DUY TAN UNIVESITY

SOFTWARE TESTING

**AGILE SOFTWARE DEVELOPMENT AND TESTING**

Teacher : Dr. Anand Nayyar

Student : Tuan, Pham Minh

Class : K22MCS

Da Nang, 01/2021

**Table of Contents**

[LIST OF FIGURES 1](#_Toc65314737)

[INTRODUCTION 2](#_Toc65314738)

[Chapter 1:](#_Toc65314739) [AGILE SOFTWARE DEVELOPMENT AND TESTING 3](#_Toc65314740)

[1. AGILE SOFTWARE DEVELOPMENT 3](#_Toc65314741)

[1.1. What is Agile Software Development? 3](#_Toc65314742)

[1.2. Agile Manifesto 4](#_Toc65314743)

[1.2.1. The history of the Agile Manifesto 4](#_Toc65314744)

[1.2.2. The content of the Agile Manifesto 5](#_Toc65314745)

[1.2.3. Advantages and Disadvantages of Agile 8](#_Toc65314746)

[1.2.4. What is Scrum? 11](#_Toc65314747)

[2. AGILE TESTING 13](#_Toc65314748)

[2.1. What is Agile Testing? 13](#_Toc65314749)

[2.2. Agile Testing Principles 14](#_Toc65314750)

[2.3. Agile Testing Methods 14](#_Toc65314751)

[2.4. Agile Testing Quadrants 18](#_Toc65314752)

[2.5. Agile testing Life-cycle 21](#_Toc65314753)

[Chapter 2:](#_Toc65314754) [AGILE TESTING IN CASE STUDY 24](#_Toc65314755)

[1. How you apply Agile Testing Life Cycle in Development 24](#_Toc65314756)

[2. How you prepare Test Cases. 28](#_Toc65314757)

[3. How you perform Agile Testing for Error Reporting and Analysis 29](#_Toc65314758)

[CONCLUSIONS 35](#_Toc65314759)

[REFERENCES 36](#_Toc65314760)

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **No.** | **Title** | **Page** |
| Figure 1.1 | Agile Model | 3 |
| Figure 1.2 | Waterfall Model | 4 |
| Figure 1.3 | The success rate of Agile Model | 10 |
| Figure 1.4 | The components of Scrum process | 13 |
| Figure 1.5 | Behavior Driven Development (BDD) Method | 15 |
| Figure 1.6 | Acceptance Test Driven Development (ATDD) Method | 16 |
| Figure 1.7 | Exploratory Testing Method | 17 |
| Figure 1.8 | Agile Testing Quadrant | 19 |
| Figure 1.9 | Agile Testing Life Cycle Process | 22 |
| Figure 2.1 | Agile Testing Strategy | 25 |
| Figure 2.2 | Defect Management Process | 29 |
| Figure 2.3 | Agile Bugs Prioritisation | 32 |

# INTRODUCTION

Life changes rapidly, technology has always been the launchpad for different areas of society to follow. Increasingly modern technology, faster computers, outdated methods, and technologies have no place but instead are new solutions. Software developers are also not out of the trend to be forced to change and find new solutions that are more suitable for the new era.

Traditional software development methods expose more and more weaknesses and high project failure rates during the technology boom. Recognizing that problem, some individuals and individual companies have come up with more modern and different software development methods to adapt to the new situation.

Not long ago, the world was still infatuated with the Waterfall process, a slightly rigid but accessible traditional process for software development. In the current development trend, this process is no longer suitable, but tends to gradually shift to Agile.

The content of the essay briefly introduces Agile Software Development and Agile Testing is presented with the following 2-chapter layouts:

