Infosys®

Building Tomorrow's Enterprise

Infosys Global Agile Developer Certification

Agile Testing Techniques

Study Material

www.infosys.com

Revision History

Version	Date	Remarks
1.0	Dec’2014	Initial Version

Table of Contents

Table of Figure

1 Introduction

Testing is an integral part of software development and it goes hand in hand with the requirement creation activities. There can be a structured and systematic approach of testing in agile projects too. As there are multiple releases usually defined in Agile projects, testing plays an important role in the quality control of the product or service.

In Agile, emphasis is given to the requirement analysis discussion where team members are expected to work closely with the Product Owner and Business Analyst for the user stories and its acceptance criteria development. Tester's involvement is necessary right from the initiation of the project so that they can actively participate in defining Definition of Done, exploration of User Stories etc. Testers should not remain idle or waiting for requirements. They should follow up on validation criteria in order to know what's expected of a new feature or a change being made in the system. In Agile projects, each team member irrespective of the work, has to become more proactive, enthusiastic, and forward thinking in their interpretation of requirements.

This module introduces how to bring agility into the software testing process. Agility not only improves software testing process but also adds flexibility and enhances ease of use. Thus, agility leads to a 'continuous process improvement', which significantly increases testing volumes because of satisfaction with the process.

2 Concepts

In Agile frameworks, both testing and development are carried out in the same sprint. Once the testing team has a clear idea on what is being developed, tester's work on how the system would behave after the new changes are incorporated and whether it satisfies the business needs or acceptance criteria. In the time when the development team works on the design and code changes, testing team works on the testing approach and comes up with the test scenarios and test scripts. Once the code is ready to be tested, testing team starts the testing which need to be completed by the end of the sprint. This is quite possible in some scenarios when a module of the business need is made available to test. The testing team is expected to work on smaller modules and test them as and when provided rather than the whole functionality to be developed completely.

2.1 Test Strategy and Planning

Test strategy is an important artifact in testing a project. It's the responsibility of the testing team to have a structured and definite testing approach and strategy in the project. Unlike the traditional way, in Agile as the scope of requirement and hence the testing do change very often, it is usually recommended to update the test strategy at the beginning of each sprint. Documentation of this test strategy starts from the Release initiation phase and acts as a living document throughout the Release. Each sprint has its own scope of stories that need to be delivered. As part of the testing team, they can create the test plan. This plan has to be updated in each Sprint and is continued till the release.

Traditionally in software projects, system testing used to begin after development team has completed the integration and then the business owners used to carry out UAT (User Acceptance Testing) after system testing. However, when it comes to Agile all these boundaries are very porous. As an Agile team member, one has to test one functionality at the same time write a test plan for another and at the same time review the design for another story and carry out many similar kinds of activities all at the same point of time. Testers participate in design discussions in Agile projects. This helps them plan their test cases in a better way.

In Agile, towards the end of every sprint, team participates in the Sprint review with Business. In these reviews it's primarily the responsibility of the team members involved in testing to prepare the test data sheets and run the tests in front of the Business Stakeholders, Product Owner and show them how the product is working and whether or not it's meeting their Acceptance Criteria. This is a key process in Agile and helps the team to collect the feedback from the client and other stakeholders on the user stories developed within the sprint. Testing team plays a pivotal role in this process.

2.2 Importance of Automation for Testing in Agile

- Agile needs continuous testing of the functionalities developed in past Sprints. Use of Automation helps complete this regression testing on time.
- It minimizes the rework that can happen due to error prone manual processes.
- It helps bring predictability to the tests run as part of Sprint testing.
- Helps to cover more regression tests than it could have been done manually. This is very much required in Agile as it has higher number of builds and more frequent changes because of its iterative nature.
- Resource is available for other tasks. Instead of retesting existing functionality, testers can put their effort into testing areas of new functionality where a manual intervention is really needed.
- The Continuous regression testing would require huge testing effort since we have to repeat the tests many number of times depending on the iterations. Use of automation helps reduce this effort.

- The smaller automated test cases when aggregated in each sprint, during the end of the project forms an exhaustive regression suite which covers almost all the functionalities from a system, regression and acceptance testing point of view. Thus, it helps in minimizing the cost of the project in the Release Sprint (last Sprint in the Release) - as majority of the testing can be handled through the existing automations.

