# Making Agile Work in the Real World Micro-Credential

## Syllabus


# Table of Contents

## Making Agile Work Testing



## References



 

# General Information

**KEYWORDS**

defect triage meeting, definition of done (DoD), definition of ready (DoR), matrix-managed, pair testing, release, sprint

# LEARNING OBJECTIVES FOR MAKING AGILE WORK

## What Makes Agile Work

(K2) Summarize how Agile safeguards relate to the safeguards in traditional models
(K2) Explain the advantages to using the Release model for production releases
(K3) For a given Agile project, apply a matrix-management structure
(K2) Explain how quality ownership can be assessed and enforced
(K2) Explain how the DoR and DoD can improve quality
(K2) Summarize examples of DoR criteria for Stories
(K2) Explain how schedule and budget constraints can impact an Agile project
(K2) Explain how the release approach can help reduce schedule slippage

## Testing in the Agile Environments

(K2) Explain how grooming sessions can improve testing
(K3) For a given User Story and set of acceptance criteria, identify what is missing to make the Story testable
(K2) Explain who is responsible for determining how the software should work
(K2) Summarize sources of information a tester can use to understand what to test
(K2) Explain when targeted regression testing should be used
(K2) Explain the leapfrogging approach to testing
(K2) Explain how risk-based testing is used in an Agile project
(K2) Explain how exploratory testing is best employed in an Agile project
(K2) Explain the purpose of pair testing
(K2) Summarize the purpose of root cause analysis

# Introduction

The Information Technology (IT) world changes almost continuously as new technologies, methodologies, and techniques are created. Some of these are adopted as-is, some are discarded, and others are adapted for various uses. The Agile lifecycle methodology has been widely embraced in principal, but in practice the methodology tends to be modified. In some cases, this modification makes sense to adapt the methodology properly to fit a particular situation, but in other cases the concept of "Agile" remains only in its name, not in the practice. Because Agile is a pervasive methodology in its various forms, it is important for all software testers to be familiar with it - in the base concepts, the pure form, and the various modifications.

For the sake of readability, the term "software tester" will be used to refer to anyone who is testing software, regardless of their formal role. In an Agile environment, each team member is responsible for contributing to the quality of the product, via the implementation of and participation in quality practices. Software testing, in this environment, is an assessment of the quality of the software that has been built.

This syllabus focuses the Agile methodology from the viewpoint of the software tester. This includes looking at how an effective Agile team works, the basic rules of an Agile methodology, and how a software tester fits into this environment. This syllabus is intended for use by all members of an Agile team as well as anyone managing an Agile project. This includes product owners, business analysts, developers, software testers, project managers, scrum masters and anyone else who is involved with the development and testing of a product in an Agile environment.

# What Makes Agile Work

Failures of Agile projects are common. While one of the goals of Agile is to achieve working code faster and with less overhead, this only works well when the team is following all of the Agile guidelines. The Agile approach has expectations for team behavior that allows the goals to be met. Agile removes some of the safeguards that are built into other methodologies, such as agreed requirements that don't change, reliance on documentation to communicate information, clear role definitions, and traditional project management structures. When the safeguards are removed and the Agile guidelines are not followed, the result will be sub-optimal at best and catastrophic at worst.

Lessons learned from many projects highlight some of the more important Agile guidelines and some modifications that help to promote project success. These are discussed in this section. While any individual project and project team's results may differ, it is good to consider each of these areas to help improve the chances of success.

## The Concept of Releases

Scrum does not have the concept of Releases, which are collections of Sprints, but most organizations think in terms of Releases as sets of functionality. Where Agile promotes the continuous flow of software into production, one Sprint at a time, many organizations cannot absorb this level of constant change. Changes to production often have a level of overhead associated with every release, regardless of how tiny. In this case, the overhead associated with the release of software developed in a two-week Sprint becomes onerous and cripples the smooth flow of code.

With the concept of releases, a set of functionality is developed in Sprints and is then released together to production as a feature set. For example, a new feature might be revamping of the medical questionnaire and subsequent risk calculation for a travel insurance application. This

is a significant set of functionality that carries its own risks regarding accuracy, security for data protection, performance, and integration with the other parts of the application. If it will take six two-week Sprints to develop this functionality, it might make sense to make this a Release.

The Release can include the following:

- A hardening Sprint to allow the final end-to-end testing to occur and all defect fixes to be verified
- A formal UAT for the users to conduct their testing and provide their approval
- A formal performance test to ensure that the new functionality will not adversely affect the production software
- A formal security test to ensure data privacy is preserved and that no new vulnerabilities have been introduced

While it is possible to do some of this testing during the Sprints, such as the security and performance testing of the components, there is still a need for a final regression test pass to ensure that the whole of the produced software has not introduced issues.