Chapter 1: Introduction about Agile Software Development and Agile Testing

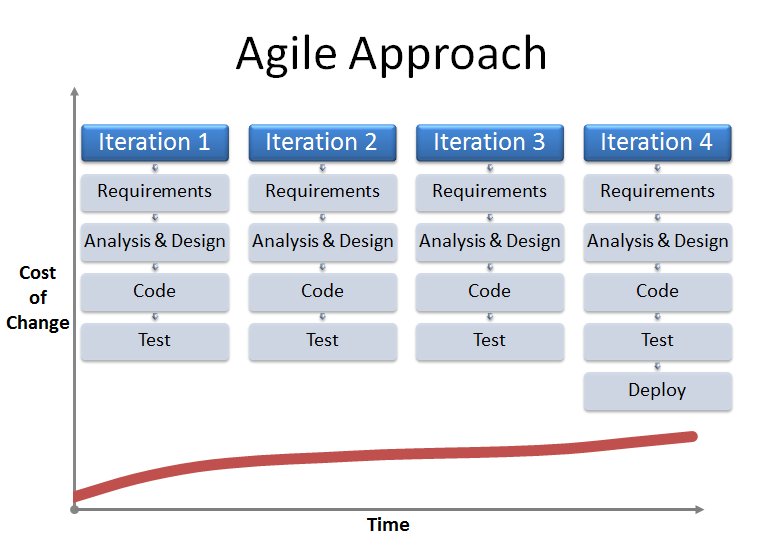
Chapter 2: Suppose your company is a App Development Company. How you apply Agile Testing Life Cycle in Development and How you prepare Test Cases and How you perform Agile Testing for Error Reporting and Analysis.

# Chapter 1

# AGILE SOFTWARE DEVELOPMENT AND TESTING

1. **AGILE SOFTWARE DEVELOPMENT**
   1. **What is Agile Software Development?**

Agile Software Development is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.



***Figure 1.1:***  ***Agile Model***

Agile Software Development is the general concept of all software development techniques and methods according to Agile philosophy.

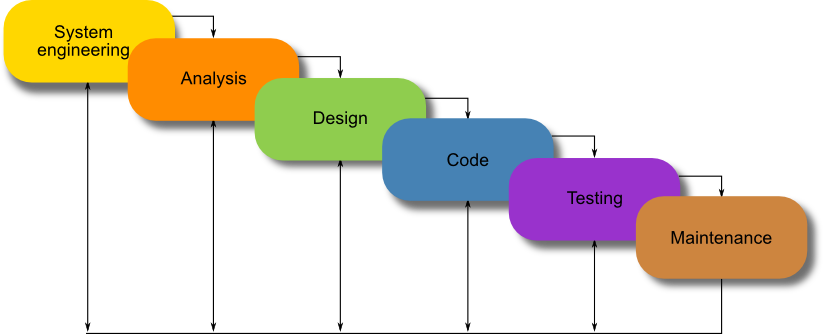
The Agile philosophy is pre-described in the Agile Manifesto through its core values ​​and universal principles, but does not specify how its values ​​and principles are to be implemented. Therefore, Agile method will help clearly define tasks for individuals and collectives to easily apply in work. The methods all encourage application planning, step-by-step development, early hand-over and continual improvement for the purpose of quickly adapting to change - a weakness of the system software development approach. Nation).

Below, we will learn about the birth history of the Agile Manifesto and some of the most common Agile methods.

* 1. **Agile Manifesto**
     1. **The history of the Agile Manifesto**

Agile was born in the context of software development experiment with the way the transmission system is developed in a waterfall model (Waterfall), or based on a plan (plan-driven).

These methods are characterized by a linear approach, performing steps according to a sequential plan. However, in practice many risks cannot be foreseen. One of the main reasons is because customers often change requirements during production. The reason is often that customers do not know what they need until using the product directly or maybe the original requirements are out of date and do not meet the business goals. When change is needed, all steps of design, development, testing, rewrite, and more. must be done again. As a result, products do not meet customer requirements, are delayed in time or exceed budget.



***Figure 1.2:***  ***Waterfall Model***

The software development methodology crisis of the 1990s of the twentieth century saw a very high failure rate of software projects. As a result, from 11-13 February 2001, 17 inventors and practitioners met together in Utah, USA to discuss new directions in software development methodology. They came to a consensus and released the Agile Manifesto (The Manifesto for Agile Software Development) and marked a new trend in software development.

