Agile and QA

The Basics

Version	2010-05-13

Version History

Date	Author	Changes	Modified Sections
2008-11-10	Manuel Cubillo	Initial Writing	All
2008-12-01	Manuel Cubillo	New sections added	All
2008-12-17	Manuel Cubillo	Add Juan Carlos' recommendations Complete pending sections	All
2008-12-24	Manuel Cubillo	Add Gerardo's recommendations Format	All
2010-05-13	Manuel Cubillo	Adding new information based on recent experience	All

AVANTICA TECHNOLOGIES
The Nearshore Advantage!

CONTENTS

1 Introduction

1.1 Purpose

Software development methodologies (and how good the results we can obtain are when using one or the other) are a common subject of discussion. There is always a group of people trying to improve the existent methodologies to get best results, while others just stick to what has been typically used.

Agile software development is one of the newest methodologies being used these days and also one that generates a lot of discussion within the software industry, due to the differences between its practices and the traditional ones.

Along with all the changes introduced by Agile from the developer's point of view, there's many things to keep in mind from QA's perspective and that's exactly what this document is about.

1.2 Audience

Target audience for this document is Avantica's QA, management and development personnel participating in projects following the Agile approach, as well as any other people in Avantica interested in QA activities from an Agile perspective

1.3 Scope

Outline the basic principles and myths about the agile methodologies (in general terms), as well as describe the role, tasks, responsibilities, concerns and keys to success when performing QA activities in Agile projects.

1.4 Terminology

Term	Definition

1.5 Abbreviations

Abbreviation	Definition

2 History

Agile software development refers to a group of software development methodologies that promote:

- A project management process that encourages frequent inspection and adaptation
- A leadership philosophy that encourages team work, self-organization and accountability
- A set of engineering best practices that allow for rapid delivery of high-quality software
- A business approach that aligns development with customer needs and company goals.

According to Wikipedia[1], the modern definition of agile software development evolved in the mid-1990s as part of a reaction against "heavyweight" methods, perceived to be typified by a heavily regulated, regimented, micro-managed use of the waterfall model of development.

In 2001, prominent members of the community met at Snowbird, Utah, and adopted the name "agile methods." Later, some of these people formed The Agile Alliance[2], a non-profit organization that promotes agile development

Some of the better known agile methods include Scrum (1986) and Extreme Programming (1996). These are typically referred as Agile Methodologies[3] since the Agile Manifesto was published in 2001.

[1] Wikipedia. http://www.wikipedia.com/
[2] The Agile Alliance. http://www.agilealliance.com/
[3] The Agile Manifesto. http://agilemanifesto.org/

3 Agile Principles

Before we start talking about QA in Agile, it is very important to know the basic principles behind the whole Agile concept. Agile methodologies try to avoid all those "tedious" tasks typically performed (when the development process follows the traditional schema) and focus on people and results instead.

It's important to mention that Agile refers to a big concept that will work as a base for multiple "implementations" such as SCRUM or eXtreme Programming (XP). Given this, there's a set of practices used in some of these implementations that are not part of the "original" Agile principles (i.e. Pair Programming, Test Driven Development) but are commonly mentioned when it comes to talk about Agile.

The Agile objectives are accomplished by sticking to the principles and commonly used practices described on this section.

3.1 The Agile Software Manifesto

In 2001, 17 prominent figures in the field of agile development came together at the Snowbird ski resort in Utah to discuss ways of creating software in a lighter, faster, more people-centric way. They created the Agile Manifesto, widely regarded as the canonical definition of agile development and accompanying agile principles.

The Agile Manifesto states:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***"Individual and Interactions"** over "Processes & Tools"*
- ***"Working software"** over "Comprehensive Documentation"*
- ***"Customer Collaboration"** over "Contract negotiation"*
- ***"Responding to Change"** over "Following a plan"*

That is, while there is value in the items on the right, we value the items on the left more.

This manifesto generated a movement in the software development industry known as the agile software development.

3.2 Short Iterations / Incremental Approach

In the Agile world, one of the main objectives is to deliver working software as frequently as we can, but software requirements are usually big and it frequently takes long periods of time before they are completely done. However, we can always break a big requirement into a set of smaller tasks that can be completed in short periods of time… That's exactly what agile is taking advantage of!

One of the agile principles is to divide the project into small iterations (typically 2 or 3 weeks long) on which small tasks will be executed. At the end of the iteration, a working "slice" or

subset of the functionality is ready and can be delivered to end users. As more iterations finish, more and more functionality is delivered, building the whole piece of software in an incremental way.

3.3 Refactoring

Code refactoring is the process of changing the source code of a computer program to improve it or simplify it, while preserving its existing functionality.

Due to the incremental approach and the multiple changes the software development process faces, code refactoring is very likely to happen and should be one of the main practices to follow when working with the agile methodology.

