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???

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

Abstract. *As organizations transition from waterfall 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 21 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
D.1.5 [**Programming Techniques**]: NEED TO ADD HERE

General Terms
Agile, Quality Assurance, Patterns, Testing

Keywords
Agile Quality, Quality Assurance, Testing

Introduction

As organizations transition from waterfall to agile processes, the role of Quality Assurance (QA) needs to evolve. QA in waterfall could be involved early in the process but in general this is not the case. Traditionally, QA people only became involved late in the development process, just before it was necessary to test and release the final product. This is primarily because of a different mindset between QA in Waterfall and Agile QA. Generally, in waterfall, QA is primarily responsible to certify the functionality of the application based upon the contract and requirements; usually with black-box tests. Most waterfall QA groups work independently from the software team. However, in Agile, QA works closely with the team integrated during the day to day development.

Not focusing on testing early enough could 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 were considered and addressed during earlier sprints, significant architectural verification could have been incorporated into the process much earlier, preventing significant disruptions or delays. 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 need to be more proactive and need to work to ensure quality at all levels of the development process. They also need to work closely and coordinate between business, management and developers. Agile QA teams require additional skills to that of a normal waterfall QA team. For example, they 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. 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 team from the start can help build quality into system and make attention to quality a 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 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. We break the Agile Quality patterns into two categories: how to identify system qualities and make these qualities visible, and ways to be agile at quality assurance. This paper will outline 21 patlets; 11 in the identifying system qualities category and 10 in the ways to be agile at quality assurance category. A patlet is a brief description of a pattern, usually one or two sentences. We take a six of these patlets and write them as patterns for this paper. The patterns outlined in this paper are: Integrating Quality into your Agile Process, Agile Quality Scenarios, Quality Acceptance Stories, Fold-Out Qualities, Whole Team and Quality Focused Sprint. The patterns are written using a modified version of Takashi Iba's Patterns 3.0 format [ref]. Our ultimate goal is to turn all patlets into full-fledged patterns.

Identify Qualities and Making them Visible

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. It is important in agile teams to identify important qualities and make those qualities visible to the team. This can be done through quality radiators, similar to what Alistair Cockburn describes in making information radiators—visible tangible things that keep people's attention. Quality radiators, just like other information radiators need to change and get adjusted and have new / changing information otherwise they become wallpaper [ref]. The following patlets support Identifying Qualities and Making them Visible:

Patlet Name	**Description**
Integrating Quality Into your Agile Process	Incorporate QA into your process including a lightweight means for describing and understanding system qualities.
Agile Quality Scenarios	Create high-level quality scenarios to examine and understand the important qualities of the system.
Quality Acceptance Stories	Create quality acceptance stories that in their own right elaborate specific conditions of the system including what be measured or verified and what constitutes "success".
Fold-out Qualities	Define specific quality criteria and attach it to a user story when specific, measurable qualities are required for that specific functionality.
Qualify the Roadmap	Examine a product feature roadmap to plan for when system qualities should be delivered.
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.
Quality Backlogs	Create quality scenarios that can be prioritized on a backlog for possible inclusion during sprints.
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.
Identify threshold values	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.
System Quality Dashboard	Define a dashboard which 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.

Integrating Quality into your Agile Process

"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. Quality Attributes 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.

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.

❖ ❖ ❖

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 Acceptance Stories to help with 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 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 Acceptance 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.