2.3 Test Coverage in Agile

Considering a limited timeframe for a Sprint to complete the testing, identifying and finalizing the test coverage for the user stories is one of the most important and challenging task in an Agile project. Inadequate test coverage might result in missing of critical tests for some requirements. Test coverage for a user story is usually discussed and finalized during the backlog grooming sessions and later during the Sprint Planning meetings. There are a couple of situations where test coverage might result in missing out a few test scenarios which can be mitigated by creating a traceability matrix between requirements, user stories and test cases. A link can also be created between test cases and user stories.

Another case might arise when a code change is done without complete impact analysis, review and the corresponding changes in test cases needed, could result in incomplete test coverage. This can be mitigated by analyzing the impact, review of source code to identify modules that are to be changed and ensuring that all the changed code has been properly tested. Also use of code coverage tools must be leveraged for verifiable thoroughness as part of the automation effort.

2.4 Test Data Management in Agile

Test data is one of the most important factors in testing. Creating and manipulating the data for testing various test cases is one of the challenging tasks in testing. Agile Testing in Sprints gets difficult when the system is dependent on another team for test data. The Sprint timelines do not give enough room for any delays in the data setup. Setting up the test data is done along the Sprint to make available them at least, 'just in time' (JIT) if not earlier, by the time when the developer checks-in his code for claiming it 'as done'. Since all of the developers usually practice test automation for unit testing and integration testing, and sometimes use the 'test-driven development' approach to complete their user stories, the synchronized supply of the test data by testing team is essential.

The analysis of the data needs is done during the backlog grooming sessions and the Sprint planning meetings to ensure the complete data set up needs. Once the analysis is done, the concerned team(s) is/are reached out accordingly. This also may not be fruitful at times, because if the data is complex and huge, we may not get the adequate support from the team owning the data. Hence the discussion should ideally start with the business, at least a Sprint early or as per the applicable SLA to get the data at the appropriate time.

Also if possible, members (from business) who are responsible for the data should be included as part of the Scrum team for the Sprint where their support is required. This way they get more clarity on the exact data required and the team can gauge/refine their progress.

Moreover, agile being an iterative model, there might be similar user stories in different sprints where the test data created in earlier sprints can either be directly reused or used with minor modifications in order to avoid redundant data requests/set up and reduced turnaround time for the test data creation and manipulation.

2.5 Impact Analysis in Agile

In the traditional practice, Team used to go through the Business requirements and develop the test scenarios in isolation. They participate in design reviews only when it was completed by development teams. Hence contribution from the testing teams in those reviews were minimal. However in Agile, there is a more significant role of testing team in impact analysis and design reviews as they are involved in the design discussions happening for a story. This helps the developer in the impact analysis and debugging the issues. Many a times, testing team members go to the code level to analyze or debug the issue along with the developer. In agile, because of the short timeframe, it is the responsibility of the entire team to ensure that things are developed correctly from day one and thus contribute to successful delivery.

3 Practices

3.1 Types of Testing

Types of Testing	Description
Unit Testing	In Agile, the unit testing is part of every Sprint for every story being developed. Unit testing allows developers identify as many problems as possible at an early stage. Doing it in a repeatable and automated fashion helps them make more frequent and stable code changes in the iterative world of Agile development. Developers use different tools to develop automated test cases depending on the framework used. JUnit, NUnit, Jmock etc. are some tools to effectively and efficiently test the code being developed. In Agile, automation of unit testing is a must.
Integration Testing	Integration testing is done to ensure all the integration points are working fine and each system is interacting with each other correctly. Traditionally, this is conducted after all the units created and tested by developers and just before the system is handed over to the QA (Quality Assurance) team for testing. In Agile, integration testing is iterative and continuous in nature. After few units or functions are developed and tested by the team, if there is a scenario where integration of systems have to be done, then that story is planned for the sprint on priority and integration testing is performed by running tests that were done in isolation previously. A project team should have outlined the capacity for integration testing based on current scenario of the project to seize any opportunity for integration. As a best practice, the team should create stories and functions to integrate and plan tasks to carry out the sequence of the integration of user stories to complete integration testing in every sprint. For example, imagine a new data feed has to be sent from System A and consumed by System B. There will be stories to create a feed from System A. There will be stories to consume the feed in System B. Now both these work happen in isolation. After these are tested by the team, there should be one more story to actually integrate the two systems for the new feed and test the same. This helps them to address any integration issues between the two systems. To summarize, in Agile it is not expected to wait till the end of development phase to carry out Integration testing.
Smoke Testing	Smoke testing is usually done to confirm whatever is dropped to QA for testing is good and can move ahead with system testing. In smoke testing, a subset of system test cases are used for targeting some of the critical features. This normally happens once in a release path prior to the start of system testing. In Agile, Smoke testing has a very high importance in scenario where there are multiple release paths catering to different releases. It happens