The content of the Agile manifesto became the guiding philosophy for later Agile methods, as follows:

* + 1. **The content of the Agile Manifesto**

Nearly 20 years ago, 17 software developers came together in Snowbird, Utah to propose a new way of developing software “by doing it and helping others do it.” Through this work, the signers of the Manifesto understood how much of an impact these principles would help them in the field of software development—but they had no idea how quickly their ideas would spread beyond their industry. Values the Manifesto creators cited as paramount were:

***The Four Values of The Agile Manifesto***

The Agile Manifesto is comprised of four foundational values and 12 supporting principles which lead the Agile approach to software development. Each Agile methodology applies the four values in different ways, but all of them rely on them to guide the development and delivery of high-quality, working software.

1. Individuals and Interactions Over Processes and Tools

The first value in the Agile Manifesto is “Individuals and interactions over processes and tools.” Valuing people more highly than processes or tools is easy to understand because it is the people who respond to business needs and drive the development process. If the process or the tools drive development, the team is less responsive to change and less likely to meet customer needs. Communication is an example of the difference between valuing individuals versus process. In the case of individuals, communication is fluid and happens when a need arises. In the case of process, communication is scheduled and requires specific content.

1. Working Software Over Comprehensive Documentation

Historically, enormous amounts of time were spent on documenting the product for development and ultimate delivery. Technical specifications, technical requirements, technical prospectus, interface design documents, test plans, documentation plans, and approvals required for each. The list was extensive and was a cause for the long delays in development. Agile does not eliminate documentation, but it streamlines it in a form that gives the developer what is needed to do the work without getting bogged down in minutiae. Agile documents requirements as user stories, which are sufficient for a software developer to begin the task of building a new function.

The Agile Manifesto values documentation, but it values working software more.

1. Customer Collaboration Over Contract Negotiation

Negotiation is the period when the customer and the product manager work out the details of a delivery, with points along the way where the details may be renegotiated. Collaboration is a different creature entirely. With development models such as Waterfall, customers negotiate the requirements for the product, often in great detail, prior to any work starting. This meant the customer was involved in the process of development before development began and after it was completed, but not during the process. The Agile Manifesto describes a customer who is engaged and collaborates throughout the development process, making. This makes it far easier for development to meet their needs of the customer. Agile methods may include the customer at intervals for periodic demos, but a project could just as easily have an end-user as a daily part of the team and attending all meetings, ensuring the product meets the business needs of the customer.

1. Responding to Change Over Following a Plan

Traditional software development regarded change as an expense, so it was to be avoided. The intention was to develop detailed, elaborate plans, with a defined set of features and with everything, generally, having as high a priority as everything else, and with a large number of many dependencies on delivering in a certain order so that the team can work on the next piece of the puzzle.

With agile approaches, project teams spend more time on development and less time on documentation, resulting in a more efficient delivery of a working product.

***The Twelve Agile Manifesto Principles***

The Twelve Principles are the guiding principles for the methodologies that are included under the title “The Agile Movement.” They describe a culture in which change is welcome, and the customer is the focus of the work. They also demonstrate the movement’s intent as described by Alistair Cockburn, one of the signatories to the Agile Manifesto, which is to bring development into alignment with business needs.

The twelve principles of agile development include:

1. Customer satisfaction through early and continuous software delivery:

Customers are happier when they receive working software at regular intervals, rather than waiting extended periods of time between releases.

1. Accommodate changing requirements throughout the development process:

The ability to avoid delays when a requirement or feature request changes.

1. Frequent delivery of working software:

Scrum accommodates this principle since the team operates in software sprints or iterations that ensure regular delivery of working software.

1. Collaboration between the business stakeholders and developers throughout the project

Better decisions are made when the business and technical team are aligned.

1. Support, trust, and motivate the people involved

Motivated teams are more likely to deliver their best work than unhappy teams.

1. Enable face-to-face interactions

Communication is more successful when development teams are co-located.

1. Working software is the primary measure of progress

Delivering functional software to the customer is the ultimate factor that measures progress.

1. Agile processes to support a consistent development pace

Teams establish a repeatable and maintainable speed at which they can deliver working software, and they repeat it with each release.

1. Attention to technical detail and design enhances agility

The right skills and good design ensures the team can maintain the pace, constantly improve the product, and sustain change.

1. Simplicity

Develop just enough to get the job done for right now.