As a result of refactoring we can obtain simpler code that is easier to read and as a consequence easier to maintain. However, refactoring requires people to be careful so that they don't break working code or enter new bugs in the system as a result of the code changes.

3.4 Empowerment / Ownership

Given the incremental approach, the short iterations and the refactoring tasks (to mention some of the factors), agile methodologies provide people with the power to make important decisions about the product, its quality and the way it should be handled.

Ownership is not as clear as it is in the traditional methodologies, where the one that created a feature is probably the one that will take care of any related future improvements and bug fixes. In agile, typically there's no official code owner and everybody can modify (refactor) someone else's code as long as the original functionality remains the same.

With this kind of "ownership" in agile, the whole development team is very likely to be aware of the way things are implemented and to provide the proper feedback in a timely manner (if necessary). The concept is also applied to most of the activities (other than programming) within the software development process, giving people certain level of power and ownership of the whole product, not only a part of it.

3.5 An Alternative Requirement Description: User stories

One of the main objectives behind the agile methodology is to always satisfy user needs. The success of a software product will be determined, at last, by its final users and that's why we need to fulfill their needs.

Traditional methodologies collect a set of requirements early on the process (probably following a structured system) and once we have "all that's needed" we start developing what was described. However, once we deliver the final product we usually notice that users needs have changed and the product we created no longer fulfill those needs. So, the software is not that helpful for end users any more.

Agile methodologies try to avoid the problems mentioned by setting a whole new way to define user requirements: User stories.

A user story is a software system requirement formulated as one or two sentences in the everyday or business language of the user. User stories are a quick way of handling customer requirements without having to elaborate vast formalized requirement documents and without performing overloaded administrative tasks related to maintaining them.

A user story is an informal statement of the requirement as long as the correspondence of acceptance testing procedures is lacking. Before a user story is implemented, an appropriate acceptance procedure (or acceptance criteria) must be written by the customer so that it can be used by QA to determine whether the goals of the user story have been fulfilled

3.6 Pair Working

Typically referred to as “pair programming” (since programming is usually seen as the main practice within the software development process), this is one of the best practices suggested by the XP implementation of Agile and one of the most typically used.

Pair working refers to a practice where 2 people work together at one keyboard. One person types in while the other reviews everything that’s been done. The person typing is called *the driver* and the one reviewing is called *the observer* or *the navigator*.

It is important to mention that the observer role is not passive at all. Both the driver and the observer are constantly interacting and working together to get the best results on what they are doing.

4 Agile Myths

Although agile methodologies are now well known in the software industry, there still are some myths around that make people reluctant when it comes to decide about start using them. The following are some of the most commonly mentioned myths about agile:

4.1 No documentation

One of the most commonly heard myths about agile has to do with documentation. People usually think there is no space for documentation when following the agile approach… Completely wrong! Documentation is actually created as part of the agile methodology, but things will be documented only when there's value in doing it.

Creating and storing a bunch of documents (whether printed or digital) that no one will ever read is seen as a waste of time in the agile world. Once again, there will be space for documentation but only when it brings value to the process (real value!).

Usually, documentation is created in response to a business request or when we can identify a big group of people requiring it. Also, since time is one of the most valuable assets in the process, an efficient way to create and publish documents is required (Wiki pages is one the most used options).

4.2 No design

The first time one hears about the agile methodologies and the way they work, a common first impression is that design is one if the "sacrificed" steps in the process, in order to get things working as soon as possible.

Agile methodologies do include design, but the design process is a bit different from what we are used to:

- Agile designs are emergent, they are not defined up front: Given the incremental approach, the product we are creating will evolve as the time goes and so design for the new modules will be needed to fulfill the new requirements.

- Models have to be "good enough": The designs created as part of the agile methodology should not describe each and every detail. It has been proven that most of the "detailed designs" need to be changed and improved once the actual coding starts.

- Feedback is always needed: In the agile world you won't have a single person to design (not even the design for a single module or feature). As in every other activity in agile, a group of people will get together and brainstorm (one or more sessions) until they get to a design that is good enough for the given purpose.
- Document only what's necessary: If the design is complex, document it thoroughly. If not, document only what's necessary to understand it (The trivial parts will be usually skipped by readers)

4.3 No testing

You cannot deliver a product that has not been tested and agile methodologies stick to that reality probably more than any traditional approach. Agile focuses in deliver high quality working software but there are differences in the way software is tested when compared to traditional software development.

We will elaborate this concept in deep on the following sections.

4.4 No planning or idea of progress

Since requirements, designs and changes seem just to "pop up" and the agile team mission is to keep up and respond to all those changes, one can argument that there is no planning involved when working agile.

There's a lot of planning as part of the agile process and most of the team is involved in the planning process. Iteration planning meetings, design and modeling meetings, goals setting meetings and decision making meetings are some of the planning related activities that happen every week.

