AT*SQA

Agile Implementation Micro-Credential

Syllabus

AT*SQA MICRO-CREDENTIAL

Agile Implementation

Table of Contents

Agile Implementation Testing

6	Introduction
8	Establishing the Team
11	Defining the Process
16	Planning
22	Defining the Requirements
25	Testing
31	Terms Used

References

35	Works Cited
35	Purpose of Document
36	Aknowledgments

General Information

KEYWORDS

acceptance criteria, behavior-driven development (BDD), burndown chart, epic, feature, feature flags, hardening sprint, hub-and-spoke, integration testing, Kanban board, planning poker, product backlog, production implementation verification (PIV), product owner, release, requirements traceability matrix (RTM), retrospective, scrum, scrum master, scrum of scrums, sprint backlog, stand up, story grooming, story points, system integration testing (SIT), system testing, task board, technical debt, test-driven development (TDD), unit testing, user acceptance testing (UAT), user story, velocity

LEARNING OBJECTIVES FOR API TESTING

Establishing the Team

(K2) Explain the common roles in a Scrum team
(K2) Explain a hub-and-spoke management structure

Defining the Process

(K2) Summarize the relationship between features, epics and user stories
(K2) Explain the purpose of the daily Stand Up meeting
(K2) Explain the usage of the Task Board
(K2) Explain the purpose and usage of the Definition of Ready
(K2) Explain the purpose and usage of the Definition of Done
(K2) Summarize the testing issues that can be caused by the use of feature flags

Planning

(K2) Explain the best usage of Test Plans and Test Strategies in Agile projects
(K2) Summarize the usual contents of a Release-level Test Plan
(K2) Summarize the content of the Product Backlog
(K2) Explain the usage and purpose of Story Points
(K2) Explain what occurs during Story Grooming
(K2) Explain how Acceptance Criteria are used during testing
(K2) Explain how the Product Backlog and Sprint Backlog are used

Defining the Requirements

(K2) Explain the relationship between a feature, an epic, and a story
(K2) Explain the usage of the given/when/then format for defining a story
(K2) Summarize the areas that should be covered by the Acceptance Criteria
(K2) Summarize the methods used to evaluate stories

Testing

(K2) Explain why automation is needed in Agile projects
(K2) Summarize how the test levels apply in Agile projects
(K2) Explain how the Requirements Traceability Matrix should be managed in an Agile project
(K2) Explain factors that influence the level of detail in test documentation
(K2) Explain the usage of a Burndown chart
(K2) Provide examples of technical debt

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.

Establishing the Team

Once an organization has determined that Agile is the proper approach for a project, the Agile team must be created. In Scrum this is usually called the Scrum Team.

Agile Team Roles

The Scrum team usually consists of the following:

- **Scrum Master (SM) -** keeps the team focused on implementing Scrum correctly, accountable for the team being effective
    - Coaches team members
    - Helps team to focus on creating deliverables that meet the Definition of Done
    - Removes blockages and impediments
    - Ensures all Scrum events take place and stay on track
- **Product Owner (PO) -** accountable for maximizing the value of the delivered product for the users
    - Develops and communicates product goal
    - Creates product backlog items
    - Prioritizes the backlog
    - Represents the needs of the stakeholders
- **Developers -** create the increments of software for each Sprint
    - Create a plan for the Sprint
    - Ensure quality in accordance with the Definition of Done
    - Adapt as needed
    - Hold each other accountable

Notice that testers do not have a defined role in Scrum, rather that they are included in the Developers due to the shared nature of quality.

In reality, the testing role does usually exist as a role separate from a pure developer role. Both of these roles constitute the majority of the team. Also notice that the team definition does not include Business Analysts (BAs), with the assumption that the PO will take over that role or the BA's will work with the PO outside the confines of the Scrum team.

Scrum Team Management

Team and project management in Scrum can be difficult to implement. One of the primary tenets of Agile is that the team should be self-managed, but in reality this is usually not totally possible. Business is usually hierarchically structured and while a team may work on its own, its deliverables are still the responsibility of some form of management.

Project management in Scrum often exists outside the team, but within the project structure. A project manager (PM) may be responsible for the outputs from multiple scrum teams and may have a role in coordinating the work between the teams, attending and participating in the "scrum of scrums" which is a collection of all the scrum teams, to coordinate efforts.

Project managers are normally responsible for the budget and schedule for a project. This often imposes a difficulty with "true" Agile where change is "welcomed" and scope is fluid. Late changes to requirements are often costly both in terms of time and budget and controlling these changes sometimes falls to the PM. This can put the PM - who is concentrating on schedule and budget - at odds with the PO who is working to fulfill the needs of the user.