	after every code backfill into the current test environment from the release path. In some cases smoke testing is also carried out after a code drop, by all team members irrespective of whether he is developing or testing. For better usage of time, since code drops happen very frequently in Agile, there should be a base-lined set of test scripts for Smoke testing in a particular sprint. This set of test scripts/cases should be distributed to all the team members irrespective of their work. This way when a code drop happens, the team members can complete the smoke testing quickly.
System Testing	Agile is iterative in nature and so is the system testing. In Agile, a requirement for a single feature is broken down into granular features then transformed to stories. This is to ensure that a story can be developed within the Sprint timelines. Hence testing also happens for those stories as and when they are developed. There can be scenarios where the system testing of a story needs a completion of another story. Hence prioritization of stories keeping an eye to the end product and the testing needs is critical in Agile. Since it's iterative in nature, the same scripts have to be executed as and when different interlinked stories are developed. The system test scripts have to be designed in line with the Story under test and not the complete feature. However as a good practice, we need to ensure that the scripts are designed in such a way that they require minimum modification when the end product/feature is ready for testing.
Regression Testing	Regression Testing is one of the key activity to ensure that all the existing functionalities are working fine. This is done prior to the release of new changes into production basically towards the end of system testing. In Agile projects, Regression should be done in every Sprint to ensure that all the previous functionalities developed till the previous sprints are working fine. Hence having an automated suite of test cases is important to help the team with their regression testing. Unless there is automation, regression testing objective to cover all prior Sprint functionalities in a given sprint cannot be realized in Agile. For maintenance kind of projects, regression testing of all the existing functionalities is carried out in the last Sprint prior to the release (also known as the Hardening Sprint or Release Sprint). As a best practice, based on the volume of changes, team should plan to execute the existing regression suite in small portions in every sprint before the Release or Hardening sprint. This way, the team would avoid maximum issues in the Release or Hardening sprint and minimize the risk to the Release.
Production Implementation Testing	When the product is released to production or to the end users of the product, the development and the testing team is involved in the implementation. The testing team prepares a checkout test plan and list the cases to be tested once the code is implemented in production. During this phase, if the team finds something which is not working as expected, they fix it immediately, thereby by ensuring quality of the product. In Agile, the concept and objective remains the same except for the fact that in the last sprint prior to the release, the capacity for the team should

	have factored in the activities that has to be done for the implementation starting from deployment to testing the changes in production and carrying out the required preparedness for the same.
Performance Testing	Performance testing is done to determine the behavior of the system at a particular load and the breakpoint of the system. It is generally recommended to be done at the release sprint after the functional testing is completed. However, if the performance is a critical aspect of the project and there is high risk involved when performance issue identified may lead to significant changes in the application, Team is advised to run performance testing after few iterations. Team needs to plan and work on the performance testing couple of Sprints prior to the Sprint where performance testing is planned. This helps to setup environment for the performance tests. The Team must be aware of the 'Done' criteria of performance testing upfront. This is applicable for many of NFR (Non-Functional Requirements) tests like availability, security, scalability etc. and not just limited to performance testing only.
Exploratory Testing	Exploratory testing can be stated as an unorthodox means of testing with no specific plan or testing approach. It is all about exploring and identifying what is working and what is not in the software. Here the plan changes constantly on what needs to be tested and where to focus more. The time is a key factor in this testing. This is one of the important testing techniques in Agile. In Agile, most of the standard test cases for the Sprint Stories are automated. Even though Automations capture the majority of functionalities, they won't be able to identify the usability, reliability, compatibility and other quality areas. Hence a manual verifications that need to happen to ensure the product is behaving as expected. This is where exploratory testing becomes handy, where testers identify areas that can have issues or impacts and then design manual test cases, run the tests, analyze the results and decide on the next phase of test cases to be run. Another theory on exploratory testing is its benefits during the story development. During the story development asking the right set of questions can bring more clarities to the acceptance criteria and help developers to design the product in a right way. For example, suppose a web page is getting developed and as part of a story there must be all the fields that need to be present in the web page. The story normally won't have the details of the upper and lower limits of all the data entry fields, the browsers it is supported or the load it can take. However all those questions bring the right kind of clarity to the story and hence story can be developed in the right way from day one.