Planning is not executed as a stage in the very beginning of the development process but as an ongoing activity that constantly provides feedback about where we are and where we should go, setting a realistic idea of progress instead of fuzzy percentages.

4.5 Agile is easy

Given the "freedom" found in agile methodologies, people easily start to think that it is a very easy way of working: "You just need to start working and react to whatever it comes", one might say.

Setting an effective agile process is a very heavy goal to accomplish. It requires a lot of attitude and discipline from the team, very strong technical and people skills and real support from business high levels (since things will not work the same way they are probably used to), to mention some of the keys.

The "freedom" mentioned before can be your best ally, but can also be your worst enemy if it's not properly managed. So, really hard work is required to get things working.

5 The Role of QA in Agile

Given the special care about delivering quality software to the users, QA has a key role in the agile methodologies not only as regular testers but also as an active part of the development team and one the most important feedback providers in the process.

QA people do much more than just running a set of test cases every single iteration. Accordingly to the agile principles described at the beginning of this document, the QA people has the empowerment to make key decisions about the product being developed or the process being followed to create the product.

Unlike traditional approaches, QA is not treated as a separate team in agile. QA is a part of the whole software development team and so the QA people must have the necessary knowledge and experience to manage the quality related issues / concerns of the product being developed.

The following are some of the main functions of QA people within the agile development process (besides just testing):

5.1 Source of Key Information for the Project

QA probably represents the part of the team that should know the most about the product being developed. Since QA takes care of the quality of the product as a whole, they should have strong knowledge of business logic, technical details, end user needs and the current status of the product at every stage of the project.

Such important information should not be ignored if we want to create a product that really fulfills end user needs and that's why, in Agile, QA is actively involved in the whole development process, providing all that valuable information from the very beginning.

Some of the topics where QA can usually be an important source of information are:

- Functional issues
- Product performance
- Data consistency / integrity
- Match of delivered software with real user needs
- Feature improvements
- Delivery decisions (go / no go?)

5.2 Provide insights from user perspectives

End users and their needs are the main reasons for creating any software product. If the product we are creating does not meet end user needs, all the hard work would have been just a waste of time.

Sometimes, developers do things the way they think is better and not the way users want (or like) just because "it still meets what's on the requirements document". This "worse practice" can easily turn into the cause of a failing project if it's not properly handled.

QA people, with all its product and business knowledge, should act as the voice of end users to let developers know what they really need and how they really need it. If the code being generated is not sticking to what end users want, QA should act as a bridge (from users to development team) and let the rest of the team know about the consequences that it can bring and, if possible, provide / help figuring out an alternative to get things into the right way again.

Additionally, QA is can become one of the main communication channels between users and the project developers. In many agile projects, QA people attend meetings where end users expose their needs, what they find valuable on a software product, what are the their main concerns from the business perspective and a lot of additional information that can take the project from fail to success if we pay attention.

5.3 Project's headlights (Where are we now? Where are we going?)

As a software development project is executed, there might be a lot of changes that will affect (for better or worse) the project itself:

- Requirements
- Business logic
- Deadlines
- Priorities for modules or requirements

That kind of changes happen all the time and we need to learn how to deal with them, while lowering the negative impact they may have for the project.

One of the keys to success on software project management is to know what the current status of the project is at all times. When you know exactly where you are, it's easier to determine if things have been executed the right way so far and if it's necessary to change something in order to achieve the final objectives.

Given that QA understands the big picture of the project, they have the responsibility to provide feedback on a timely manner regarding what's the project status from their perspective and what's the next step to take in order to assure that the project will succeed in the end.

At any point of the project, QA must let the rest of the team know about any possible quality mismatch with users or business needs as well as any other issues that can affect the project in a negative way. Furthermore, QA can object the delivery of the product at any time if they consider that it doesn't match what's required by end users and the business itself.

6 Agile QA Principles

Now that we are clear about the role that QA plays in agile, it's important to describe some of the basic principles that lead the QA efforts when following agile methodologies.

6.1 If it's not tested, it's not done

This is probably the most important of the principles for agile QA: No feature or requirement can be considered as done or completed if it has not been tested. QA is responsible of testing every feature that is developed in the project and, once the feature has been tested, QA can accept it or reject it depending on the results obtained.

If a feature is rejected by QA, it should not be delivered to end users and cannot be considered as completed until all the issues detected have been properly fixed, tested and accepted by the QA team

Given this, QA can be seen as the last resort for all the functionality in the project before it is delivered to end users. QA has the power to determine what the right time for the product to go out is and so they have a big share of the responsibility in case that any quality issue reaches the end users.

6.2 Requirements based testing

Since the agile methodologies deliver functionality based on an incremental approach and project's requirements / specifications may change at any time, QA activities cannot be easily set and planned up front.