People managers are also not included in the Scrum team. The concept of a Development or Test Manager doesn't exist in Scrum. Instead, everyone is a responsible, self-managing individual. While this construct may work on a day-to-day basis within the team, there is still a hierarchical reporting structure for the team members, meaning that they have a "manager" who helps to guide their career, controls merit and promotion increases, and has a long-term relationship with the individual. Scrum teams are formed and dissolved as projects come and go. The manager / employee relationship persists across projects.

In both development and testing, the concept of a central organization (e.g. center of excellence, community of practice, chapter) with a traditional management structure to which individuals administratively report is often used. In this model, sometimes called hub-and-spoke, the central organization provides administrative support for the individuals including training, career development, promotions, hiring / firing, salary increases, etc. The individuals are then "loaned" to a project to create a Scrum team. In this way, the individual still has organizational support for their career but can work independently in a self-managed way within the Scrum team.

While this is conceptually clear, in practice it can be difficult to understand where the self-management begins and ends. More mature, capable individuals will thrive well with a relatively hands-off management approach, where those with less maturity or experience may require more support from the central organization and their peers.

To avoid problems occurring during development and planning, it is important to clarify the traditional roles and ensure there is a working relationship and good communication between the individuals in these roles.

Defining the Process

The level of adherence to the Agile processes, particularly the Scrum processes, will vary by team. This section discusses the most commonly used practices in industry.

Product Backlog

A User Story is a short definition of a small, deliverable, piece of meaningful functionality. Stories are usually written in the format of "As a... I want to... So I can...". This is easier to understand with an example. Example: the team is building a mobile application that will get data from a sensor on a dog's collar to provide the owner with status information on the dog. One User Story could be:

> As a Dog Owner
> I want to check my dog's temperature
> So I can decide if I need to turn up the air conditioning

Once defined, a User Story is added to the Product Backlog, which is a collection of Stories and tasks that form the workload for the team. Tasks in the backlog might include defects that need to be fixed or maintenance work that needs to be done such as refactoring a piece of code. Primarily though, the backlog is composed of User Stories.

A feature is an overarching set of functionality that will be decomposed into Epics. Epics are then further decomposed into User Stories. In the above example, a Feature for this product might be "Determine my dog's overall health". This would be based on taking multiple sensor inputs, plotting them over time, creating a health profile and comparing that to a norm found in some reliable source. Each of these components could be defined as an Epic. Each of these Epics describes a considerable amount of work and requires multiple stories and potentially work from multiple teams.

Projects tend to start by defining Features wanted by the users. These are decomposed into the Epics which are then decomposed into individual Stories. Stories are then further refined by Grooming sessions in which the team discusses additional information needed to implement the Story and test the result. Epics are also stored in the backlog and may create a hierarchy where an Epic has sub-tasks which are the User Stories that will implement the Epic.

The Backlog is the basis for the "requirements" for an Agile project. All work that is to be done must be entered in the backlog. This includes additional work that is discovered as the project progresses, including technical debt as well as changes from the users.

Stand Ups (also called Daily Scrums)

Scrum teams must communicate openly and consistently. To help facilitate this, daily Stand Ups are conducted. These are designed to be short meetings, generally 15 minutes, where the entire team stands to provide individual status. Each person is expected to report:

- Tasks completed yesterday
- Tasks planned for today
- Any blocking or problem issues

These meetings are usually held in the morning at a standard time and each person on the team is expected to attend and participate. The Scrum Master is usually the moderator of the meeting, ensuring everyone has a chance to talk and taking any extended discussions offline. The goal of the meeting is to ensure everyone is able to progress toward the Sprint goal and has a plan for forward progress.

Stand Ups are a powerful way to keep the team focused and working together, but should not replace on-going communication within the team. Open chat channels, instant messaging tools, conversations, emails, etc. are all methods to help keep up the open communication. If the team is not co-located, having effective tools becomes more important and ensuring everyone is available via those tools becomes imperative.

Task Board

An Agile Task Board or a form of a Kanban Board, is commonly used for the team to track progress of the individual stories being worked within a Sprint. Tools are available that can provide a workflow-based digital task board. Task boards can be physically created using a white board and sticky notes representing the stories. Task boards, like Kanban boards, can have columns defined as needed by the team.

In its most simple form, a task board has three columns - To Do, In Progress, Done. Cards are moved through the columns as the task represented by the card is completed. In a more complex form there are columns for the stages of the workflow, such as Analysis, Design, Development, Testing, and Deployment. For any one column, the To Do work is in the previous step of the workflow and the Done work is now in the next column of the workflow. Cards in a particular column represent the work that is currently in progress for that column.