Client/User Acceptance Testing	Any defects identified during the UAT could lead to significant impacts to the project cost and schedule. In Agile, there are regular Sprint Reviews (at the end of the Sprint) where team get inputs from the Business Stakeholders on the product that is being developed. Also a formal User Acceptance Testing is planned in the Release Hardening Sprint when all functionalities are fully developed. The chances of identifying defects during this phase gets low because of the regular feedbacks received as part of Sprint Reviews.

3.2 Defect Management in Agile

There is a myth that defect management is not needed in Agile. However, it is important to understand on how does it works when there is a bug which needs more effort and could not be fixed in the stipulated time of a sprint. In such cases, defect management is needed in agile projects too. Whenever the stories are available to test, the testing team starts testing and finds bugs. The usual practice is, if the bug or the issue can be fixed in a day, defects are not raised, and the development team is communicated to work on the issue. If the issue which is found is not fixed in a day, then a defect is raised to make the team and the relevant stake holders aware of the issue. This has to be tracked formally in a defect management tool. The critical defects are discussed with the team along with Product Owner who finalizes on the criticality of the defect and the criticality of the story under which the defect has been logged.

Following is the depiction of how Defect management is done in Agile:

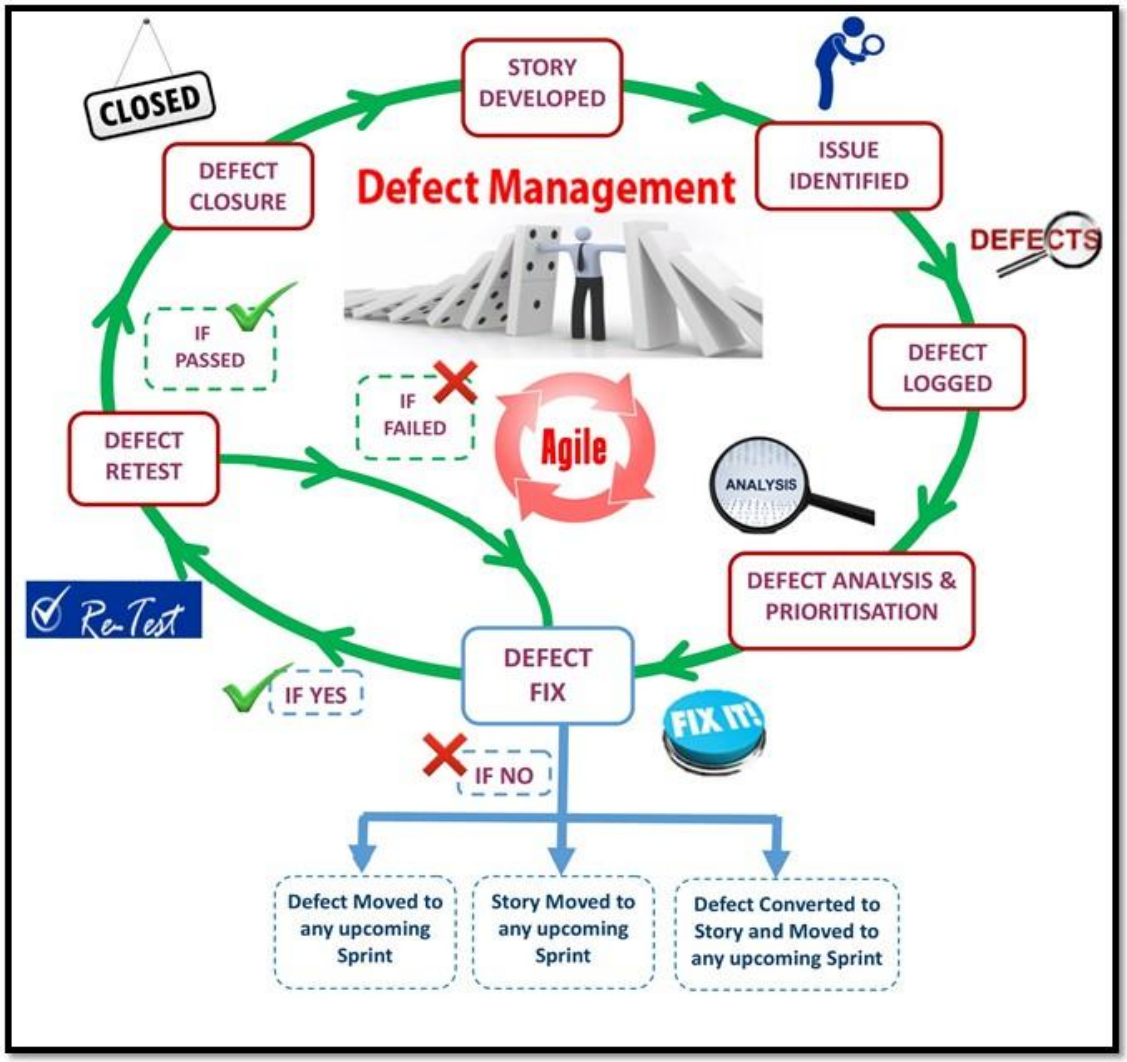

Figure 1: Defect Management Flow within a Sprint

The defects are prioritized on the following basis:

- Impact on the test coverage
- Impact on meeting the acceptance criteria of the story

What if a defect comes up late during the sprint or some defect which is directly not related to any of the stories taken up in the sprint? In such cases, a defect is raised and discussed whether it can be fixed in the same sprint and also how much of effort is required to fix and retest it. If the effort is too high and if forecasted that it cannot be delivered as a part of the current Sprint, it is either moved to a story related to the defect which is then planned in future Sprint, or it is converted to a new story and gets assigned to a future sprint.

It is very important to capture the details of how effective the testing techniques are which were deployed by the team. Following are few of the suggested metrics which the team can track and analyze as part of the defect management activities:

3.2.1 Defect Removal Effectiveness (DRE)

Defect removal effectiveness is a metric that tells that how many defects testers could identify in their testing and how many of them got slipped to next stage i.e. UAT or Production or Warranty period etc. DRE is expressed as %.

$$Defect\ Removal\ Effectiveness\ (DRE) = (No:\ of\ In\text{-}process\ defects\ /\ Total\ no.\ of\ defects) * 100$$

(Total number of defects = In-process defects + Defects found in Sprint/Release review)

3.2.2 Test Coverage (using code coverage)

One important parameter in testing is "How much of the source code is actually exercised by my test system?" Unexercised code can mean that the test system is deficient OR the source has "dead code" (code that cannot be reached by any path), OR extra code is in the codebase that is not necessary (perhaps a feature was removed but not all of the associated code was deleted).

When the User Stories are tested using Black box technique, the team does validation to compare if the output is correct for given inputs or not. But when the focus is on Test Coverage, testers can look inside the Black box code and analyze how much of the code was actually exercised by the test set. The team can do this with Test Coverage tools available commercially like 'Semantic Design' test coverage tool, 'Bullseye Coverage' etc. If there is a focus on test coverage, it also expands view of Black box testing to White box testing. This is how a high level process is:

- Compile the code with test coverage tool integrated
- Use that code to test your test cases
- After testing is complete, check the tool report to see the code coverage achieved by tests that were just executed
- Based on the report and discussion with developers, testers should take decision to add/update test suit to have additional coverage of code where the tests did not even reach to. The tool report throws reports on which paths/classes/branches/files were covered when the software was executed to run the test suit.
- Based on the report and discussion with developers, testers must decide if additional/updated tests to be created so as to have better code coverage through the tests and surety that each corner of the code is tested by your tests
- Many a times these tools provide additional details than just % code coverage. They are Product Quality Metrics (PQM) like memory issues, dead code, clone identification etc. That eventually helps developers in cleaning and optimizing code further.