1. Self-organizing teams encourage great architectures, requirements, and designs

Skilled and motivated team members who have decision-making power, take ownership, communicate regularly with other team members, and share ideas that deliver quality products.

1. Regular reflections on how to become more effective

Self-improvement, process improvement, advancing skills, and techniques help team members work more efficiently.

The intention of Agile is to align development with business needs, and the success of Agile is apparent. Agile projects are customer focused and encourage customer guidance and participation. As a result, Agile has grown to be an overarching view of software development throughout the software industry and an industry all by itself.

* + 1. **Advantages and Disadvantages of Agile**

Today, Agile is such a buzzword that even teams outside software development try to incorporate it into their workflow. But Agile is not for everyone.

For example, a marketing agency can never implement Agile, because clients don't want to pay for a half-finished marketing campaign and iterate. There are revisions, but their number is clearly specified in the contract. Plus, there's no such thing as "working increments" - you either have the deliverables, or you don't.

Agile isn't the right approach for every software project, either. If you don't have access to customers, can't iterate, or if you have a complex organizational structure, it's very difficult to adhere to Agile principles.

Agile works best when:

* You can't estimate the time you'll need and don't know the full scope of requirements
* You don't know whether there's a need on the market for your software
* You can't map out the business needs, so the design needs to emerge through trial and error
* You have unlimited access to your customer who's ready for extensive involvement
* You can afford to iterate and don't need to deliver fully functional software at once

Neither you nor your client has a complex bureaucracy that delays decision-making

Clients don't have a fixed budget/schedule

You need to capture the market before there's any competition

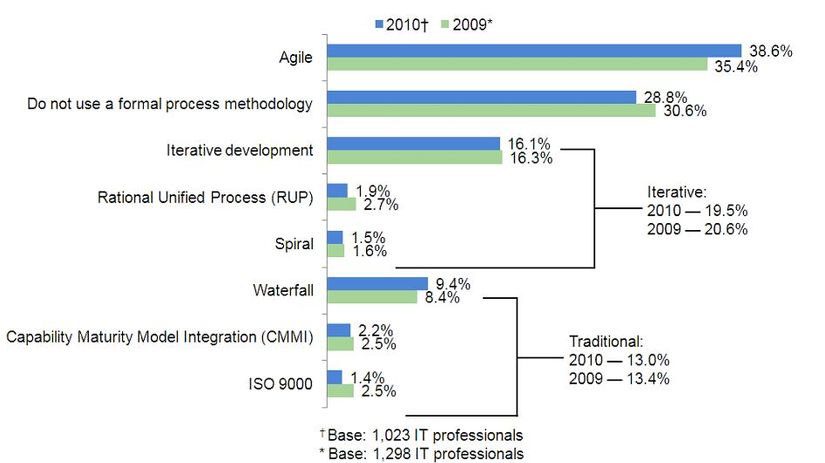
Your customers don't have trouble updating their software (or don't even notice it, e.g., they use a web app)

As you can see, Agile is more suited for small-to-medium size organizations than it is for corporations. The reason is simple: the fewer people there are, the easier it is to make a decision and respond to change. Also, Agile is more suited for product companies over consultancies.

Agile is also great for startups, where "fail fast" is the dominant mantra. Venture capitalists encourage startups to try crazy ideas and let the market do the work. Most of the ideas will fail, and those few that succeed will change the world.

***Advantages***

* You can deploy software quicker, so your customer can get value sooner rather than later
* You waste fewer resources because you always work on up-to-date tasks
* You can better adapt to change and respond faster
* Faster turnaround times
* You can detect and fix issues and defects faster
* You spend less time on bureaucracy and busywork
* There's a big community of Agile practitioners with whom you can share knowledge
* You can get immediate feedback (which also improves team morale)
* Developers can improve their skills based on QA feedback
* You don't have to worry about premature optimization
* You can experiment and test ideas because its costs are low



***Figure 1.3:***  ***The success rate of Agile Model***

***Disadvantages***

Agile has substantial advantages, and it's important to know the disadvantages, limitations, and risks it brings.