Task Boards are often reviewed during team Stand Up meetings to ensure that all cards are in the correct columns and nothing has been languishing for too long in one column. The Task Board provides an easy way to see what is in progress, what hasn't been started, and what is completed. It's also very easy to see where there are bottlenecks, or potential bottlenecks, in the process.

Definition of Ready (DoR) and Definition of Done (DoD)

Movement of work through the workflow is governed by the DoR and DoD.

Definition of Ready (DoR) is a set of rules that are applied to assess when a task has achieved readiness to start a new step of the workflow. The DoR varies depending on the step. For example, for a Story to move from the Backlog into Analysis, it must have completed grooming, the effort must be estimated and agreed by the team, the acceptance criteria (i.e. how to know if it has been implemented correctly and completely) must be defined and the team must agree to an understanding of the functionality. Ideally, this information is documented as part of the Story to help alleviate later questions. The DoR for a Story to move from Development to Testing may include testing notes, test data, test environment readiness, stubs/drivers available, etc.

There is not a standard set of DoR rules - each team and project has its own needs and requirements. It's not unusual for a team to alter the DoR rules as a project progresses and issues are discovered in the process.

The Definition of Done is similar to the DoR, but it is used to determine that all the tasks for a particular stage in the workflow have been completed. For example, the DoD from the development stage may include completed unit testing (usually determined by reaching a coverage goal), completed code reviews, code check-in, successful build and integration, implementation of the unit tests into the automated testing flow, etc.

The DoR and DoD can be considered to be quality gates ensuring that all necessary tasks are completed before the story moves to the next step in the process. In some contexts, the DoR may be equated to Entry Criteria and the DoD may be equated to Exit Criteria. Strict application of these rules helps to reduce technical debt and encourages the quality ownership across the team.

Some teams will use only one DoR which is used to determine if a story is ready to development, and one DoD which is used to determine if the story is ready for deployment. This less rigorous approach to quality gates may work for an accomplished team with good internal quality practices, but may be insufficient for teams needing more quality checks during the process.

Workflow

In an Agile project, the flow the work is usually as follows:

1. The project is defined
2. Features are identified
3. Features are converted to Epics
4. Epics are decomposed into Stories
5. Stories are selected for work in an Iteration
6. Completed Iterations are formed into Releases
7. Software is released to the users

In a CI/CD model, continuous integration and continuous delivery are in play where software is released through the pipeline in a continuous manner potentially all the way to production. In the case of feature implementation though, all the Stories required to implement a feature are gathered together into a Release. The software may flow to production as the stories are completed, but this completed code is usually held behind a "Feature Flag" that is turned on when all the software for the feature is completed and tested. Only at that time is the new software actually visible in production, although it has been in the environment for a period of time - just hidden behind the Feature Flag.

The use of Feature Flags complicates testing. Because flags can be on or off in production, and there may be more than one flag in use, all combinations of on and off must either be tested, or certain combinations must be prevented from occurring in production. Once a feature flag has been on for a period of time in production (usually a month), it must then be removed from the code so it can no longer be turned off, potentially by accident. This helps to remove some of the testing burden.

Planning

Planning for an Agile project has the normal project planning requirements as well as specific Sprint and Release planning functions.

Project Planning

While Agile projects can be run without the assistance of tools, this necessitates a co-located team and an “everyone present all the time” model. This is rarely reality, so tools should be considered as a necessity in an Agile project. Agile management tools help with storing the Backlog, providing a Task Board, supporting a workflow model, tracking metrics, and providing dashboards and reporting. Because project management is likely to be needed unless the project is quite small, normal project reporting is likely needed. While Agile promotes minimizing documentation and providing working code instead, it is unusual to find a management team that is happy to wait to see working software rather than being able to monitor progress.

Tool selection is highly influenced by the needs of the team for storing, accessing and reporting information. It is important to accurately assess those needs prior to selecting a tool. While most tools do support export/import to/from .csv files, these sometimes do not provide a readable format that is easily imported into another tool.

When a task management tool is selected, it is important that the team understands the usage guidelines. This will improve the accuracy and reportability of the data. Using tags to identify stories that will be reported together is a way to work with the tool to increase its reporting capabilities. As most of these tools are essentially task management tools, they tend to be weak in providing good trend reporting unless that information is built into the tracked information. Standard team agreements such as priority, severity, risk and other ratings must be agreed and documented.

It may be necessary for tools to integrate together. For example, an existing source code management tool may need to be merged with the task management tool so that code releases can be reflected in the task management tool. Some of these integrations can be tricky and may require development work. It's important to identify any integration requirements and ensure the integrations can be created as needed. Relying on manual replication of information is time consuming and error prone.