With this scenario, the QA team should plan the activities to be performed based on the requirements available for testing and the specific testing needs at a given point of time. This means that probably QA will not perform the same set of tests in all the iterations but those that are required depending of the project current status.

It's very likely that a subset of the tests will be performed on every iteration in order to assure that all that has been delivered so far it's still working properly (regression testing / acceptance testing). However, depending on the current needs, a given iteration can be focused on product performance while the next one can focus on integration testing.

The importance of being aware of the current status of the project, business / user needs and the tests that have been already executed (and its results) is bigger in agile. Determining what test strategy to use for a given iteration is not as easy as it seems, it requires a lot of project knowledge and cross functional teams discussion.

6.3 Quality as part of the development process from the beginning

As mentioned before in the document, in agile, QA is not a small activity at the end of the cycle that can be skipped if we are running out of time. The role of QA is critical for project success and that's why it's not a single task but an ongoing activity from the beginning of the development process.

Starting from project outlining and going through user stories definition, testing and acceptance, QA people must be involved in the project at all stages in order to assure that the product being created will meet user and business needs.

QA's job is not only to run test cases and say how many passed or failed. QA must work together with the whole project team, at all stages, sharing information and ideas, providing feedback in a timely manner not only related to the product being created but also related to the process followed to create it.

QA people getting involved late in the project will probably miss a lot of key information that will be needed when decision making processes take place and QA perspective (representing not only QA but end users also) is required to make the right choices.

6.4 Several testing areas involved

When it comes to test, there are a lot of areas that need to be covered if we really want to assure the quality of a software product. Unit testing, regression/integration testing and acceptance testing are only some of those areas.

In agile projects, it is necessary to have all those testing areas clearly identified and assign the necessary resources to take care of each one of them, reducing the risk of leaving the application uncovered at any level. The key for covering all the testing areas is to assign the right people to the right task.

In agile, QA is not the only responsible for product testing. Developers have a big share of the responsibility by automating unit / integration tests.

Additionally, a group of people assigned to define and execute regression or acceptance, load and performance or integration testing (with each one of these areas clearly identified) will collaborate to get the job done at all levels.

6.5 Automation is critical

It's important to keep one thing in mind when following an agile approach: Manual testing is not responsive enough!

As mentioned in previous sections, one of the basic agile principles is working with short iterations and following an incremental approach. It means that every 2 or 3 weeks (even more often sometimes) a new QA cycle starts and we have a small time window to perform all the necessary tests and make sure the product meets the quality and business requirements.

Additionally, with the incremental approach QA gets new functionality every iteration and that functionality needs to be tested thoroughly (functional testing). Besides, QA needs to make sure the new functionality is not breaking any of the features previously delivered (regression testing) and that the application is still working properly as a whole (integration testing).

QA has to do the whole job before the next iteration starts. So, trying to manually run everything might work the first few cycles, but as the time goes and the product gets bigger and bigger, the

manual effort is simply not responsive enough. This is the main reason to start automation as many test cases as we can.

With the automation approach, QA has a way to keep testing the whole set of features previously delivered while taking care of the new stories arriving with every new iteration. It requires a big effort not only for creating the automated test cases but also to maintain them so they still remain valid as the application changes in time.

Automation is the only way for QA to keep up with the growth of the product being certified, keeping the same quality in the job performed and also allowing QA people to be always on track with the current status of the project so they can provide the right feedback in the right moment.

Given this picture, developers can easily become a really valuable ally on the automation effort. The bigger coverage we get from developers (by unit / integration tests) the better for QA and the product.

7 Typical Agile Concerns

There's a series of concerns that typically comes to QA people when they first hear about QA in agile projects. The following is a list of some of the most commonly mentioned concerns and the way agile face every one of them:

- Can QA make it even if there's no documentation? Yes, as mentioned on previous sections, important things will be document. It's just that in agile you won't waste your time documenting unnecessary things.

- Processes not defined: Processes should be well defined in agile and they should be followed by the teams. However, the main difference with traditional methodologies is probably that changes occur more often in agile and people need to adapt faster to follow the new process.

- Not enough time to finish QA cycles: If you are planning to run all of your tests every iteration and most of those test cases are executed manually, then it's very likely that you'll run out of time before you finish the execution. The right approach to follow is to automate as many test cases as possible and prioritize them so that only the most important of the remaining manual test cases are run every time and the rest of them will be run only if needed (depending on the purpose of a given test cycle)

- Hard to estimate: If you are thinking on a one time planning then it's very likely that your estimations will not be accurate. Since agile projects change a lot every iteration, you will probably have to estimate one iteration at a time and it should not happen way ahead in time, otherwise you might get frustrated as you notice your estimations don't fit reality at all.