Using a Release concept still allows the team to work in an Agile way, but also provides some safety and comfort to the business in the ability to control change. Schedules are more predictable and expected functionality is delivered in a usable set. It does take a bit more time, usually an additional 2-4 weeks, but that time can save time spent resolving production issues. This time also allows test automation to be completed for the software in the release without the requirements for the stubs/drivers that are required when the test automation is developed during the sprints.

## Maturity & Management of the Team

For an Agile team to be successful, there must be a level of maturity and respect within the team. This is critical for self-management, collaboration, and equality within the team. Without maturity, an Agile effort is likely to fail and another lifecycle model should be selected that has less reliance on each individual working in a responsible and mature manner. Individuals who do not demonstrate maturity will not be able to self-manage.

Leadership is a tricky aspect of Agile. While everyone is expected to be self-managed, realistically, everyone does need to report to someone at least in an administrative capacity. This results in individuals being matrix-managed where they are reporting into the team but also into their administrative manager. Added to this complication are the roles of the Product Owner and the Scrum Master. When there is a management void, it is not unusual for one of these people to try to fill in. Unfortunately, these may not be the right people to actually "manage" the team as their roles are not supposed to include management. When they do try to manage the team, the natural checks and balances expected in an Agile team are compromised.

The best solution to the management quandary is for each individual to have a membership within an organization aligned with their role, e.g. tester, developer. This provides the administrative support for the individual - career growth, training opportunities, mentoring, pastoral care. This is often accomplished in organizations by having the individuals report into specialized chapters or centers of excellence that provide their long-term relationship with the organization. Individuals are then "loaned" out to Scrum teams where they will have an internal reporting structure. This allows individuals to be self-managed within their teams, but to still have an administrative management structure to which they always belong. Scrum teams will come and go and it's important for employee retention for people to feel they have a longer-term role within the organization.

It's important to remember that not all individuals will like working in Scrum teams. There is an expectation of independence, collaboration, and

having the knowledge, ability and willingness to achieve success. While some teams will work well with nurturing junior team members, others will not have the time to provide the necessary support. Some people prefer less collaboration and more independent working. Testers are sometimes frustrated by the changing requirements and the continuous need to retest. A Scrum team can be very rewarding, but it's not for everyone.

## Establish the Rules

There is a tendency in industry for Agile to be implemented as a "learn while doing" experiment. There may be some initial training and some coaching, but the team is largely left on its own to make mistakes and learn how to work together. Setting some ground rules early can help set up the team for success and to minimize misunderstandings.

Agile projects use a significant amount of terminology that may be new to the team. Understanding the terms that will be used by the team and clearly defining these terms will help everyone to communicate more effectively. Similarly, understanding the scope and responsibilities of the various roles will also help. What exactly does the PO do? Is there a BA on the project? Who decides who is right if there is a debate between the developers and testers? These are all valid questions and should be resolved before starting into that first Sprint.

Quality rules must also be established. It's great to say that everyone is responsible for quality, but how will that be exhibited, verified and enforced? What happens if a developer is consistently introducing low-quality code which is resulting in accumulating technical debt and interfering with effective testing progress? Quality gates must be established and must be measurable and agreed. It's a good idea to set SLAs for defect fix turnaround to help eliminate blockages.

Clear and unambiguous definition of the Definition of Ready (DoR) and the Definition of Done (DoD) are very helpful in keeping the quality standard high and enforceable. If DoR is not met, the story or code cannot advance to the next stage. For example, if there is a DoR requirement for testing

that the code must have achieved 65% coverage from unit testing, that is easily measured, verified and enforced. Similarly, the DoD helps to keep the team conscious of the requirements for moving to the next stage. If the DoD from testing is that all high-risk stories are covered with automated tests, the team cannot count the story points if a high-risk story's automation is not completed. This will result in a reduced velocity, which may be correct, and will help reduce the pressure to automate later - since later may never come.

DoD and DoR don't just apply to coding and testing. They also apply to the assessment of stories that are being selected for a Sprint. If a story doesn't have adequate Acceptance Criteria or is not defined enough for the team to be comfortable with the implementation and testing, it's not ready. The DoR for stories should include the level of clarity, the definition of the acceptance criteria, prioritization, risk assessment, and a realistic estimate of effort.