While this section is focusing on Scrum, there are other methodologies and approaches that may be employed within a project. In a big project, it is likely there will be multiple methodologies in use with some teams working with Agile while others are using Waterfall. This can work, but it requires planning to ensure that the terminology across the methodologies is understood, such as Definition of Done, and that the resultant project releases will be able to integrate and provide the necessary functionality to the user.

Test Strategies and Test Plans are not a part of the Agile methodology, but projects may still need these. The Test Strategy is used to define the overarching testing methodology, quality goals, test approach, environments, tools, etc. for an organization. The Test Plan is the application of the Test Strategy for a particular project. A Test Plan should not be at the Sprint level as that information, such as which test cases will be executed, is better tracked in the task management tool. A Test Plan is an agreement on the testing approach for a particular project. This agreement is still needed, even in an Agile project. These agreements feed into the DoR and DoD in areas such as the amount of test automation to be developed, the test environments, test data, etc.

Release Planning

While the concept of a release is not strictly within the definition of Scrum, it is commonly used in business to organize functionality into a "release" that will meet the needs of the business users. Releases are sometimes aligned to Epics in that the release contains all of the Stories required to complete an Epic. In organizations where users require significant UAT time and training, continuous delivery/deployment to production may not be feasible. In this case, organizing stories into Releases may make sense - allowing a single UAT for a set of features and a single set of training and deployment.

When planning a Release, or even assembling a set of Sprints, there is sometimes a planning sprint, called Sprint 0. While this is not a Scrum concept, it is commonly used in practice to create the core upon which the Sprints will build, to create the test and development plans, to organize and prioritize the Stories, and to define the goal of the Release.

From the testing standpoint, with or without a Sprint 0, the test team still needs to conduct the release-level test planning which may result in a release-level test plan. This test plan will include such information as:

- Test techniques to be employed (exploratory, decision table, etc.)
- Test environments (including definition of the required integrations)
- Test data (how it will be created, who will own it, anonymization)
- Test schedule
- Test responsibilities and assignments (unit testing, automation development, etc.)

Regarding the test schedule, there are a number of different approaches to testing in an Agile project. While Scrum defines that each Sprints ends with a deliverable that is useful to the users, that is sometimes not practical. For example, the code developed in a Sprint might need to be tested with code from previous Sprints, or may require particular environments or data, or may require extensive regression testing. In this case, there may not be enough time to conduct the testing within the timeframe of the Sprint, particularly if the Sprint is short. In this case, testing sometimes starts in the Sprint where the code is created, but continues into the next Sprint. This allows more time for adequate testing, but does require that the testers either work across multiple Sprints at the same time, or the testers divide into groups so that while one group is testing the software from the previous Sprint, the other group is interacting in the planning and activities for the current Sprint.

This is a problem also seen with test automation. Ideally, the test automation is developed during the Sprint in which the code is developed, but the stability of the code is rarely achieved early enough to allow adequate time for the development of the test automation. Test automation usually lags at least a Sprint behind the development of the code. This makes test automation useful for regression testing, but not of much value for functional testing.

In Sprint 0, release planning, or even prior to that, the Product Backlog is built. The backlog consists of stories planned for that release as well as other tasks such as defect fixing and refactoring. The backlog is not static and tends to grow as the project proceeds and more stories are discovered and technical debt is incurred. Any work done in a Sprint should be aligned directly with a Story or Task in the Backlog. That means that every defect report generates a Task. Every instance where refactoring will be needed also generates a Task. Additional Stories are added as the need is discovered as the requirements are further defined.

Because the Backlog is continuously changing, the prioritization of the Backlog must be addressed frequently. This is usually done at the start of each Sprint when the stories are selected for implementation in that Sprint. The Product Owner is responsible for prioritizing the Backlog - with input from the rest of the team - to ensure the product is progressing toward the final deliverable.

Sprint Planning

Sprint planning determines which stories and tasks will be addressed during the Sprint. A Scrum team has a Velocity, which is the number of story points it can implement in a Sprint. Story Points are an indication of the level of effort that will be required to fully implement (and test) a Story. Story points are normally based on the Fibonacci scale (1, 2, 3, 5, 8, 13…) where each number defines the level of effort. This is not directly aligned to a timeframe and is designed to allow the team to determine which stories will "fit" into a Sprint.

Story points are arbitrarily assigned based on the team's understanding of the level of effort required to implement the story. In general, anything at 13 or above should be broken down into smaller stories because estimation becomes less accurate as the level of effort increases. Good estimation takes practice and a team will become more accurate in their estimations as time progresses. This allows the team to become more accurate in setting their Velocity as they learn how much work can be completed during a Sprint.