- QA out of sync with development: This should not happen as long as the communication channels are properly used by both teams. If people effectively use their communication spaces (such as meetings), they will always be aware of what the other teams are working on and what the current status of the product is. Remember: Both QA and developers are part of the same team!

- Typical practices don't fit: If you are planning to use typically used practices "as they are" (without any modifications) then this will be true. It's not that typical practices are not worth in agile, but they will need some tweaks to get the most of them.

- Product never finished: At the end of each iteration you won't really get a final version of the product, in the sense that probably not all of the modules will be already present. However, for the included modules you should get a completely functional version, meaning that you should not get "half-baked" features.

- TDD = No QA?: Not true, TDD is just a way for developers to make sure their code is less likely to fail. However, there will always be a lot of issues that won't be discovered by TDD (or unit testing or any other developer tests) and require QA's work to be found.

- No big QA teams: Agile requires people to be capable to perform tasks from multiple fields, so QA people should be able to automate or run manual / load and performance / other tests at any time. This situation ends up reducing the size of the teams since the whole job can be accomplished with less people due to that flexibility. This does not mean that, if necessary, a big QA team can't exist on Agile projects.

- Always in a rush: Depending on the way things are prioritized / planned the QA team will be always on a rush or just busy doing the required job. Certainly, there are a lot of activities to be performed within every QA cycle, but as long as things are properly planning the team should not go into a rush.

8 Agile QA Tasks and Responsibilities

On agile, responsibilities for QA people changes a little from what is usually required on traditional approaches. This section describes the most important tasks and responsibilities to be performed on a regular agile QA cycle.

8.1 Story definition/acceptance (validate acceptance criteria)

Requirements are presented as user stories in the agile world. Since the base for any comparison between what's expected by users and the actual product released is what's written in user stories, QA must be involved not only in the acceptance of user stories (given some criteria) but also in the process to define those stories.

A user story will be the main reference for QA to determine whether a given feature (or set of features) meets end user needs, so QA people must assure that user stories provide all the necessary information to validate a given functionality or change in the product.

It is important to mention that, given the incremental approach followed by agile, it is hard to accomplish "perfect" user stories since its very creation, basically because it's very likely that people involved in the story definition might still miss some implementation details that will get clear as the time goes and some code is written.

QA must make sure that at least the final objective of the user story is clearly defined from the very beginning and the general behavior of the feature is described with enough details.

Additionally, QA is responsible to accept or reject user stories once these are completed (and delivered). That's why they need to make sure acceptance criteria was properly set in the first place, otherwise the acceptance process wouldn't be accurate and QA wouldn't have the necessary information to say (for sure) the software implementation really meets end user needs.

8.2 Test case definition

Test case definition is one of the typical activities for QA to perform and we will not elaborate on how to do it. However, one will notice that test case creation process works a bit different than usual in the agile world.

Traditional approaches have a design stage where use cases for the entire application are defined and outlined. After design, all the test cases for the product can be properly created based on the use cases now available and, once the product is finished, they will be executed to find any existent issues.

Since agile doesn't believe in a big design stage at the beginning of the process, the traditional approach just doesn't fit and QA people must follow a different approach: Ongoing test case definition

As the development process goes, new feature stories will be created and developed, and QA must keep up the pace and create test cases to cover all the new functionality. Also, corner

cases discovered during manual testing and (sometimes) reflected as bug reports must be documented as test cases to make sure the same issue will be tested in future QA cycles.

8.3 Automation

Automation is one of the key activities within the whole agile QA methodology. Given the many tasks QA people should execute every iteration, it's very important to cover as much functionality as possible with automated test cases.

Having good automated coverage allows QA people to focus their efforts on different areas such as load and performance, metrics, test case creation or process improvements, in order to add value and quality not only to the product but to the project itself.

Sometimes it's hard to find time to spend in automation, but it's necessary that some resources or at least part of their time can be spent on automating test cases.

Automation should be planned since the very beginning of the project and performed on a regular basis to obtain the expected results early on the process.

Additionally, QA should encourage developers to automate some tests on their end (by unit tests for example) and make sure that, whenever they say they are doing it, they are actually doing it.

8.4 Manual testing

Even though automation is one of the keys to get real agile QA, manual testing cannot (and must not) be avoided. With the beginning of any QA cycle, automation tests will be run and that way QA will have a general idea of the current status of the product (based on the obtained results), but there's additional work to do after that.

There are certain tests that, due to its specific objective, cannot be automated easily (at all sometimes). Besides, manual testers will always have better criteria to find bugs than any automated tool, so manual testing will always be needed to find non trivial bugs and catch all those details typically missed by automated test cases.

8.5 Bug reporting/verification

This is not very different from what QA usually do on typical development projects: As the testing cycle goes, new issues related to functionality, performance or feature requests will be found and it is QA's responsibility to report them on a timely manner.