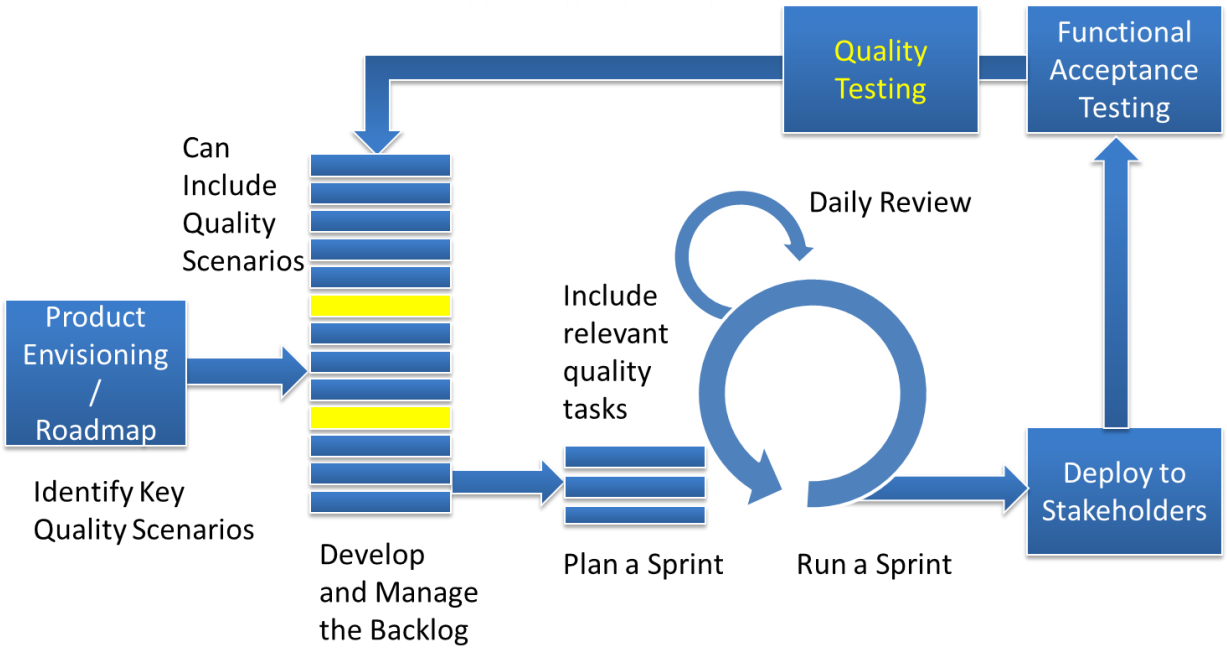


Figure 1 - Quality in Scrum

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 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 on a product backlog that just 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?

❖ ❖ ❖

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.

❖ ❖ ❖

Therefore, early on in the process create high-level quality scenarios that address important non-functional requirements such as performance, load, reliability, security, etc. If we 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.

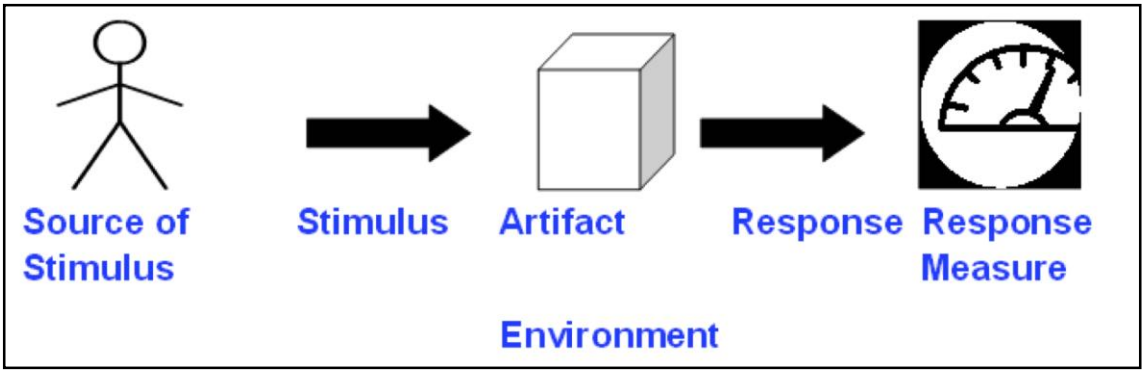


Figure 2 - Quality Scenario

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 six 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 part of the system under what operation conditions), the *response* (what happens when the system reacts), and the *response measure* (some concrete, tangible result you expect).

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.

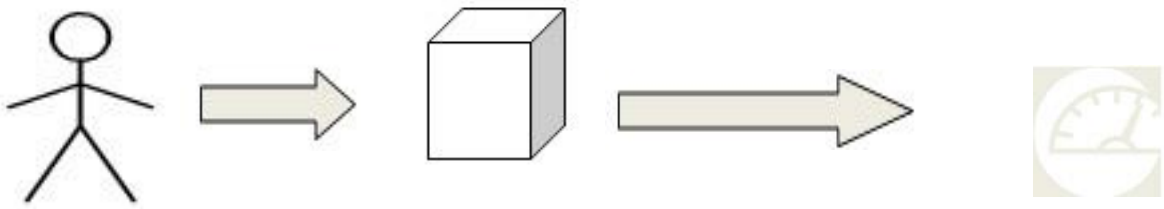
Reliability Quality Scenario:
Predicting Fire Danger when < 80% sensors report.

