What is Agile Testing?

Agile Testing is a testing practice that follows the rules and principles of agile software development. Unlike the Waterfall method, Agile Testing can begin at the start of the project with continuous integration between development and testing. Agile Testing methodology is not sequential (in the sense it's executed only after coding phase) but continuous.

Principles of Agile Testing

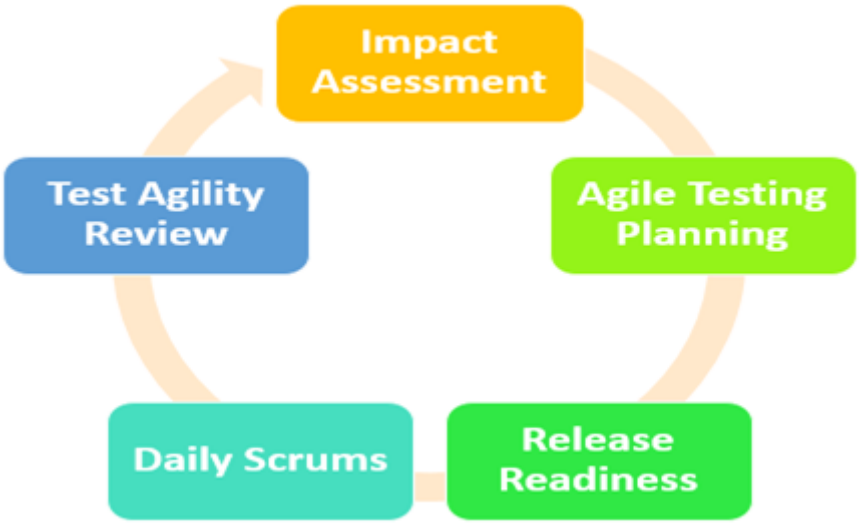
Here are the essential Principles of Agile Testing:

- In this Agile testing model, working software is the primary measure of progress.
- The best result can be achieved by the self-organizing teams.
- Delivering valuable software early and continuously is our highest priority.
- Software developers must act to gather daily throughout the project.
- Enhancing agility through continuous technical improvement and good design.
- Agile Testing ensures that the final product meets the business's expectations by providing continual feedback.
- In the Agile Test process, we need to execute the testing process during the implementation, which reduces the development time.
- Testing process in Agile should work on the consistent development pace
- Provide regular reflections on how to become more effective.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- Each time the team meets, it reviews and adjusts its behavior in order to become more effective.
- Face-to-face conversation with the development team is the most effective and efficient method of conveying information within the team.

Agile Testing includes various principles that help us increase the Software's productivity.

Agile Testing Life Cycle

The agile Testing life cycle is completed in five different phases, as we can see in the following image:



Here are the Agile process testing steps:

Phase1: Impact Assessment: In this initial phase, we gather inputs from stakeholders and users. This phase is also called the feedback phase, as it assists the test engineers in setting the objectives for the next life cycle.

Phase 2: Agile Testing Planning: It is the second phase of the Agile testing life cycle, where all stakeholders come together to plan the schedule of the testing process and deliverables.

Phase 3: Release Readiness: At this stage, we review the features that have been developed/ Implemented are ready to go live or not. In this stage, it is also decided which one needs to go back to the previous development phase.
Phase 4: Daily Scrums: This stage includes every standup morning meeting to catch up on the status of testing and set the goal for the entire day.
Phase 5: Test Agility Review: The last phase of the Agile life cycle is the Agility Review Meeting. It involves weekly meetings with stakeholders to regularly evaluate and assess progress against goals.