* Documentation tends to get sidetracked, which makes it harder for new members to get up to speed
* It's more difficult to measure progress than it is in Waterfall because progress happens across several cycles
* Agile demands more time and energy from everyone because developers and customers must constantly interact with each other
* When developers run out of work, they can't work on a different project since they'll be needed soon
* Projects can become ever-lasting because there's no clear end
* Scope creep and experience rot
* Clients who work on a specified budget or schedule can't know how much the project will actually cost, which makes for a very complicated sales cycle ("Until iteration ends," is not something clients like to hear)
* The product lacks overall design, both from a UX and architecture point of view, which leads to problems the more you work on the product
* Teams can get sidetracked into delivering new functionalities at the expense of technical debt, which increases the amount of unplanned work
* Features that are too big to fit into one or even several cycles are avoided because they don't fit in nicely into the philosophy
* You need a long-term vision for the product and actively work on communicating it
* Products lack cohesion, and the user journey is fragmented because the design is fragmented. The more time passes, the more disjointed the software ends up becoming
* Short cycles don't leave enough time for the design thinking process, so designers have to redevelop the experience over and over due to negative feedback
* Check here for some more Scrum sprint planning anti-patterns and product backlog and refinement anti-patterns work. Most of the ideas will fail, and those few that succeed will change the world.
  + 1. **What is Scrum?**

Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one.

* A “process framework” is a particular set of practices that must be followed in order for a process to be consistent with the framework. (For example, the Scrum process framework requires the use of development cycles called Sprints, the XP framework requires pair programming, and so forth.)
* “Lightweight” means that the overhead of the process is kept as small as possible, to maximize the amount of productive time available for getting useful work done.

A Scrum process is distinguished from other agile processes by specific concepts and practices, divided into the three categories of Roles, Artifacts, and Time Boxes. These and other terms used in Scrum are defined below. Scrum is most often used to manage complex software and product development, using iterative and incremental practices. Scrum significantly increases productivity and reduces time to benefits relative to classic “waterfall” processes. Scrum processes enable organizations to adjust smoothly to rapidly-changing requirements, and produce a product that meets evolving business goals. An agile Scrum process benefits the organization by helping it to

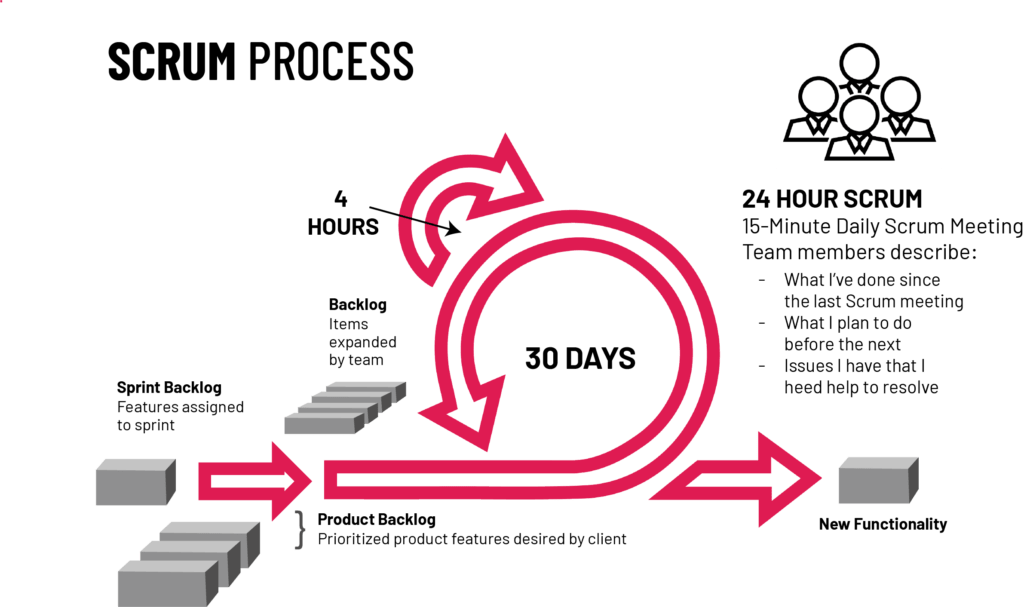
* Increase the quality of the deliverables
* Cope better with change (and expect the changes)
* Provide better estimates while spending less time creating them
* Be more in control of the project schedule and state

The Scrum Framework itself is very simple. It defines only some general guidelines with only a few rules, roles, artifacts and events. Nevertheless each of these components is important, serves a specific purpose and is essential for a successful usage of the framework.