The information required for a bug report is pretty much the same typically used:

- Summary
- Related component
- Priority / Severity
- Report type (Bug / Feature Request)
- Environment information (OS / DB / Server / Browser / …)
- Steps to reproduce
- Actual Results

- Expected Results
- Additional info / comments (If needed: Screenshots / logs / …)

Additionally to what's been mentioned, there are some additional details to include:

- *Reference to User Story*: Everything in agile is treated like a story and bugs are not the exception. The new story for the bug reported should include the bug summary as the summary of the story and a reference to the bug report in the story description.
- *Bug Type*: To distinguish between new issues and regression issues. This is very important information that can be used to analyze and detect trends in time.

There's more than just reporting when it comes to bugs, we should also verify they are fixed. This is probably once of the bigger differences between traditional approaches and agile: Bug verification is an ongoing activity.

On traditional approaches, a bunch of fixed bug reports will be accumulated and, once we get a build likely to be released to end users, a bug verification cycle will start to make sure everything was actually fixed.

In agile, new issues will be found by QA and old issues will be fixed by the development teams on every cycle. So, one of the ongoing QA tasks will be to always verify all issues that have been fixed and delivered.

Whenever QA finds issues still not working as expected, the bug report will be reopened and the related story should be rejected, so that developers notice the situation and the problem can be addressed within the next few iterations.

The main purpose of this ongoing activity is to always address the issues as soon as possible and, along with that, provide as higher quality as possible to end users.

Aside to this ongoing verification, there will always be some bug reports left behind due to low priorities or changes in the focus of the project. QA people must keep an eye in all those old reports that have not been addressed and raise the flag once in a while, asking for development to resolve the issues.

Sometimes the resolution will be to get the actual fix for the issue, sometimes it will be just to get the issue marked as "Will not fix" or "Invalid". Despite the resolution (higher levels will determine if it's right or not) QA should not allow issues to be left behind without any decision being made for them.

8.6 Meetings Involvement

One of the most important things in agile is people involvement in the development process. Many meetings are held (all of them with a very specific purpose) and people from the team areas should participate on them.

Some of the most commonly held meetings are:

- Stand up meetings (every day): Very short meetings – 10 to 15 minutes – where each team member let the rest of the team know about what he/she performed the day before,

what's the current status (outstanding findings, issues, …) and what's to be done that day.

- Iteration planning meetings (every 1 or 2 weeks): Usually a 1 hour meeting where all the activities to be performed during the current/next iteration (or week) are planned / outlined and outstanding issues are analyzed. Details added to user stories will probably need to be improved / extended after the meeting.
- Retrospectives (every few months): Intended to determine, for the last few months, what was ok – to keep doing it -, what was wrong – to stop doing it – and things we are not sure about so we can determine if it's worth it to keep doing.
- Bug Triage (every few weeks): To determine priorities or resolutions for reported bugs.
- Post-mortem meetings: Sometimes, there are situations on the development process that don't go the way we expected them to go and the team has to change plans in order to solve the issues found. When this kind of things happen, we can just leave them behind or we can sit and analyze what went wrong and how we can avoid it. Post mortem meeting are meant to do that analysis and, in Agile, QA is a key participant and has valuable things to share.

In most of these meeting we will find participants from development, QA, project management and even end users and the idea is to provide all of them with the necessary information to understand, not only the business logic involved in the project but also the current status of the work being performed by all related teams, so people can perform their own job better.

Demos for new features and Tech Talks about specific subjects related to the project are examples of other meetings held to keep people aware of what's being done in the project and how it's being accomplished.

8.7 Process and Quality Feedback

One of the most important QA responsibilities is to constantly provide feedback not only related to the quality of the product but also to the process being followed to create that product. One probably would think it's obvious that a QA resource should communicate all of his/her findings but in practice it's not always true.

Issues related to quality are very likely to be reported by QA people since it's perceived as the purpose of their job. However, when it comes to report issues related to the process the next step is not that clear.

A very important component of product's quality is the process followed to create it. If that process is designed to allow "quality holes" in the product and those "holes" are detected, they should be reported as soon as possible so the proper actions to correct it can be taken.

Many things can be seen as process issues depending on the project stage and purpose. Some of the most common are:

- People not following the defined process
- QA people not having enough time to test the product
- Bug reports not being resolved within specified time frame
- Too many builds delivered too frequently, so QA cycles are not fully executed

- People using testing environments for the wrong purpose (i.e. Functional testing server used for Load and performance tests ends up in wrong performance results)

The issues mentioned above are only part of what we can find in practice. Providing feedback in a timely manner is a key practice to guarantee the quality of the product in the long term and will also provide a way to improve the process as project's needs change.

8.8 Visibility

Quality can be easily considered as a subjective "feeling" if solid bases are not provided to support our perception on whether we have a quality product or not, and one of the better strategies we can choose to support our quality statements is metrics and indicators.