The DoD and DoR must be applied consistently. As soon as variances are allowed, the rules are being broken and there's likely to be a cascade of "special cases". This can result in sloppy work that pushes the difficult and time-consuming work, such as test automation, to a later time. At times it may make sense for the entire team to re-evaluate the DoR and DoD, but this must be done consciously and by the entire team so that the ramifications of change are understood.

## Time & Money Constraints are Real

It is rare to find a project that is truly not constrained by time and money. This immediately breaks the Agile rule of allowing the software to grow organically as new requirements are found. Welcoming change sounds great for the user and demonstrates flexibility, but will the changes actually fit within the necessary timeframe and budget? If not, welcoming change may not be possible. It's important that the PO and end users have an understanding that while some requirements change during development may be possible, not all changes can be accommodated without budget or schedule changes. Sadly, end users and POs are often sold into the Agile plan by understanding that they will have a wide ability to

make changes - and they are sorely disappointed when that doesn't happen. As a result, while it would be great to start coding and see what happens, business reality doesn't allow that flexibility. The end product must do x and it must do it by x date within x cost. This is one of the ways the release structure can help to control the changes in the project.

Another important concept is that the MVP is not the final product; or, maybe more accurately, it shouldn't be. When projects are discussed and proposed, it is the full set of capabilities, not a scaled down version, that must exist in production for a significant period of time - including being supported by the team for that time. When the MVP becomes the final product, it's usually because the time and/or money have run out. This continual slippage is also better controlled with the release approach rather than just a large set of Sprints. Slippage is better controlled with frequent milestones; if quarterly releases are planned, then each 12-week cycle allows an assessment of the progress toward the final product. This results in less surprises at the end and some hard decisions earlier.

A project that delivers only the MVP is not successful. A project that delivers the MVP and then continues to enhance that MVP to reach the full planned product, is successful. Particularly in organizations that are new to Agile, it's important to deliver the expected product.

# Testing in the Agile Environment

Working in an Agile environment poses some unique challenges to testers. Fortunately, good testing practices still apply. As with all projects, the accuracy of testing depends on the accuracy of the requirements.

## Get the Requirements Right

Since the requirements will be in a User Story format, it's important for the testers to be involved in the grooming sessions where the Story is explained and the acceptance criteria are defined. The more complete and accurate this information is, the better the testing can be targeted and the risk can be assessed. Proper prioritization, risk analysis, and an understanding of any non-functional requirements should all be outcomes from the grooming session.

Testers may have a tendency to take a less active role in the grooming sessions or even to be intimidated during the sessions, but it's important to remember that all team members in an Agile team are equal - each with their own responsibilities. Everyone's voice matters. This is the time to speak up and persist until there is a shared understanding of each story. Clearly defined acceptance criteria are critical for everyone to understand what must be delivered. Be sure the acceptance criteria are testable and cover error handling, data requirements, and usage scenarios. For example, "must accept an address" is not clear enough. What is the format? What happens if the address is not valid? What happens if it's not supplied at all? It is the tester's responsibility to ask these questions and get the answers during the grooming session. If the questions are not asked and answered, the developer will decide how it will work and there's no way to assess if that decision was correct.

Ultimately, the Product Owner determines how the software should work. The PO is responsible for communicating with the end users to understand

what they need. If possible, the tester should also work with the end users to understand usage scenarios and use cases that can be used in exploratory testing. It's important to testers to seek out all sources of information to best understand what to test and to prioritize tests. Reading documentation, investigating legacy systems, talking with other teams regarding integrations are all ways to expand the knowledge needed to test effectively.

## Organizing the Testing

With any project, the incoming quality will determine the testing effort. It is a good practice to ensure that unit test coverage is in the DoD for the developers to release code into the build. This coverage must be assessed and reviewed by the team before code is allowed to be released. By ensuring this practice is in place and is consistently reviewed, quality ownership becomes shared by the team.

It can be quite difficult to keep up with the testing work, particularly in the later Sprints when there is a larger amount of code to be regression tested. Targeted regression testing is the best approach, requiring close collaboration with the developers. As mentioned above, the use of a hardening sprint to conduct final end-to-end regression testing provides an additional safeguard from regressions creeping in, particularly later changes that affect early code.

The leapfrogging approach can work well to allow the testers to fully test and automate the code from a Sprint. This requires at least two testing teams who will alternate owning a Sprint, effectively giving each team two full Sprints of time to test the code from one Sprint. In this case, the first team would test Sprint 1, the second team would take Sprint 2, then the first team would take Sprint 3 and so on. This allows more time for testing without compromising the ability of the testers to be involved throughout the Sprint, including participating in the planning and grooming sessions.