The main components of Scrum Framework are:

* The scrum roles: Scrum Master, Scrum Product Owner and the Scrum Team
* The artifacts: Sprint backlog, product backlog, burndown chart, log, etc…
* Scrum events: Sprint planning, spring review, daily standup, sprint retro, etc…
* Sprint

The diagram below shows the key elements of the SCRUM framework. The process has been applied by the agile software tool – Scrum Process Canvas.



***Figure 1.4:***  ***The components of Scrum process***

1. **AGILE TESTING**
   1. **What is Agile Testing?**

Agile development is focused on delivering working, iterative software as quickly and as frequently as possible. Agile testing is the process of testing software for issues or bugs during an Agile workflow. Unlike the Waterfall approach, where QA team members would not get involved until the end of software production, Agile testing does not wait to repair or mitigate errors; work is completed as soon as the issues are discovered, leading to faster development and product delivery.

But how is Agile testing folded into Agile software development? Instead of waiting to contribute to the project once it’s in the final stages, an Agile tester must reframe their thinking and their actions. They are no longer “just a tester” but a contributing member of the team, working to improve upon the product throughout the process.

* 1. **Agile Testing Principles**

Agile’s continuous testing model requires users to test smarter rather than harder, operating more flexibly than other more rigid QA processes, such as the Waterfall method. Here are some additional principles of Agile testing to keep in mind:

* **Continuous testing and feedback:** Since testing and feedback are the only ways to ensure the progress of a product, Agile teams test on an ongoing basis, regularly providing feedback regarding the product’s quality.
* **Testing conducted by the entire team:** Rather than leave testing up to a designated test team, Agile teams ask developers, business analysts, and other stakeholders to test an application before its release.
* **Test-driven processes:** While other testing methods perform testing after implementation, Agile testing requires testing before and during implementation to ensure quality upon release.
* **Decreased feedback response time and documentation:** Agile testing is completed using the same checklist over and over, making documentation less of a priority. Since the entire team is involved in continuous testing, feedback can be completed and delivered at a much faster rate.
  1. **Agile Testing Methods**

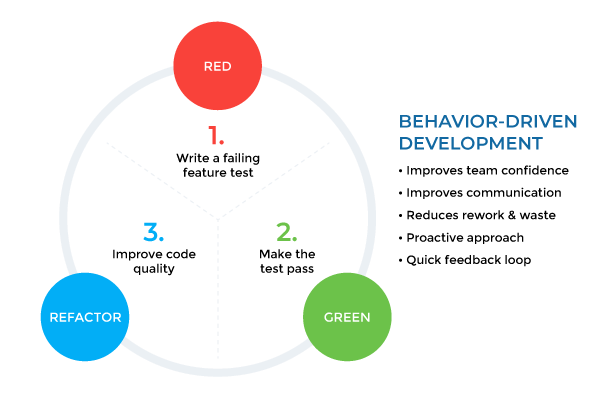
Agile testing can be divided into three different methods. Each method offers various benefits, depending on how your team is structured and how you want to structure your Agile QA process. Take a look below to see which method might be best suited to your project:

***Behavior Driven Development (BDD)***

BDD encourages communication between project stakeholders so all members understand each feature, prior to the development process. In BDD, testers, developers, and business analysts create “scenarios”, which facilitate example-focused communication.

Scenarios are written in a specific format, the Gherkin Given/When/Then syntax. They contain information on how a feature behaves in different situations with varying input parameters. These are known as “executable specifications” as they are made up of both specifications and inputs to the automated tests.

The idea of BDD is that the team creates scenarios, builds tests around those scenarios which initially fail, and then builds the software functionality that makes the scenarios pass. It is different from traditional Test Driven Development (TDD) in that complete software functionality is tested, not just individual components.