Most of the activities performed by the QA team can be broken down in simple numbers (metrics and indicator) that reflect the status of the product in a more comprehensive and summarized way that supports QA's perception of quality.

By providing indicators and comparing them with specific metrics, the whole development team can easily determine how far/close they are from the expected quality level and the actions that need to be taken in order to reach that level.

Additionally, when QA provides solid bases to support their opinion they obtain better credibility from other areas, and that credibility really helps when it comes to say that a product should / should not be delivered based on its quality (previously determined by the a QA cycle)

It's important to mention that it's not a matter of how many metrics and indicators we create and show to the rest of the team, but a matter of how much those metrics and indicators really tell about the product and its current status.

9 Agile QA Keys to Success

Software development methodologies traditionally used are very different from agile and that means that typical “best practices” might not fit agile needs. This section presents keys to success when it comes to start doing QA in agile.

9.1 Tools

Agile requires people to focus their efforts effectively. Given the number of activities to be performed on every single test cycle and the amount of information that needs to be collected, QA people requires the proper tools to make the process easier.

In general terms, we can mention the following essential tools:

- Test Case Management Tool
- Bug Tracking Tool
- User Stories Management Tool
- Proper Test Case Automation Tools
- Any necessary tool to access project’s code, databases, etc.

It is desired that all these tools provide ways to relate the information stored across other tools. Some of the relations you’d probably like to have are:

- Bugs related to a given test case
- Bugs related to a given user story
- Markers for those test cases already automated
- User stories related to a given bug
- Manual test case related to a given automated test case

As long as QA has the proper tools to perform all these activities effectively, developers / QA / Management will have easier ways to access relevant information about the process and so they will have a strong base for decision making and process improvement.

9.2 Priorities

Agile methodologies usually require people to complete many activities in short periods of time. As we have mentioned before, sometimes you just don’t have enough time to execute the whole set of activities and you need to have a way to wisely decide what’s to be done (and what’s not).

Priorities need to be set (ahead on time) for almost everything in agile and test cases are not the exception. Since we are not running all our set of test cases every iteration, we need to prioritize them based on how critical is the functionality being tested for a given test case.

Additionally, we need to have a way to easily identify, group and execute test cases given an specific criteria (i.e. All the test cases with priority = P0), so we can easily adapt to the testing requirements of a given iteration.

9.3 Multiple testing environments

One of the most important objectives of agile is to deliver working software as frequently as possible. For QA purposes, that might mean that the software to be tested will be constantly changing, QA cycles will be hard to complete (properly) and it will be very hard for QA to assure that a given software version is ready to be delivered to customers since it's very likely that is has not been tested thoroughly.

One of the well known (but not always followed) best practices is to have multiple testing environments and use each one of them for a very specific testing objective. The following are some of the most commonly used testing environments.

9.3.1 Stable Environment

Even though new functionality will be finished and released very frequently, QA needs to have a stable version of the product where a complete test cycle can be performed so that, at the end of the test cycle, QA can say whether the product is ready to be delivered to customer with complete certainty.

Since regular iterations typically take no longer than 1 or 2 weeks, it is a very common practice to "freeze" the code of the product every week / iteration and pass that "version" of the product to QA for detailed testing. That way, QA has a stable version of the product that will not be replaced until the next week / iteration, allowing the QA team to:

1. Complete all the necessary tests in the cycle (including regression, integration testing…)
2. Properly determined whether the product is ready or not, based on the results of a complete test cycle, not only a smoke test (that is very likely to miss important issues)

If QA states that the build is ready to be delivered, it is very likely that it will be pushed to the production environment since it has been tested thoroughly and QA has certified its quality.

9.3.2 Constantly Updated Environment

Development team is constantly adding new features, fixing bugs found by QA or refactoring the code to improve a given functionality. Since many of this changes start from a higher level requirement that needs to be satisfied as soon as possible and QA needs an environment to check them without affecting the regular test cycles.

It is very common to have a testing environment regularly pushed (daily) with all the changes recently included, so that QA can verify them as soon as possible (even though they are not integrated on a "final" version of the product as in the stable build).

The idea behind this "changing build" is to have new features and newly introduced changes tested at least on a "partially integrated" environment so that any defects related can be detected early on the process.

9.3.3 Load and Performance Environment

Load and performance tests have certain environment requirements that need to be met if we want to get useful information from their results. Some of the requirements are:

- The “version” of the product being tested should be as similar to the one pushed / to be pushed as possible.
- The testing environment (including hardware) that will run the product should be as similar as possible (ideally exactly the same) to the real production environment.
- The only purpose of the environment should be testing, so that no other processes running affect the results obtained (unless production environment also has additional load).