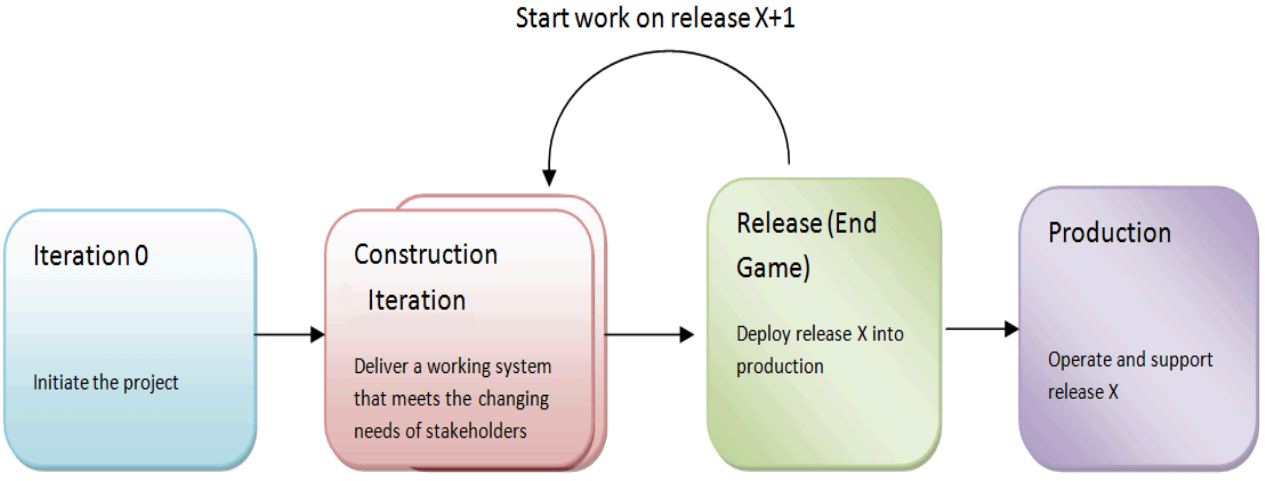
Agile Test Plan

Agile test plan includes types of testing done in that iteration like test data requirements, infrastructure, test environments, and test results. Unlike the waterfall model, in an agile model, a test plan is written and updated for every release. Typical test plans in agile includes

- Testing Scope
- New functionalities which are being tested
- Level or Types of testing based on the features complexity
- Load and Performance Testing
- Infrastructure Consideration
- Mitigation or Risks Plan
- Resourcing
- Deliverables and Milestones

Agile Testing Strategies

Agile testing life cycle spans through four stages



Agile Testing Strategy

Iteration 0

During the first stage or iteration 0, you perform initial setup tasks. It includes identifying people for testing, installing testing tools, scheduling resources (usability testing lab), etc. The following steps are set to achieve in Iteration 0

- Establishing a business case for the project
- Establish the boundary conditions and the project scope
- Outline the key requirements and use cases that will drive the design trade-offs
- Outline one or more candidate architectures
- Identifying the risk
- Cost estimation and prepare a preliminary project

Construction Iterations

The second phase of agile testing methodology is Construction Iterations, the majority of the testing occurs during this phase. This phase is observed as a set of iterations to build an

increment of the solution. In order to do that, within each iteration, **the team implements** a hybrid of practices from XP, Scrum, Agile modeling, and agile data and so on.
In construction iteration, the agile team follows the prioritized requirement practice: With each iteration, they take the most essential requirements remaining from the work item stack and implement them.
Construction iteration is classified into two, confirmatory testing and investigative testing. **Confirmatory testing concentrates** on verifying that the system fulfills the intent of the stakeholders as described to the team to date, and is performed by the team. While the investigative testing detects the problem that confirmatory team has skipped or ignored. In Investigative testing, tester determines the potential problems in the form of defect stories. Investigative testing deals with common issues like integration testing, load/stress testing, and security testing.
Again for, confirmatory testing there are two aspects **developer testing** and **agile acceptance testing. Both of them** are automated to enable continuous regression testing throughout the lifecycle. Confirmatory testing is the agile equivalent of testing to the specification.
Agile acceptance testing is a combination of traditional functional testing and traditional acceptance testing as the development team, and stakeholders are doing it together. While developer testing is a mix of traditional unit testing and traditional service integration testing. Developer testing verifies both the application code and the database schema.

