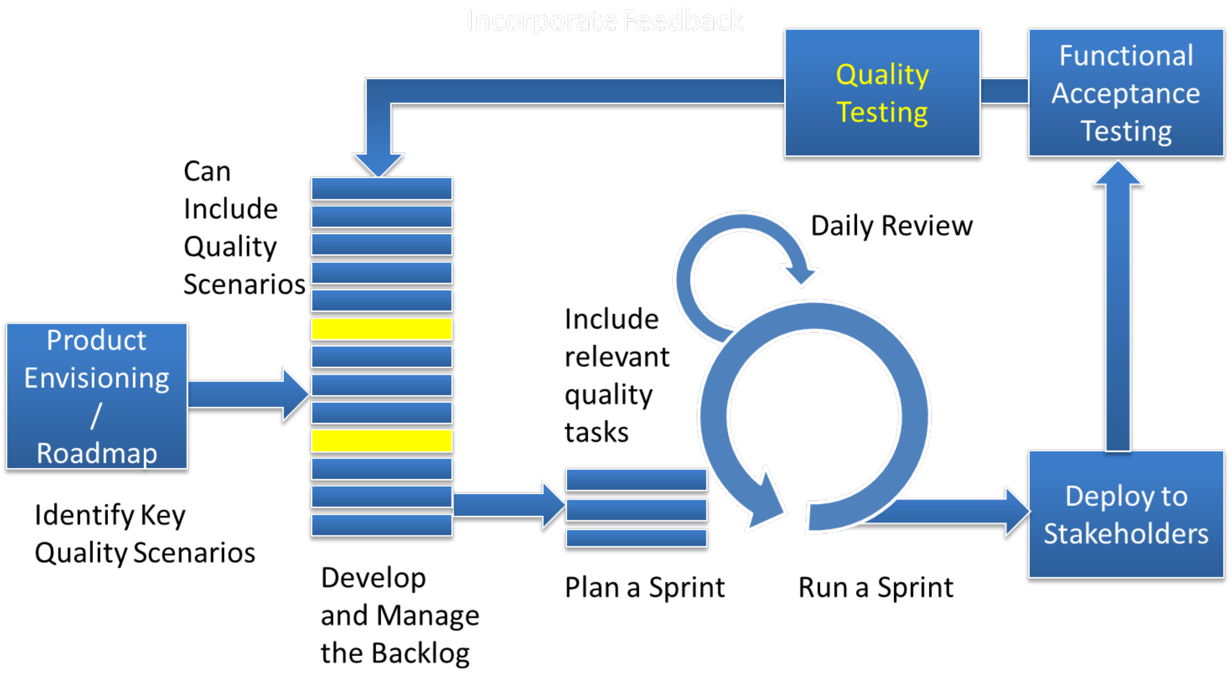
Many team members do not have a quality focus and often do not understand various system qualities. Developers are good at implementing the system based upon the product requirements from user stories, while QA has a lot of expertise understanding system qualities and how to validate these.

❖ ❖ ❖

Therefore, 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 with *Quality Scenarios*. Work on quality can be included in the product backlog tasks and ultimately you can write *Quality Stories* to help with identifying, testing and validation of system qualities. This could include having your QA person with the product owner (PO) to identify important qualities and ensure they get included on the backlog for inclusion in sprints.

There are various ways for an agile team to do this. 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. Figure 1 outlines an example of how you might add quality activities and focus to your Scrum process.

During the envisioning phase, important quality attributes should be considered and understood. Then, by working with the product owner, these can be prioritized into the backlog for consideration during sprints. During a sprint, any relevant quality tasks will be included and QA can assist with the creation of *Quality Stories* and identify *Fold-Out Qualities* for specific user stories. 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.



Incorporate Feedback
Quality Testing
Functional Acceptance Testing
Can Include Quality Scenarios
Daily Review
Product Envisioning / Roadmap
Include relevant quality tasks
Deploy to Stakeholders
Identify Key Quality Scenarios
Develop and Manage the Backlog
Plan a Sprint
Run a Sprint

Figure 1 - Quality in Scrum

Identifying Qualities

An important but difficult task for software development teams is identifying 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. The following patlets support Identifying Qualities:

Patlet Name	Description
Find Essential Qualities	Brainstorm the important qualities that need to be considered and list them for inclusion on the product roadmap.
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.
Measureable 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.