There are a number of ways in which Story Points are estimated. One of the common ways is to use Planning Poker in which every team member reviews the Story and provides their estimate of the Story Points. Each team member reveals their point estimate at the same time so as not be influenced by others. When there are discrepancies, those are discussed and re-estimation occurs until there is agreement. For example, there could be a wide discrepancy in a story which is simple for the developers (such as changing the format of an address) but can be time consuming to test because of all the places in the application where it is used. Where the developers may think this is a 2, the testers may think this is an 8. More discussion then occurs which might result in everyone agreeing it's a 5.

Once the story points are assigned to the candidates for the Sprint, the prioritization must be considered. Priority is assigned by the Product Owner but may be discussed with the larger team to ensure that both technical risk and impact are understood. For example, a PO may feel that fixing a particular defect is not important for the users, but it might be a critical technical risk for the product (such as potential for data corruption). This is one of the reasons that a proper discussion of the prioritization occurs within the team. This is considered part of the "grooming" exercise

Grooming is done so that the team understands the story, its implications, uses, risk, and priority. In addition to discussing the priority, this is often when the Acceptance Criteria are fleshed out. The Acceptance Criteria define what the code must do to be considered acceptable to the users. The combination of the description of the Story and the Acceptance Criteria fully define the functionality for the team. The priority of the Story determines when it will be implemented. The risk indicates the criticality of testing. The Story Points indicate the level of effort to implement the story. All of this information is added and finalized for the Story during grooming.

Once a Story has been groomed, it is eligible to be added to the Sprint backlog - which is the refined list of Stories and Tasks to be implemented in the Sprint being planned. Note that this is a much smaller set of Stories than the Release Backlog which will contain all the Stories and Tasks. Based on the Team's Velocity, the Stories are selected for the Sprint based on how much effort is available for that Sprint. For example, if the team's Velocity is 30, then the team could implement 10 three-point stories, or six five-point stories, and so on. Because of the limits on what will fit into a Sprint, it's important that the prioritization is correct to be sure the most important Stories are groomed.

Determining the risk of a story, based on the technical risk and potential impact to the user, helps to determine the necessary depth and breadth of testing. In general, the higher the risk, the more thorough the testing. Low-risk stories may be tested with just exploratory testing whereas high-risk stories may require more extensive testing, a wider set of test data, and more testing techniques such as boundary value analysis, decision tables, etc.

In general, a "Sprint Plan" is not a test plan - it's just a list of what is going to be implemented in that Sprint. It's important not to confuse the two as they have quite different meanings, audiences and purposes.

Defining the Requirements

Features, Epics and Stories

Agile requirements are usually classified as follows:

A Feature, which contains Epics, which contain Stories.

A Feature is a high-level requirement that captures a piece of functionality, such as the ability to Buy Travel Insurance. This is then broken down into Epics, such as Sign Up, Manage Insurance Products, Medical Questionnaire, Purchase. These are then broken down into Stories such as Create New Product, Delete Existing Product, Modify Product Price, Present Product List, Compute Medical Score, Accept Credit Card, etc.

A Feature is normally defined in from the end user perspective using common language. Epics and Stories are usually defined in the given/when/then format or the “As a ...., I want to... so I can ...” language. Either way, the requirement is defined from the end user perspective.

The given/when/then format is often used in Behavior-Driven Development (BDD), where the behavior of the software is described. This format is sometimes called Specification by Example. Some of the benefits of this approach include:

- Precise guidance to organize the discussion and grooming
- Because they are written in natural language, anyone can understand what is needed
- This format sets up the requirements for test automation with tools such as Gherkin.

This is a conceptual approach and can be awkward for people to use until they become familiar with it. In some cases, this format is used to define test cases because it allows a higher level of specificity regarding who the user is, what they are accomplishing and how they are getting it done. This helps to define test data and is an easy path to future automation efforts.

An example of the BDD syntax for the Travel Insurance application is:

> Given that I am a new user
> When I click on Buy Travel Insurance
> Then I am asked to enter my personal information

The other common format used to define Stories is the As a... I want to.... So I can.... Using the same example as above, this would be:

> As a new user
> I want to select my travel insurance
> So I can enter my details

In both of these cases, grooming will be needed to define the particulars for implementation and testing. For example, we don't know what personal information is, we don't know what kinds of travel insurance might be available, etc. The looseness of the requirements is a both an advantage and disadvantage. The PO is able to change the details and to supply additional information at story grooming, but the scope is hard to define without knowing those details. This makes the grooming sessions critical to accurate estimations.