Using the right tools will also save significant time and will support continuous reporting. Sprint testing doesn't allow time for gathering and reporting metrics - those need to be accumulated

automatically as data is entered into the test/defect management system. Rather than using a general tool such as MS Excel, it's better to use tailored tools such as Jira, Azure DevOps, Rally, etc. to store the testing and defect information. These tools support maintaining the backlog, prioritizing the work, assigning the work, monitoring completion of tasks, and traceability.

When configuring a tool, it's important to consider the reporting that will be needed. In Agile projects, dashboards are frequently used to provide up-to-date information on the development and testing efforts. Configuring these dashboards at the beginning of the project and ensuring the proper tagging is being used on test cases, etc. will enable automatic reporting and will save significant time throughout the project.

Atlassian's Confluence is frequently used in Agile projects to store architecture and design information. While Confluence is an easy tool to use, using it with proper versioning and ensuring that outdated information is labelled as such, tends not to happen. Because Agile projects move rapidly, it's easy for information to be valid only for a point in time. This can be misleading when an old design document becomes the basis for testing. If Confluence, or something similar, is used to store project information, be sure that versioning and validity dates are also used.

## Good Testing Practices

As with all time constrained projects, risk-based testing provides an effective approach to address the highest risk areas first and to determine the depth of testing required. Tests should be prioritized based on the risk that will be mitigated by the test. Features with high risk levels will require more depth in testing whereas low risk features may be adequately covered with more cursory testing. Setting risk mitigation requirements in the DoD will help to ensure that quality standards are met and that the project is achieving the risk mitigation goals.

The proper testing techniques to use depend on the criticality of the software, the risk mitigation goals, the time available, and the level of detail available in the requirements. In general in an Agile project, exploratory testing should always come first. When software is released into testing, conducting exploratory testing will provide rapid feedback to the developers and will help confirm the tester's understanding of the software.

There are several schools of thought on the use of detailed test cases in Agile projects. Some projects require evidence of due diligence in testing, such as safety-critical projects. These may require detailed test cases with defined steps and a record of execution. Developing and maintaining detailed test cases takes time, particularly when the requirements may be changing.

Test documentation needs to meet the needs of the project and must meet the DoD for testing. In some cases, the Acceptance Criteria for a Story may be the basis for a testing checklist. In other cases, mind maps may be used to plot out various feasible tests for a particular feature. There is no one right answer in Agile projects. The test documentation must be suitable for the project and the tester and the team.

Testing is sometimes done in pairs. This is an approach that was first introduced in Extreme Programming but has also been adopted in some Agile circles. Pair testing simply means that two people work together on a test. This pair could be a tester and a tester, a tester and a developer,

or even a tester and a user. The goal is to get the best ideas from the two minds, to evaluate the outcome together, and to maximize the coverage and efficiency of testing.

Regardless of the test approach used, the depth of test documentation and even the completeness of the requirements, the tester must consider the quality characteristics of the software being tested. This includes the functional areas, but also includes the non-functional areas such as:

- Performance
- Security
- Usability
- Compatibility
- Accessibility

The importance and depth of testing for these areas should be considered in the risk analysis. Software that meets all the functional requirements may still fail if the non-functional requirements are not met. It is particularly tricky to schedule the non-functional testing in an Agile project since it sometimes requires the complete product to finalize the testing (such as performance or security) or requires testing labs that are expensive to procure for each Sprint (such as usability labs). While these areas can be assessed at the Sprint level, they are often assessed in totality during the hardening sprint.

## Defect Management

Accurate defect reporting and good defect management are important in Agile projects. Solid defect practices, such as supplying steps to reproduce, screenshots, and environment information, are just as applicable to Agile projects as to any other project. Prioritization is particularly critical in Agile projects because of the positioning of the defect into the backlog. It is more efficient for the testing if defects are resolved quickly - this avoids blockages and also allows testers to explore more areas of the software. While the PO is responsible for prioritizing defects for fixing, the tester also needs to supply input regarding the impact of the defect on the testing progress.

Defect triage meetings can be particularly helpful to ensure prioritization is accurate. It also helps to review the backlog to ensure there is sufficient turnaround time on defect fixing. A bloated backlog may cause slow defect resolution which

may in turn block or severely hamper manual testing and test automation development. Short turnarounds and defined Service Level Agreements (SLAs) for defect resolution will help the entire team work more efficiently. Defects should not be left for the hardening sprint - they must be addressed as they are found and triaged.