Thus using code coverage tools help testers to understand what part of code got tested and what more needs to be tested. And it helps developer to fine tune and refactor the code further.

Simple formula:

$$Test\ Coverage\ \% = \%\ of\ Code\ covered\ through\ the\ Test\ Suit.$$

Test coverage targets should be 90% to 100%

3.2.3 Automation Test Coverage

From a testing standpoint, Automation of test coverage provides an insight to the percentage of test cases being automated as a part of the project. Each Sprint covers certain test cases to be automated which can be run in an automated fashion in the future Sprints. In this way, the number of test cases being automated increases Sprint by Sprint reducing the number of manual test cases thereby providing higher percentage of automation which can cover system cases during release level testing, rather manually executing the cases.

$$\boldsymbol{Automation\ test\ coverage\ metrics} = \left(\frac{\boldsymbol{Number\ of\ automation\ test\ cases}}{\boldsymbol{total\ number\ of\ test\ cases}}\right) * \mathbf{100}$$

Team having a higher percentage of automation can concentrate more on exploratory and ad-hoc testing so that the regular test cases can be covered through automation.

3.2.4 Automation Coverage Effectiveness

The effectiveness of Automation is used to define how effective the Automation test cases are by having the number of defects caught through automation testing. The more the defects caught through running automation test cases, higher is the effectiveness. The team works in identifying the components which could be modified so as to identify more defects.

$$\boldsymbol{Automation\ coverage\ effectiveness} = \left(\frac{\boldsymbol{Number\ of\ defects\ caught\ by\ automation}}{\boldsymbol{total\ number\ of\ defects\ caught}}\right) * \mathbf{100}$$

With each Sprint, team can work on the defects caught manually in the earlier sprint to incorporate the changes in automation so that the effectiveness of the automation test cases increase. If the effectiveness remains the same/decreases, then the team need to strive to have some changes in the automation testing strategy.

4 Challenges in Testing and Suggested Solutions

Following are the challenges commonly faced while testing in Agile projects and the suggested solutions to address the same:

Challenge	Description	Suggested Solution
Code broken accidently due to frequent Builds	Since code gets changed and compiled daily, the likelihood of code breaking the existing features is much higher	Team should have a smoke test bed created that needs to be executed after every code build to ensure no major hiccups. As resource constraints prevails, so it is not practical to have team members do this daily thereby having efficient automated test cases built to confirm the environment stability after code drops.
Performance Testing Bottlenecks	Agile involves regular builds and subsequent testing, and hence environment availability becomes difficult for the performance testing activities.	Performance testing is mandatory for projects undergoing changes which have architectural impacts. It is very essential for Agile projects to carry out performance testing. To overcome the challenge, team should always have a separate environment for performance testing. The setup should be backfilled with the correct code base and then a round of smoke testing has to be done on the same. There should be separate stories to track the performance testing progress.
Wrong selection of Sprint to start Automation of test cases	It's always a challenge to decide on the correct Sprint for starting with the automation of manual test cases. If Automation is planned at the early stages of the Release (i.e. from the initial Sprints onwards), it leads to a lot of wastage of effort as the team is not settled properly and the output is meagre to do any automation. This leads to the automation work getting redone in subsequent Sprints, leading to wastage of effort. In case of late start of automation (i.e. during the last one or two Sprints in the Release), the team would have more manual test cases than automated ones. This would significantly impact the team's velocity and delivery capabilities.	Although it varies from project to project, but planning the automation ideally after 3 Sprints is a good idea, since by then there would have been significant application development and stability to carry out the automation work.
Wastage of Automation scripts after couple of sprints.	There is always a risk of lot of rework and wastage of automation scripts in case they are not developed properly. With change in	Automation scripts should have high reusability factor with data sheets driving the automation execution. This way team can reduce the code changes to the automation