Acceptance Criteria

Regardless of the format, the most critical aspect of a story for testing is the proper definition of the acceptance criteria. The acceptance criteria define what will constitute the proper implementation for the story. For the above story, we might see the following acceptance criteria:

1. Travel insurance options are: Comprehensive, Travel Only, Medical Only
2. User details are: Name, Address, DOB, gender, Travel From, Travel To, Dates of Travel
3. Medical questionnaire is covered in story xxx
4. When completing this information, the user is taken to the "Check out" page

This still doesn't provide a high level of detail, but as testing commences, more information can be added such as "what is the proper address format?"

Acceptance criteria are critical in determining the accuracy and correctness of the implementation and testing. Acceptance criteria should cover the following:

- Functional / non-functional requirements (usability, performance, security, etc.)
- Use cases (what scenarios are applicable)
- Business rules (what business rules are applied, such as rejection of certain age groups)
- External interfaces (to what will this software connect, such as the medical questionnaire)
- Constraints (how is personal information stored)
- Data (what is requested, how is it used, how is it passed between applications)

Evaluation of a Story

One quick way to evaluate a user story is to apply the INVEST approach [agilealliance.org/glossary/invest]:

- "I" ndependent of all other stories
- "N" egotiable in that this is not a specific contract for a feature
- "V" aluable to the user
- "E" stimatable by the team
- "S" mall enough to fit in an iteration
- "T" estable based on the information provided

Another evaluation method is called the 3Cs [premieragile.com/3cs-of-user-story]. This model was introduced by Ron Jeffries in 2001 for Extreme

Programming but is still commonly used in Agile projects. The 3 Cs are:

- **Card -** The concept of a story card started when index cards were used to record the story, thus limiting the amount of detail that could be put on the card. This has since been replaced by tools that track the information, but the card image is still prevalent.
- **Conversation -** The conversation is used to take the story and convert it to a deliverable. This is commonly referred to as grooming but includes collaboration of the team, brainstorming and exchanging ideas.
- **Confirmation -** This is the acceptance criteria which define how to determine if the story has been implemented correctly and completely.

While the requirements are generally in the Story form, it is important to remember that there must be agreement between the user (what they want), the developer (what they will implement) and the tester (how they will prove that it "works"). Without this agreement, the implementation will extend, the scope will creep, and the user may not get what they want even after significant effort has been expended.

Testing

Good testing is good testing, regardless of the lifecycle, framework, methodology, or organization. That said, there are some ways to modify testing to make it fit better into the Agile lifecycle.

Test Planning

While Agile projects tend to have less documentation, agreement within the team still must be reached to ensure that the quality levels are achieved. Because of the tighter timelines in an Agile project, it is important that the level of quality is high throughout the development process. The testing levels are still needed but may be compressed to all be performed within a Sprint.

Unit testing is still the responsibility of the developer who wrote the code. Some Agile projects incorporate Test-Driven Development (TDD), which is the process where the developer writes the unit tests before writing the code. The goal is to make the developer think about what must be achieved and how it will be verified, before writing the code. TDD is not commonly used in industry primarily because it requires extensive creation of the unit test framework with drivers and stubs to execute the piece of code being tested. While this leads to higher quality code, if the tests are well-designed and appropriate, it generally takes longer to write the code.

Automation is a strong goal in Agile environments. This is because of the short release cycles and the need to regression test increasingly larger sets of code as Sprints are completed. Automating the unit tests is an important responsibility for the developers - they need to be incorporating their tests into a unit test framework that can be executed with each build. This improves the quality, catches regressions quickly, and improves the test coverage.

Integration testing (testing of individual units working together) is still needed. This is often a split responsibility shared by testers and developers and is ideally automated early in the development cycles. Incorporating this low-level integration testing into the release pipeline is a necessary component of DevOps but is also useful for any Agile project.

System testing is still commonly used in Agile projects and is usually conducted manually by testers on the code that is completed for each Sprint. Automation for this level of testing is usually restricted to automating the regression tests that will be run on each Sprint rather than automating the full functional testing. Theoretically, because only good quality code should be coming through the build and integration process, functional testing should not have to be re-run for each iteration. This, of course, is heavily dependent on the quality of the code and the extent of the regression tests. Any test that is run repeatedly should be automated.

System Integration Testing (SIT) is normally conducted at the Release level rather than the Sprint level. This is because an individual Sprint may not have code that integrates across systems whereas the Release will likely contain APIs that will facilitate communication across applications. Ideally, this testing is conducted within the Sprints that are introducing the cross-system code, but Sprint testing may be conducted in environments that don't contain the necessary integration. SIT testing must be planned at the Release level where multiple teams may be contributing their Sprint products into the overall release.