Given this, it is clear that QA needs an additional testing environment exclusively dedicated to load and performance testing.

9.3.4 Other Environments

If possible, additional environments for miscellaneous purposes such as Demos or development testing are welcome but not necessary.

9.4 Customer involvement

In a sense, QA represents end users of the product being developed. An important part of QA people’s job is to assure the product meets end user requirements and users need to be involved in the whole process in order to accomplish it.

End users attend many of the meetings held in the process and QA must take the most of any of those meetings by clearing any doubts / concerns and understanding what users want.

The more QA knows about users and their needs, the better results QA will produce.

9.5 Communication

The software development process generally involves the work of a big group of people. Keeping everybody in the same page is not trivial and communication is the key to accomplish it. The better the communication between QA and rest of the team is, the better the chances for QA to obtain the desired results are.

Constant communication with developers, project managers, end users and within the QA team (of course!) will end up in many benefits for the whole team. Some of the benefits are:

- Better understanding of the product being developed
- Everybody’s efforts focused in accomplishing the same goals.
- Lower rate of conflicts between QA and Development due to misunderstandings.
- Lower rate of invalid quality issues reported
- High integration between teams (QA, Development…)
- Better criteria for QA team to effectively focus testing efforts
- User needs well known across the entire team, so they are affectively addressed.

One of the most commonly found communication issues in traditional approaches has to do with test coverage sharing between developers and QA.

Typically, the testing efforts are completely shifted to the QA team and once developers start some kind of testing effort, such as unit testing, there is no control of what's being already covered by developers so that no duplicate work is performed. Close work and the proper communication may help resolving this kind of issues.

9.6 Flexibility

QA people in agile projects need to be able to perform any of the possible activities within a QA cycle (or most of them) whenever is needed. This flexibility of the QA members will allow the team to:

- Easily adapt and react to changes in the team line up (i.e. due to people leaving the project or the company) without affecting QA performance.
- Perform periodical assignment rotations so no one in the team get "isolated" testing a specific part of the product forever.
- Spread knowledge about different testing areas (regression, automated, load and performance, …) through all the team members

It's important to mention that this flexibility is not just a matter of organization. It is necessary that QA people have a strong technical background (or the necessary time frame to get it) so they can perform all the different activities effectively when required.

Additionally, QA functions should be clearly defined in a way that any QA resource assigned to a specific testing role knows what are the tasks to be performed and what are the expected results for that (in terms of time, coverage, reports…)

10 Typical Agile QA Cycle

Depending on the objectives of a given QA cycle the activities to be performed might vary a little. However, there's s set of activities that are typically executed on any regular Agile QA cycle and can be used as a base point to start no matter what the objectives are.

1. Run your set of automated test cases
    a. Verify the outcome
    b. Manually check failed test cases
    c. Report new / regression bugs if needed

2. Verify any bug fixes included in the new build
    a. Close / Update / Reopen bug reports accordingly
    b. Update user stories related (Accept if fixed, Reject if not)

3. Verify new stories delivered in the new build
    a. Accept or Reject depending on the verification results
    b. Add the necessary comments
    c. File new bugs if needed

4. Run your manual test cases
    a. Select which ones you need to run based on test case priorities
    b. Report any new/regression bugs introduced in the build
    c. Create the necessary user stories for the new issues
    d. Communicate any findings to the rest of the team (QA / Developers / Management)

5. Update your test status in a timely manner (In the proper tools for this purpose)

6. Update existent test cases in case of any modifications in the product

7. Create test cases for new functionality included

8. Review user stories for the current (also for the next cycle if possible) to make sure they have all the necessary information to be verified

9. Attend iteration planning meetings, demos or any other meeting of interest for QA

10. Attend daily stand up meetings (both QA and Development meetings if possible)

11. Automate test cases (as many as time allows you to)

11 References

The following sources have been used as reference for the creation of this document:

- InfoQ. Video [online]. InfoQ, 2008 [Consulted on: September 6th, 2008]. Available at http://www.infoq.com/presentations/role-of-testing-in-agile-scott-ambler
- Agile Development Thoughts. The Role of QA in an Agile Development Project. Damon Poole, 2008 [Consulted on: September 7th, 2008]. Available at http://damonpoole.blogspot.com/2007/05/role-of-qa-in-agile-development-project.html
- Agile Journal. Software testing in an Agile environment: Ten Challenges QA have to face and solve, 2008 [Consulted on: September 7th, 2008]. Available at http://agilejournal.com
- InfoQ. Book [PDF] Scrum and XP from the Trenches. Henrik Kniberg, 2007
- Agile Journal. Webcast [online]. Automated Agile Functional Testing Practical Approaches to Quality that Get Results, 2008 [Attended on: September 30th, 2008]
- Agile Journal. Webcast [online].Agile Testing Realities, 2008 [Attended on: December 16th, 2008]