This paper will describe three of these patlets as patterns: *Agile Quality Scenarios, Quality Stories, and Fold-out Qualities*. *Agile Quality Scenarios* are important to make sure the teams can glean important system qualities and understand what is important early on so that these qualities are not only understood, but also prioritized and included on the road map and the most responsible time. *Quality Stories* are useful to the team while developing the system for prioritizing and including these qualities on the backlog. Sometimes, it is good enough to just include some of the important system quality requirements as part of a normal agile user story and *Fold-out Qualities* provides a means for this.

Agile Quality Scenarios

“Quality begins on the inside... then works its way out.” —Bob Moawad

During sprints, items from the product backlog are taken off and estimated. 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?

❖ ❖ ❖

Delaying the consideration and implementation of core system qualities can lead to an upheaval of the system tearing up the architecture, possibly requiring a lot of refactorings or muddy code.

It is important to identify what qualities are important for consideration early so that they can be prioritized and also help with the “definition of done.”

It can be hard to understand how qualities affect different parts of the system. Having some way to show the important qualities from a high level perspective can be very useful to the agile team.

Agile teams do not like a lot of detailed documentation and prefer lightweight methodologies for describing important requirements including system qualities.

❖ ❖ ❖

Therefore, 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. *Quality 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.

The *Quality Scenarios* can ultimately be used to create *Quality Acceptance Stories* to help with testing and validation of qualities. Figure 2 is an example template of a high-level quality scenario as adapted from the SEI [BCK].

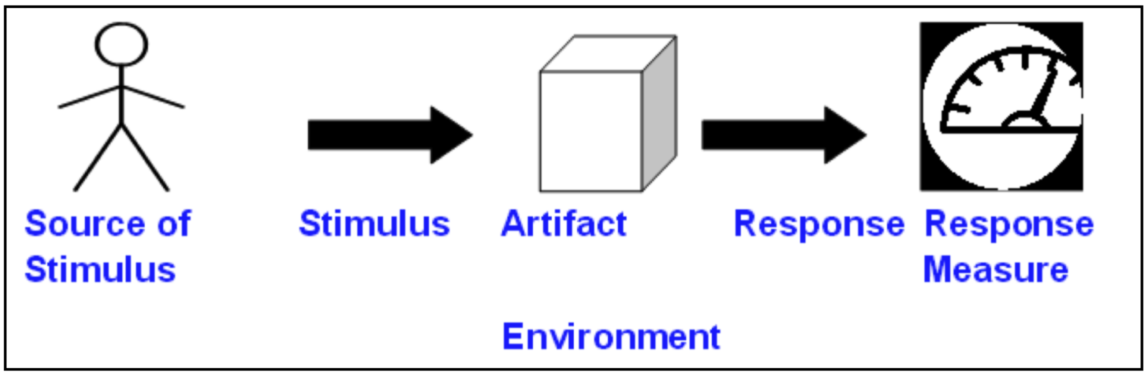


Figure 2 - Quality Scenario

Quality Scenarios briefly describe how software responds to specific conditions that demonstrate one or more system qualities. We use the term system quality (or system quality attribute) to mean a non-functional characteristic of a system. These are sometimes called the "ilities" after their suffix.

Here are some common qualities you may wish to write quality scenarios for:

Performance—the responsiveness of the software.

Availability—the time that the system is up and running correctly; the length of time between failures, or the length of time needed to resume operation after a failure.

Modifiability—the ability to make changes quickly and cost effectively.

Portability—the ability of the system to run under different computing environments.

Usability—the ease of use or ease of training users to interact with the system to accomplish a task.

Security—the ability to resist unauthorized attempts at using or modifying the system.

You can write quality scenarios to describe desired qualities following a general pattern 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).

Additionally, give each scenario a meaningful descriptive name.

Here are two quality scenarios that demonstrate reliability expectations for a forest management software system. The software predicts fire danger using historical data and current weather reported by sensors. Different scenarios about the same quality can be written to show how the system should behave under slightly different conditions.

Reliability Quality Scenario:
Predicting Fire Danger when < 80% sensors report.