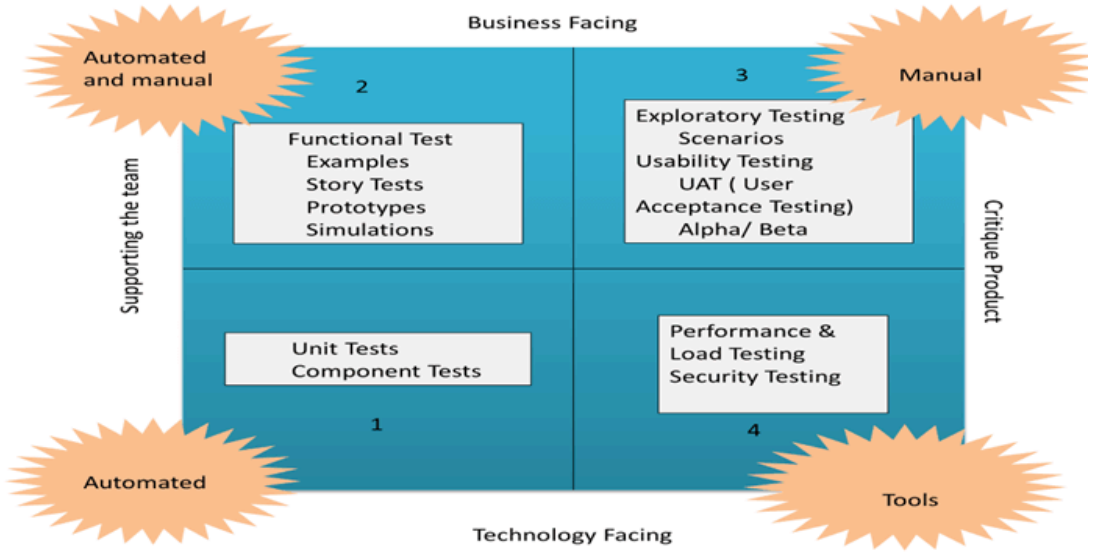
Release End Game Or Transition Phase

The goal of "Release, End Game" is to deploy your system successfully into production. The activities include in this phase are training of end users, support people and operational people. Also, it includes marketing of the product release, back-up & restoration, finalization of system and user documentation.
The final agile methodology testing stage includes full system testing and acceptance testing. In accordance to finish your final testing stage without any obstacles, you should have to test the product more rigorously while it is in construction iterations. During the end game, testers will be working on its defect stories.

Production

After the release stage, the product will move to the production stage.

The Agile Testing Quadrants



The agile testing quadrants separate the whole process in four Quadrants and help to understand how agile testing is performed.

Agile Quadrant I

The internal code quality is the main focus in this quadrant, and it consists of test cases which are technology driven and are implemented to support the team, it includes

- Unit Tests
- Component Tests

Agile Quadrant II

It contains test cases that are business driven and are implemented to support the team. This Quadrant focuses on the requirements. The kind of test performed in this phase is

- Testing of examples of possible scenarios and workflows
- Testing of User experience such as prototypes
- Pair testing

Agile Quadrant III

This quadrant provides feedback to quadrants one and two. The test cases can be used as the basis to perform automation testing. In this quadrant, many rounds of iteration reviews are carried out which builds confidence in the product. The kind of testing done in this quadrant is

- Usability Testing
- Exploratory Testing
- Pair testing with customers
- Collaborative testing
- User acceptance testing

Agile Quadrant IV

This quadrant concentrates on the non-functional requirements such as performance, security, stability, etc. With the help of this quadrant, the application is made to deliver the non-functional qualities and expected value.