While it is common practice for all defects reports to be entered into the backlog, some defects require fast fixes to enable test progress. There must be a method for classifying these defects and getting them into the developer workload in the current Sprint. This means that when stories are selected for a Sprint, the sum of the Story Points must be less than the expected Velocity to allow time for defect fixes. Over time the team will learn to adjust the Story Point allocation to allow for a percentage of defect fixes. Some teams actually include the defect fix time within the Story Points, but this can be difficult to assess, particularly early in a project.

Defect triage meetings, if used, must be efficient. Only new defects should be reviewed and prioritized. All the necessary people must attend, including the developers, PO, and testers, and decisions and action plans must be the outcome from the meeting. When determining the time needed for defect fixes, it's also important to include the time necessary for confirmation and regression testing. For critical defects, the fix and testing should occur in the current Sprint. For non-critical defects, the fix and testing will occur in a future Sprint.

In order to ensure the convergence of found and fixed defects, each defect should be prioritized and the Sprint Planning Session must review all defects for possible inclusion in the release. A release with all critical defects fixed can still contain a large number of defects that will be annoying for the users so it is important to be sure that all defects are reviewed and planned for resolution.

Because there is a higher attention to quality in an Agile project, there should be less defects. If that is not true, review is needed of the DoR and DoD requirements to see where defects are slipping through. All escapes should be analyzed to understand the root cause and to improve the processes to reduce future defects. Prevention is much less expensive and time consuming than resolution.

## Lessons for Testers

There are few projects more fun, exhilarating and satisfying than a well-executed Agile project. Successful Agile teams bristle with energy, develop deeper skills and understanding, and gain a higher level of satisfaction from their releases than traditional teams. Agile does come with challenges though and those must be addressed and dealt with rapidly and maturely. Agile is not for the faint hearted, but it can be immensely rewarding. Embrace the challenge, but don't expect it to be easy.

# Terms

**Defect triage meeting:** a scheduled meeting with all stakeholders to review and prioritize new defects and assign each one for action.

**Definition of done (DoD):** The collection of exit criteria which is used to determine if a backlog item is complete.

**Definition of ready (DoR):** The collection of entrance criteria which is used to determine if a Story or task is ready to move into the next phase of implementation.

**Matrix-managed:** A management structure where an individual reports to multiple entities, directly or indirectly.

**Pair testing:** The practice that leverages two different viewpoints for a single test effort.

**Release:** A set of Sprints in an Agile project that provide fully implemented Epics and Features to the users.

**Sprint:** An iteration of development in the Scrum framework, normally defined as 2-4 weeks long, which results in a small piece of usable functionality.

# References

## Works Cited


*AWS. (2023). What is DevOps? Retrieved from DevOps: https://aws.amazon.com/devops/what-is-devops*

Beck, e. a. (2001). Principles behind the Agile Manifesto. Retrieved from Agile Manifesto: https://agilemanifesto.org/principles.html

Beck, o. (2001). Manifesto for Agile Software Development. Retrieved from Manifesto for Agile Software Development: https://agilemanifesto.org/

Leffingwell, D. (2021). Welcome to Scaled Agile Framework. Retrieved from Scaled Agile Framework: www.scaledagileframework.com

Monday. (2023). The ultimate guide to Kanban and how to use it in 2023. Retrieved from Mondayblog: https://monday.com/blog/project-management/kanban

scrum.org. (2020). scrum.org. Retrieved from Scrum.org: www.scrum.org

Sutherland. (2020). The 2020 Scrum Guide. Retrieved from Scrum Guides: https://scrumguides.org/scrum-guide.html

Sutherland, S. (2020). scrumguides.org. Retrieved from Scrum Guide: https://scrumguides.org

## Purpose of this Document

This syllabus forms the basis of the AT*SQA certification for Agile Software Testing Methodologies. AT*SQA is an International Standards Organization (ISO) compliant certification body for software testers. AT*SQA provides this syllabus as follows:

1. To training providers - to produce courseware and determine appropriate teaching methods.
2. To certification candidates - to prepare for the exam (as part of a training course or independently).
3. To the international software and systems engineering community - to advance the profession of software and systems testing and as a basis for books and articles.

AT*SQA may allow other entities to use this syllabus for other purposes, provided they seek and obtain prior written permission.

## Acknowledgements

This document was produced by a core team from the AT*SQA Syllabus Working Group – Agile Syllabus:

Authors:
Judy McKay

Reviewers:
Randy Rice
Earl Burba

The core team thanks the review team for their suggestions and input.

 


www.atsqa.org