User Acceptance Testing (UAT) is frequently confused in Agile projects. At the end of each Sprint, there should be a demonstration of functionality to the users. This tends to be quite an isolated view of the software and doesn't usually show end-to-end transactions. True UAT has to be conducted at the Release level where the full functionality is available to the user so that they can validate that they can use the software to accomplish their objectives. Doing UAT at the Sprint level provides valuable early feedback, but it is not a substitute for full UAT at the Release level.

It is important to remember that defects can be found at any level of testing. This means that the earlier a defect is found, the more likely it can be fixed within the Sprint. When defects are not found until Release level testing, this can delay the release or cause the defect fix to roll into the next Release.

In some projects a “Hardening Sprint” is used to provide time to complete the testing and to fix any defects that are found late in the process. While this is a great way to provide adequate time to complete the testing, it should not be used for a significant defect fixing effort (such as pushing all defect fixing into this Sprint) because that will introduce more instability into the system.

Test Documentation

In keeping with the tenets of Agile, working software is favored over documentation, but some documentation will always be needed.

The Requirements Traceability Matrix (RTM) is a necessary matrix to show that testing has covered the right areas to the right level. This can be built into the test management tools by tagging test cases to stories so that the execution of the test cases shows coverage of the stories. This assumes that the right coverage was achieved when the test cases were created, of course. If a particular story needs five test cases to cover the acceptance criteria adequately and only four are executed, then only 80% coverage is achieved. If five are needed and only one is created and executed, the tool will show 100% coverage, but that’s not an accurate measurement of coverage.

Test cases should contain only the minimum amount of information necessary to be repeatable. Exploratory tests with associated objectives (or charters) can be considered to be test cases that are executed against a story. There is no requirement in Agile for test cases to contain detailed steps unless the software itself is of a nature that requires that level of documentation (such as safety-critical software). The goal in Agile is to keep test documentation to the level required - no more. If functional tests will not be re-used, then less detail is needed and a title may be sufficient. If the tests will be used to develop automation, more detail may be needed. The testers need to define the level of detail expected and set those standards during Release planning.

Even though the term “test case” is used here, that does not mean a detailed script. It is an entity used to guide the tester. This could be a mind map that has resulted from a brainstorming session, a goal defined for an exploratory testing session, or a checklist based off the acceptance criteria. There is rarely time in an Agile project to define detailed tests and, given that the requirements may change during development, the more detail that is defined, the higher the maintenance effort for those tests. Practicality needs to reign: the tester should define only to the level needed for the purpose of the test, and no further. Test cases have to be designed for flexibility and the inevitable change that goes with Agile projects.

While most Agile management tools are task trackers, many have a test management component as well. There are two commonly used plug-ins for Jira, Zephyr and X-Ray, which both provide test management capability and allow tests and defects to be linked back to stories. When selecting a test management tool, it's best to ensure it will integrate with the Story definition tool for ease of use, traceability, and consistent and readable reporting.

Deliverables

While one of the goals is to minimize documentation, there are still deliverables required in Agile projects. The primary deliverable is working software and that should always be the primary goal for the team.

To help support the working software and for the team to tune its working practices, it is good to track metrics that can be used to compare releases and to feed into the retrospectives for areas of achievement as well as areas of improvement. It is important that this information be readily available to the team while the release is in progress so that adjustments can be made at the end of each Sprint as needed.

Velocity is the number of story points that should be implementable by the team. Burndown charts are used to show the number of story points estimated vs. the number of story points achieved. This chart is usually tracked in real time and updated when stories are marked as "Done". Story points are only collected at the point of "Done", which means that the points for a story that has failed testing cannot be collected until the failure is resolved. The burndown chart shows if the team is on track to complete the estimated points and allows the team to adjust its velocity to be more realistic over time.

The number of stories implemented is also sometimes tracked - as well as the number of story points. This provides a way to see the accomplishments in completing smaller stories while larger ones are still in progress. Similarly, tracking the size of the backlog is useful to see if technical debt is accumulating. Ideally, the backlog should be consistently reduced, but sometimes completing a story generates more backlog items in terms of required refactoring, defects, and other technical debt. Remember technical debt is not just in code, it could be time required to configure test systems, create test data, etc. When the backlog is rising instead of falling, it's an indication that the Sprint is experiencing unexpected issues.

Defect reports are still needed in Agile projects. While it might be easy to just tell a developer about a problem, that bypasses the concept of the backlog and tracking all tasks to determine velocity. It is important in an Agile project to document all defects, trace them back to the Story, and assign a priority. The developer can then estimate the effort in Story points, and the defect can be incorporated and prioritized into the backlog.