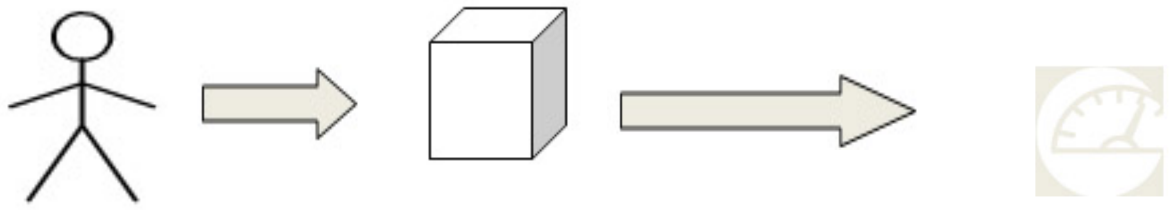
The forest ranger requests a fire danger prediction for the entire forest, specifying the amount of historical sensor reports to be used. The system will return a prediction for fire danger of low, medium, high, or extreme along with a low confidence rating since less than 80% of the sensors have been reporting regularly.

Reliability Quality Scenario:
Predicting Fire Danger when 80% or more sensors report.

The forest ranger requests a fire danger prediction for the entire forest, specifying the amount of historical sensor reports to be used. The system will return a prediction for “fire danger” of low, medium, high, or extreme, along with a confidence rating of “high” when 80% or more of the sensors have been reporting regularly.

Example of Quality Scenario:

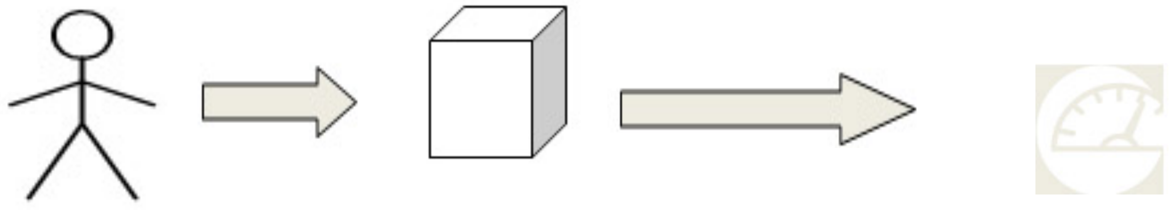
MAKING A PREDICTION WITH < 80% SENSORS REPORTING



Source of Stimulus	*Stimulus*	*Artifact*	*Response*	*Response Measure*
User	Fire Danger Prediction Request	Prediction Fire Danger Analyzer & Sensor DB	Rating Report generated with low confidence level	Report Stored and Processed

Environment: Intermittent sensor reporting

Example of Quality Scenario:
MAKING A PREDICTION WITH 80% OR > SENSORS REPORTING



Source of Stimulus	*Stimulus*	*Artifact*	*Response*	*Response Measure*
User	Fire Danger Prediction Request	Prediction Fire Danger Analyzer & Sensor DB	Rating Report generated with good confidence level	Report Stored and Processed

Environment: Intermittent sensor reporting

Quality Stories

"If you don't care about quality, you can meet any other requirement." – Gerald M. Weinberg

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 features. At other times you may have a specific system quality (e.g., increase ETL accuracy by 1%) 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?

❖ ❖ ❖

You want to be able to plan for, track and manage the work required as outlined by functional user stories on your backlog. While working on these functional requirements, certain qualities are needed to complete the implementation or can become known and sometimes involve multiple stories.

The primary focus during sprints is to focus on functional requirements and there is often not a lot of time given during estimation for system qualities. Also, some qualities and how they affect different parts of the system do not become apparent until the implementation of certain functional requirements.

Although *Quality Scenarios* are useful to understand and details the importance of important system qualities, sometimes they are either too high level or not focused enough to prioritize for the team to help know when they are "done."

❖ ❖ ❖

Therefore, create separate quality stories and add these to your backlog. An agile user story is a short, brief description of a desired feature, told from the perspective of the person who desires that capability. They focus on important user functionality usually prioritized by product owners. While working on different parts of the system or when important qualities need to be prioritized and included in the backlog, create a quality story to represent the system qualities. In contrast to normal agile user stories, a quality story is a short, brief description of some aspect of system quality that is important to achieve. 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.