***Figure 1.5:***  ***Behavior Driven Development (BDD) Method***

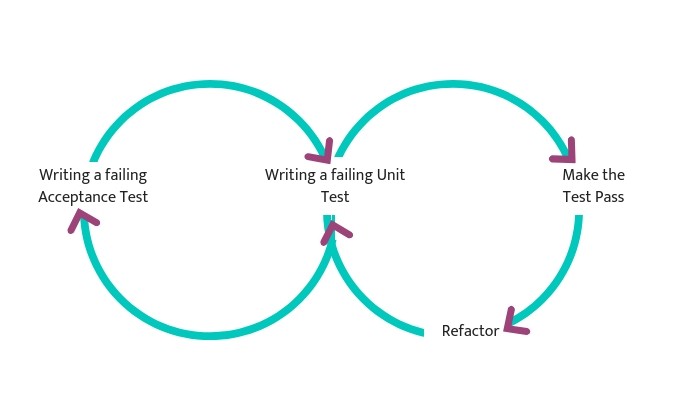
Best practices for testers using a BDD methodology:

* Streamline documentation to ensure the process is efficient
* Embrace a “three amigos” approach, where the developer, product owner, and tester work together to define scenarios and tests
* Use a declarative test framework, such as Cucumber, to specify criteria
* Build automated tests and reuse them across scenarios
* Have business analysts write test cases and learn the Gherkin syntax

***Acceptance Test-Driven Development (ATDD)***

ATDD involves the customer, developer, and tester. “Three Amigos” meetings are held to gather input from these three roles, and use them to define acceptance tests. The customer focuses on the problem, the developer pays attention to how the problem will be solved, and the tester looks at what could go wrong.

The acceptance tests represent a user’s perspective, and specify how the system will function. They also ensure that the system functions as intended. Acceptance tests can often be automated. Like in the BDD approach, acceptance tests are written first, they initially fail, and then software functionality is built around the tests until they pass.



***Figure 1.6:***  ***Acceptance Test Driven Development (ATDD) Method***

Best practices for testers using an ATDD methodology include:

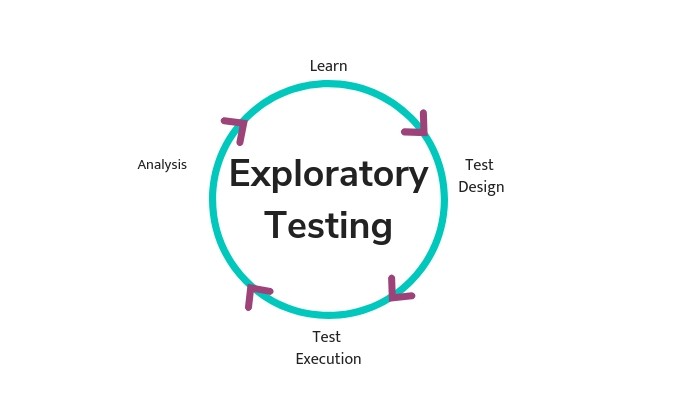
* Interact directly with customers to align expectations, for example, through focus groups
* Involve customer-facing team members to understand customer needs, including customer service agents, sales representatives, and account managers
* Develop acceptance criteria according to customer expectations
* Prioritize two questions: How should we validate that the system performs a certain function? Will customers want to use the system when it has this function?

***Exploratory Testing***

Exploratory testing is arguably the most flexible method of testing, allowing users to adapt to changes quicker and more effectively. This type of testing values a working product over documentation, collaboration more than contract negotiations, and interactions and individual users over technical processes and tools. As testing begins, it’s the tester’s goal to identify a product’s functionality through exploration, learning the application as they go and designing and carrying out test plans according to findings.

Best practices for exploratory testing:

* Organize functionality in the application, using a spreadsheet, mind map etc.
* Even though there is no detailed documentation of how tests were conducted, track which software areas were or were not covered with exploratory testing
* Focus on areas and scenarios in the software which are at high risk or have high value for users
* Ensure testers document their results so they can be accountable for areas of software they tested



***Figure 1.7:***  ***Exploratory Testing Method***

***Session-Based Testing***

This method is similar to exploratory testing, but is more orderly, aiming to ensure the software is tested comprehensively. It adds test charters, which helps testers know what to test, and test reports which allow testers to document what they discover during a test. Tests are conducted during time-boxed sessions.