- Non-functional tests such as stress and performance testing
- Security testing with respect to **authentication** and hacking
- Infrastructure testing
- Data migration testing
- Scalability testing
- Load testing

QA challenges with agile software development

- Chances of error are more in agile, as documentation is given less priority, eventually puts more pressure on QA team
- New features are introduced quickly, which reduces the available time for test teams to identify whether the latest features are according to the requirement and does it truly address the business suits
- Testers are often required to play a semi-developer roled
- Test execution cycles are highly compressed
- Very less time to prepare test plan
- For regression testing, they will have minimal timing
- Change in their role from being a gate-keeper of quality to being a partner in Quality
- Requirement changes and updates are inherent in an agile method, becoming the biggest challenge for QA

Risk of Automation in Agile Process

- Automated UI provides a high level of confidence, but they are slow to execute, fragile to maintain and expensive to build. Automation may not significantly improve test productivity unless the testers know how to test

- Unreliable tests are a major concern in automated testing. Fixing failing tests and resolving issues related to brittle tests should be a top priority in order to avoid false positives
- If the automated test are initiated manually rather than through CI (Continuous Integration) then there is a risk that they are not regularly running and therefore may cause failing of tests
- Automated tests are not a replacement for an exploratory manual testing. To obtain the expected quality of the product, a mixture of testing types and levels is required
- Many commercially available automation tools provide simple features like automating the capture and replay of manual test cases. Such tool encourages testing through the UI and leads to an inherently brittle and difficult to maintain tests. Also, storing test cases outside the version control system creates unnecessary complexity
- In order to save time, much times the automation test plan is poorly planned or unplanned which results in the test fail
- A test set up and tear down procedures are usually missed out during test automation, while Performing manual testing, a test set up and tear down procedures sounds seamless
- Productivity metrics such as a number of test cases created or executed per day can be terribly misleading, and could lead to making a large investment in running useless tests
- Members of the agile automation team must be effective consultants: approachable, cooperative, and resourceful, or this system will quickly fail
- Automation may propose and deliver testing solutions that require too much ongoing maintenance relative to the value provided
- Automated testing may lack the expertise to conceive and deliver effective solutions
- Automated testing may be so successful that they run out of important problems to solve, and thus turn to unimportant problems.

What is Test Driven Development (TDD) : Approach & Benefits

The evolution of Agile development has introduced many pragmatic practices for delivering quality software at high speed. Test-Driven Development (TDD) is one such practice that is now recognized as an efficient approach that drives positive results.

What is Test Driven Development (TDD)?

In layman's terms, Test Driven Development (TDD) is a software development practice that focuses on creating unit test cases before developing the actual code. It is an iterative approach that combines programming, the creation of unit tests, and refactoring.

The TDD approach derives its roots from the Agile manifesto principles and Extreme programming. As the name suggests, the test process drives software development.

Moreover, it's a structuring practice that enables developers and testers to obtain optimized code that proves to be resilient in the long term.