For example, consider a system that supports payment processing using credit card, banking, and PayPal transactions. Initially, you defined several user stories related to different payment methods ("as a user I want to pay for my order using my bank account," "as a user I want to pay for my order using a credit card," "as a user I want to pay for my order using my PayPal account," etc.) and implemented the functionality to support them. Each of these user stories has its own acceptance criteria and tests. Now you would like to define a performance requirement that spans all types of payment processing transactions.

You could go back to your original user stories and attach specific performance objective acceptance criteria to each. But that doesn't give you the overall performance picture you need. Also, you are willing to accept variations in performance among different payment methods as long as your aggregate performance target is achieved. So instead of attaching

Fold-Out Qualities to specific stories, you create a separate quality story that represents your overall performance target:

"The system should able to handle x number of payment processing transactions per hour under peak operation."

When you need to describe qualities that apply to specific functional user stories, you can always attach quality acceptance criteria to those user stories (see *Fold-Out Qualities*). Thus in addition to the broader performance requirement that applies across different payment methods, you can always define specific performance acceptance criteria for a specific payment method, if you so desire.

Fold-Out Qualities

"Now you know the rest of the story"—Paul Harvey

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 (e.g. "The user can choose to pay by credit card" instead of, "The user can select to pay by credit card by clicking on a radio button."). **But how can you define and describe agreed upon system qualities that should be exhibited by an implemented story?**

❖ ❖ ❖

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.

The primary focus during sprints is to focus on functional requirements and there is often not a lot of time given during estimation for system qualities. Also, some qualities and how they affect different parts of the system do not become apparent until the implementation of certain functional requirements.

❖ ❖ ❖

Therefore, in these situations, create and attach specific quality acceptance criteria to the user story. We call these 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. These are often important qualities that help describe the definition of done.

For example, to satisfy the story "as a user I want to pay for my order using a credit card," you might specify a foldout performance quality that you expect, "the system to be able to handle 100,000 VISA credit card payment transactions per minute."

To uncover additional acceptance criteria related to system qualities desired for this story, you can to ask a number of quality-related questions:

- Usability: Can I cancel an order placed using a credit card? If so, when?
- Security: Does the system retain my credit information? If so, can I control how that information is retained?
- Security: Is my credit information protected from unauthorized access and securely transmitted?
- Performance: How fast can I place an order and receive confirmation? When there are lots of users?
- Availability: What happens if the credit card service is unavailable?

Answering these questions commonly leads to quality acceptance criteria that are attached directly attached to the story, such as the requirement for securely transmitting secure credit information or meeting specific performance objectives. Sometimes more broadly applicable qualities might be identified at the same time. For example, not only should credit card information be securely transmitted, but all personal or financial information should be as well. In this case, in addition to attaching specific quality acceptance criteria to this story, you might also identify several quality-related stories that are added to your backlog as well as write a few specific quality scenarios.

Sometimes, by asking quality-related questions, even new functionality may be identified. For example, usability concerns about placing orders can lead to identifying the need to cancel and track orders. In this case, new user stories can be written about canceling and tracking orders can be written and added to the backlog.

Making Qualities Visible

It is useful for team members to be aware of important system qualities and have them readily available. This can be done through quality radiators, similar to what Alistair Cockburn describes in making any information radiator—visible tangible things that keep people's attention. Quality radiators, just like other information radiators need to change and get adjusted and have new and changing information otherwise they become wallpaper [Co]. The following patlets outline ways to make qualities visible:

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

Our goal is to describe all these patlets as patterns in future papers. We include them here for understanding of the big picture when becoming more agile at quality.

Becoming Agile at Quality

Agile software development is an iterative and incremental development process. The software evolves and adapts to changing requirements. Self-organizing, cross-functional teams perform the work. Most agile processes embrace quick responses to change. The ability to change and adapt is accomplished through short sprints with flexible planning, short delivery and extensive feedback. Agile processes focus on prioritizing the most important requirements and elaborating on those requirements just in time.

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.