Defect reporting in an Agile project is the same as any other project, reporting trends, priority levels, fix turnaround time, etc. It's important that the test management system be configured with the proper defect fields so that reporting is easy and automatable onto the dashboard.

Retrospectives are more of an activity than a deliverable, but each retrospective should result in a list of what the team does well as well as what needs to be improved. The improvement list should be a discussion item for the next planning session to ensure the same mistakes are not made and that continuous improvement is achieved. Retrospectives are held to allow the team to honestly assess what has been done and how well it worked. These sessions are only valuable if the output is subsequently used as input - otherwise the session is a waste of time. Teams always have areas for improvement. Part of working in an Agile model is continuously improving.

Terms

Acceptance Criteria: The exit criteria that a component or system must satisfy in order to be accepted by a user, customer, or other authorized entity.

Behavior-driven development (BDD): A collaborative approach to development in which the team is focusing on delivering expected behavior of a component or system for the customer, which forms the basis for testing.

Burndown chart: A chart showing the expected velocity compared to the actual velocity of a team across Sprints.

Epic: A large form of a user story which forms the basis of a set of smaller, related user stories. An epic may define a set of identifiable user functionality or an internal capability of the software needed to support user functionality.

Feature: A capability that delivers value and fulfills a user's need. A Feature is usually then further defined by Epics which are further defined by Stories.

Feature flags: A software switch that controls whether the software associated with a particular feature is enabled or disabled.

Hardening sprint: An increment of time concentrating on completing testing tasks and other quality practices after the code for a Release has been completed.

Hub-and-spoke: The practice of setting up teams where there is a hub of capability that is loaned out to a particular project or Agile team. For example, a Testing team may exist as an entity and individual testers are loaned out to projects as needed.

Integration testing: Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.

Kanban board: A tabular form used to visualize work and the workflow and to minimize work in progress.

Planning poker: A consensus-based estimation technique, mostly used to estimate effort or relative size of user stories in Agile software development. It is a variation of the Wideband Delphi method using a deck of cards with values representing the units in which the team estimates.

Product backlog: The list of prioritized functionalities which will form the basis of the product being developed. This is sometimes referred to as the to-do list and may include tasks such as refactoring and defect fixing as well as new development.

Production implementation verification (PIV): A test that is used to verify that the software is working correctly in the production environment.

Product Owner (PO): The person who represents the future product users and interacts with the development team by defining requirements (user stories), prioritizing stories/tasks, and answer questions throughout the development process.

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

Requirements Traceability Matrix (RTM): A tool used to track the relationship between the requirements (stories) and the tests that verify the defined functionality.

Retrospective: A regular event in which team members discuss results, review their practices, and identify ways to improve.

Scrum: An Agile development framework commonly used in industry.

Scrum Master: A person who facilitates an Agile team in implementing and applying Agile practices.

Scrum of scrums: A planning meeting that takes place across multiple scrum (Agile) teams.

Sprint backlog: The tasks and stories that have been selected for implementation within the Sprint.

Stand up: The short daily meeting of the Agile team to discuss what is in progress, what has been completed and to work through any blocking issues.

Story grooming: The process by which a user story is further defined and prioritized by the team.

Story points: The amount of effort assigned to fully implement the given story, including all tasks such as development and testing.

Systems integration testing (SIT): Testing the integration of systems and packages; testing interfaces to external organizations (e.g., Electronic Data Interchange, Internet).

System testing: Testing an integrated system to verify that it meets specified requirements.

Task board: A visual means to track the progress of stories and tasks through the implementation process.

Technical debt: Work that has accumulated during a Sprint or Release which is required to improve the quality and maintainability of the developed software.

Test-driven development (TDD): A way of developing software where the test cases are developed, and often automated, before the software is developed to run those test cases.

Unit testing: Testing usually performed by a developer to ensure that the unit or component of code is working as intended. This testing is often automated via the implementation of a unit testing framework.

User acceptance testing (UAT): Acceptance testing carried out by future users in a (simulated) operational environment focusing on user requirements and needs.

User story: A high-level user or business requirement commonly used in Agile software development, typically consisting of one sentence in everyday or business language capturing what functionality a user needs and the reason behind this, any non-functional criteria, and also includes acceptance criteria.

Velocity: The expected number of story points that a specified team can complete within a Sprint.

References

Works Cited

Agile Alliance. (2023). What does INVEST Stand For? Retrieved from glossary: https://www.agilealliance.org/glossary/invest

Premier Agile. (2023). 3 C's of User Stories. Retrieved from 3 C's of User Story: https://premieragile.com/3cs-of-user-story/

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.

AT*SQA

MICRO-CREDENTIAL

Agile Implementation

www.atsqa.org