	project course is quite normal in Agile, automations would need modifications which may impact its ROI (Return on Investment).	scripts for any change in the Business functionality. It's an industry wide best practice, but required more in Agile since it's prone to more frequent changes.
Wrong selection of Automation Tool	In Agile, User Stories get re-prioritized and nothing is constant if compared on a day to day basis. Hence if the Automation Tools are not selected properly, there is a chance of the project team running into higher costs and less ROI for the automation.	For Agile Teams, it is always recommended to go for Open Source Tools without any licensing costs. This would give automaton access to all the team members ensuring better/higher outputs at low cost. However for projects, where there are members dedicated for Automation work, tools should be selected based on the kind of project being developed. Ex: A tool with Command Line Support is perfect for projects which involve testing for backend systems rather than User Interface.

5 Best Practices

- **Project/Release Initiation Phase**:

  - Testers should get involve right from the beginning of the Project initiation to understand the requirements better
  - Understand the high level requirements and highlight the issues (if any) in the very early phase of the project
  - Provide inputs to Product Owner or Business Analyst while setting business priorities in Product Backlog
  - Understand the business priorities and help in the Release Planning

- **Project/Release Planning and Sizing**:

  - Both, developers as well as testers should participate in the Product Backlog grooming sessions
  - Developers and Testers should together participate during Planning Poker Game which helps in accurate estimates and Story Sizing
  - Customized Estimation Template is used to capture defect rate, Injected defect rate and rounds of Regression Cycle based on past agile project experiences

- **Sprint Planning**:

  - Testers should mandatory attend the Requirement walkthrough sessions to capture more number of requirement/design findings & production issues in early phase
  - It is always recommended to revisit and revise estimates after Sprint level requirement analysis
  - Modular approach for Test Planning and Scripting should be adopted
  - Test scenario walkthrough with the team at end of planning phase will always help every member to get a broader idea

- **Sprint Execution**:

  - Consolidated/modular approach for Test Data setup activities
  - Test data repository to increase the reusability
  - Metrics Based test execution for better tracking and coverage
  - Risk Based testing in the critical situations
  - “Sprint Traceability Metrics” to maximize the test coverage
  - Agile Review Checklists to increase the quality of deliverables
  - Strong collaboration with waterfall team for workload balance during crisis

6 Practice Questions

Question 1

Which of the following is FALSE about Defect Management in Agile?

a) Defects are logged in a tool when identified
b) Defects are analyzed and fixed based on prioritization
c) Defects are re-tested before closure
d) All of the above
e) None of the above

Question 2

When is Integration Testing done in Agile projects?

a) Integration Testing is done once all the user stories are completed and accepted by the Product Owner
b) Integration Testing is only done during the hardening sprint of the Release
c) Integration Testing is done within every Sprint of the Release as the functionality gets developed
d) Integration Testing is done once at the mid of the Release and then at the end of the Release

Question 3

Why is Smoke Testing needed in Agile projects?

a) To ensure that the bad code is not dropped into the testing/production environment
b) To save time within the Sprint so that effort is diverted towards the development of the stories
c) To bypass system testing whenever there is a time constraint before demonstrating the incremental software
d) Smoke testing is not needed in Agile projects as it is just an additional testing activity

Question 4

Which of the following is NOT a challenge while Testing in Agile projects?

a) Identifying the Sprint for starting the automation of Test cases
b) Frequent builds breaking the existing features
c) Environment availability for Performance testing
d) Analyzing the effectiveness of automation

Question 5

In Agile, both testing and development is carried out in same Sprint duration. TRUE or FALSE?

a) TRUE
b) FALSE

7 References

- Lisa Crispin and Janet Gregory, "*Agile Testing: A Practical Guide for Testers and Agile Teams"*

Websites:

- http://www.mountaingoatsoftware.com/reviews/agile-testing
- http://www.ambysoft.com/essays/agileTesting.html
- http://www.softwaretestinghelp.com/category/agile-testing/
- http://lisacrispin.com/downloads/AdpTestPlanning.pdf
- http://www.testertroubles.com/2009/11/exploratory-testing-agile-approach.html

www.infosys.com