For small teams, including a QA expert as part of the team can seem natural and fit into the organization without too much pandemonium. However, this might not scale well for larger projects that require more and larger interactive teams; i.e. 6 Scrum teams doing a scrum-of-scrums to deliver an enterprise application. The following patlets support Becoming Agile at Quality:

Patlet Name	**Description**
Whole Team	Involve QA early on and make QA part of the whole team.
Quality Focused Sprints	Focus on your software's non-functional qualities by devoting a sprint to measuring and improving one or more of your system's qualities.
QA Product Champion	QA works from the start understanding the customer requirements. A QA person will collaborate closely with the Product owner pointing out important Qualities that can be included in the product backlog and also work to make these qualities visible and explicit to team members.
Agile Quality Specialist	QA provides experience to agile teams by outlining and creating specific test strategies for validating and monitoring important system qualities.
Monitor Qualities	QA specifies ways to monitor and validate system qualities.
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.

Whole-Team

"Teamwork makes the dream work." —Bang Gae

"The way a team plays as a whole determines its success." —Babe Ruth

Traditionally QA 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. And 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?

❖ ❖ ❖

Wanting to make sure systems qualities are not biased is very important.

Often there are limited resources and people dedicated to QA.

QA people have a lot of experience understanding qualities and testing issues.

QA can be seen as the enemy just looking to find defects rather than helping the team.

This is a big loss to the team if this expertise and mindset is not utilized until late in the software process.

❖ ❖ ❖

Therefore it is important in Agile Quality Teams to 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. A lot of value is added when QA is part of team from the start. They can help to build quality into system throughout the entire development process. 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 specify system qualities best fits into the process (when to do what for different qualities).

Quite often the way QA becomes integrated with an agile team is to assign a QA person specifically to the team. This QA person will be part of the daily standups, meet with the product owner, support the team with testing efforts, and help identify important qualities and help create a *Quality Roadmap*.

Quality-Focused Sprint

"Quality is not an act, it is a habit."—Aristotle

Features don't make a viable system; rather a viable system is accomplished by focusing on features accompanied by paying attention to system qualities.

You have concentrated on implementing functionality. You are delivering working software each sprint. But you are worried that it doesn't 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?**

❖ ❖ ❖

Prioritizing and implementing the necessary functionality keeps the project moving forward and yields positive feedback from the customer. However, just focusing in functionalities doesn't produce a system that is good enough to be released with specifically if there are important security, performance and other qualities that have not been addressed yet.

On the other hand, focusing too much on certain non-functional requirements can cause some premature abstraction and optimization. It can be hard to know what system qualities should be focused on during every sprint.

❖ ❖ ❖

Therefore, 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.

If your focus is on performance, then the goal of your sprint should be to identify specific areas to improve. 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.

Depending on what qualities you are working on you perform different tasks. And some of these tasks will be easier to estimate than others.

If you are concerned about performance, you will want to measure current performance before tuning critical parts of your system. Although the exact level of performance improvements can be hard to predict, you still need to break your quality stories into estimable tasks such as measuring current performance, load testing, analyzing system hotspots and design rework.

Improving one quality can impact other system qualities. Implementing usability improvements may mean that you revise user-system interactions and rework system APIs. Also, it may not be clear what is a "better" user interaction approach until you perform usability experiments or a/b testing.

Thus, the definition of "done" for a *Quality 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 or *Agile Landing Zone*.

Summary

This paper outlined core patlets to be considered for transitioning from traditional Quality Assurance (QA) to Agile Quality (AQ). This includes both ways of incorporating QA into the agile process as well as an agile means to describe and validate important system qualities. A few of the patlets were described in this paper using the patterns 3.0 format. Ultimately it is the authors plan to write all of these patlets into patterns and weave them into a pattern language to help with becoming more Agile at Quality.

Acknowledgements

We thank our shepherd Hironori Washizaki for his valuable comments and feedback during the AsianPLoP 2014 shepherding process. We also thank our 2014 AsianPLoP Writers Workshop Group, Taichi Isaku, Kiro Harada, Masashi Kadoya, Lei Wang, and Kenichiro Ata, for their valuable comments.

References

[BCK] Bass, Len, Clements, Paul and Kazman, Rick, *Software Architecture in Practice (2nd Edition),* Addison-Wesley, 2003.

[CG] Crispin, Linda and Gregory, Janet, *Agile Testing: A Practical Guide for Testers and Agile Teams*, Addison-Wesley, 2009.

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

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

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

[MR] Manns, Mary Lynn and Rising, Linda, *Fearless Change: Patterns for Introducing New Ideas,* Addison-Wesley, 2005.