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.

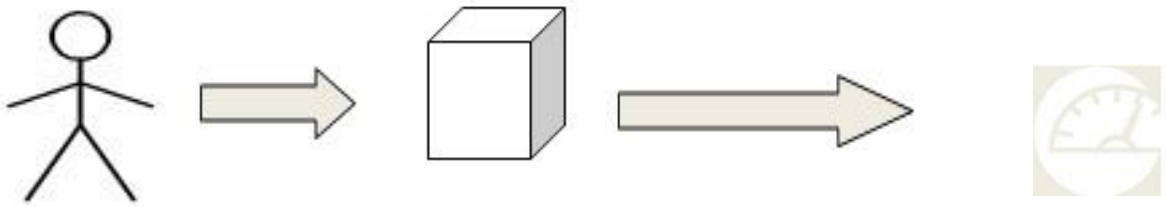
Example of Quality Scenario:
MAKING A PREDICTION WITH < 80% SENSORS REPORTING



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 Acceptance Stories

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

A user story or feature is considered shippable when it meets the expectations of a product owner and is of the agreed quality. 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 quality?

Sometimes you need to describe acceptance criteria for performance, usability, internationalization, reliability or other non-functional qualities that apply to specific stories, e.g. (e.g. you expect the system to be able to handle n credit card payment transactions per time unit) or (95% of first time users should be able to select the credit card payment option without using the built-in help function). If so, then simply quantify and attach quality acceptance criteria to specific user stories (see *Fold-Out Qualities* for how to describe story-specific qualities).

At other times you will need to specify acceptance criteria for system qualities that broadly apply to many stories or features (e.g. you expect the system to be able to handle x number of financial transactions including credit and bank authorizations, PayPal per hour under peak operation). In this case, write quality acceptance stories that are "attached" to a specific group of features, or apply to the system as a whole.

Therefore, create separate quality acceptance stories and add them to your backlog because in their own right there may be a need to elaborate specific conditions of the system, describing what is be measured or verified and what constitutes "success".

Prioritize these stories during your sprints along with other functional stories and create tests and dashboards to analyze and validate these qualities.

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 is of the agreed quality. 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?

Sometimes you will want to specify acceptance criteria for system qualities that broadly apply to many stories or features. In this case, write quality acceptance stories that are "attached" to a specific group of features, or apply to the system as a whole (see *Quality Acceptance Stories).*

In addition to working functionality, some stories have explicit quality-related goals that are part of accepting this story. In these situations, in order for a story to be of acceptable quality it must meet specific performance, usability, internationalization, reliability or other non-functional requirements.

Therefore, in these situations, create and attach specific quality acceptance criteria to a specific user story. We call these fold-out qualities because they are integral to accepting the story. While your initial concern is correctly implementing that functionality, satisfying a fold-out quality can strongly influence your design and implementation.

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

Or, to satisfy the user story, "As a user I want to be able to specify a credit card to be used as payment," you may specify specific usability criteria, such as, "95% of first time users should be able to select the credit card payment option without using the built-in help function".

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.
Monitoring 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.
Recalibrate the Landing Zone	Readjust landing zone values based on ongoing measurements and benchmarks.
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 througout 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 a developer 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 like 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 prefered to test a fully functioning system in order to minimize re-testing and rework. And since QA typically has not been engaged until of 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 integrating 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; features accompanied by attention to system qualities do.

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 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 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 as 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 for transitioning from Quality Assurance to Agile 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, YYY, for their valuable comments.

Note to Shepherd and Workshop Reviewers

Our references have not been completed yet as we are working on these for the final paper. We will go through all our references and update them. The following section is a place holder for when we complete this task.

References

[EIDE] http://www.eclipse.org/

[EEQX] http://www.eclipse.org/equinox/

[FCW08] Ferreira, H. S., Correia, F. F., and Welicki, L. 2008. *Patterns for data and metadata evolution in adaptive object-models*. Proceedings of the 15th Conference on Pattern Languages of Programs (Nashville, Tennessee, October 18 - 20, 2008). PLoP '08, vol. 477. ACM, New York, NY, 1-9.

[SEL] Spring Expression Language. http://www.springsource.org/.

[YR00] Yoder, J.; R. Razavi. *Metadata and Adaptive Object-Models*. ECOOP Workshops (ECOOP 2000), Cannes, France, 2000.