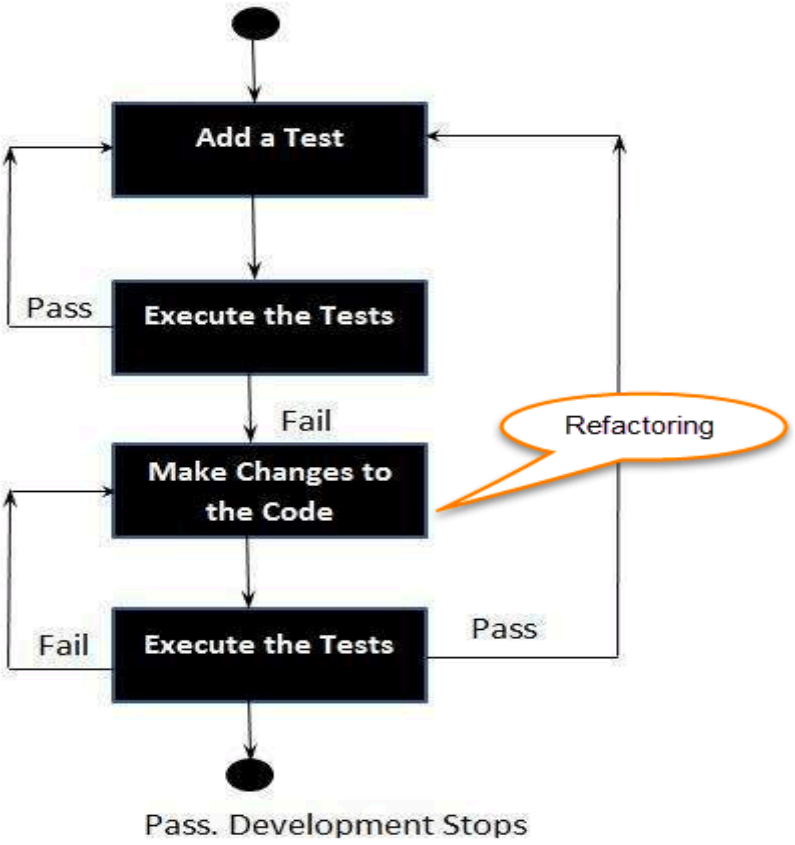
In TDD, developers start creating small test cases for every feature based on their initial understanding. The primary intention of this technique is to modify or write new code only if the tests fail. This prevents duplication of test scripts.

Three phases of Test Driven Development

1. **Create precise tests:** Developers need to create precise unit tests to verify the functionality of specific features. They must ensure that the test compiles so that it can execute. In most cases, the test is bound to fail. This is a meaningful failure as developers are creating compact tests based on their assumptions of how the feature will behave.
2. **Correcting the Code:** Once a test fails, developers need to make the minimal changes required to correct the code so that it can run successfully when re-executed.

3. **Refactor the Code:** Once the test runs successfully, check for redundancy or any possible code optimizations to enhance overall performance. Ensure that refactoring does not affect the external behavior of the program.

The image below represents a high-level TDD approach towards development:



How TDD fits into Agile development?

Agile development demands regular feedback to develop the expected product. In simple terms, one can also term Agile development as Feedback Driven Development.

There's a high probability that project requirements may change during the development sprint cycle. To deal with this and to build products aligned with the client's changing requirements, teams need constant feedback to avoid dishing out unusable software. TDD is built to offer such feedback early on.

TDD's test-first approach also helps mitigate critical bottlenecks that obstruct the quality and delivery of software. Based on the constant feedback, bug fixes, and addition of new features, the system evolves to ensure that everything works as intended. TDD enhances collaboration between team members from both the development and QA teams as well as with the client. Additionally, as the tests are created beforehand, teams don't need to spend time recreating extensive test scripts.

Benefits of Test Driven Development (TDD)

1. Fosters the creation of optimized code.
2. Helps developers better analyze and understand client requirements and request clarity when they are not adequately defined.
3. The addition and testing of new functionalities become much easier in the latter stages of development.
4. Test coverage under TDD is much higher compared to the conventional development models. This is because the TDD focuses on creating tests for each functionality right from the beginning.

5. Enhances the productivity of the developer and leads to the development of a codebase that is flexible and easy to maintain.

Frameworks for Test Driven Development

Based on unique programming languages, there are multiple frameworks that support test driven development. Listed below are a few popular ones.

1. **csUnit and NUnit** – Both are open source unit testing frameworks for .NET projects.
2. **PyUnit and DocTest:** Popular Unit testing framework for Python.
3. **Junit:** Widely used unit testing tool for Java
4. **TestNG:** Another popular Java testing framework. This framework overcomes the limitations of Junit.
5. **Rspec:** A testing framework for Ruby projects

The process of delivering quality products requires not just debugging but also demands optimization in the development process. When incorporated correctly, the TDD approach provides numerous benefits, particularly in terms of bringing cost-efficiency in the long run and delivering true value to businesses.