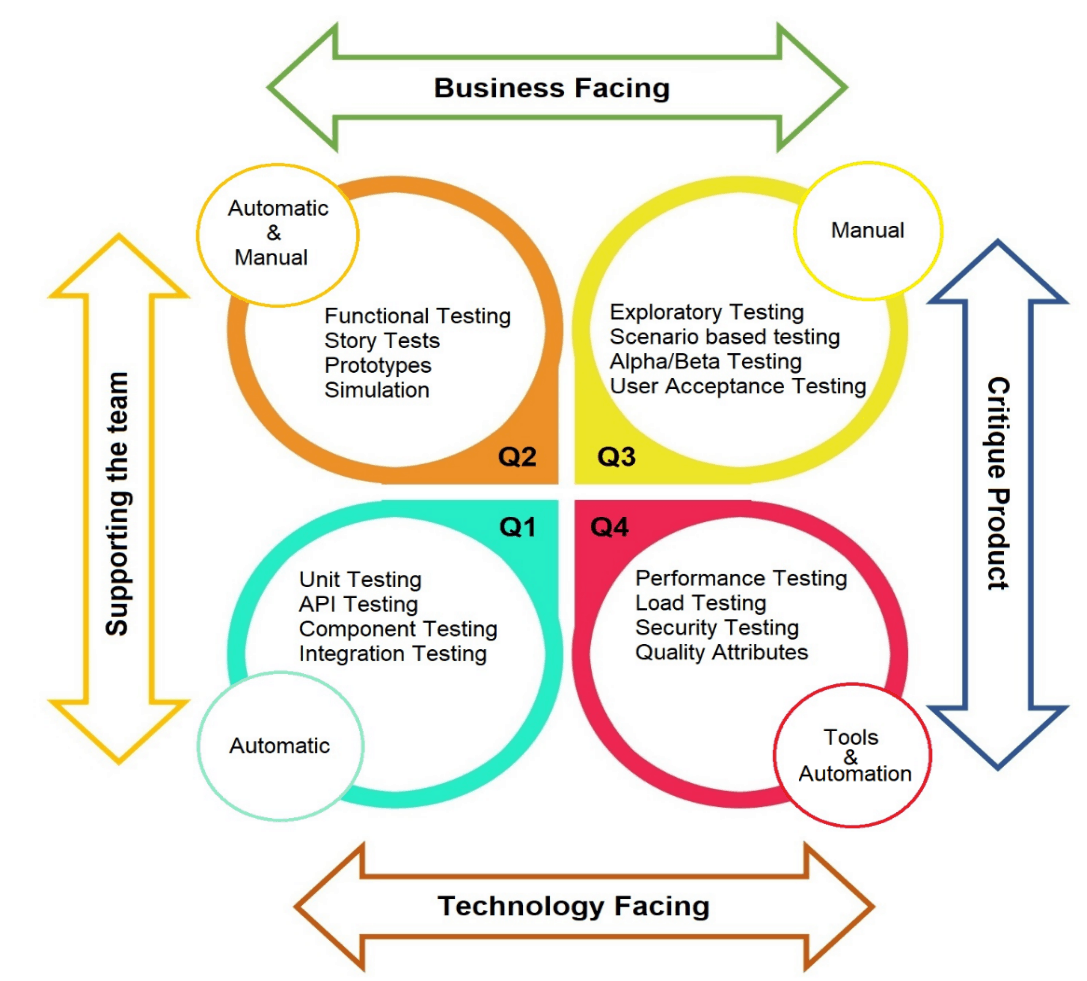
Each session ends with a face-to-face brief between tester(s) and either the developers responsible, scrum master or manager, covering the five PROOF points:

* What was done in the test (Past)
* What the tester discovered or achieved (Results)
* Any problems that got in the way (Obstacles)
* Remaining areas to be tested (Outlook)
* How the tester feels about the areas of the product they tested (Feelings).

Best practices for session-based testing include:

* Define a goal so testers are clear about priorities of testing in the current sprint
* Develop a charter that states areas of the software to test, when the session will occur and for how long, which testers will conduct the session, etc.
* Run uninterrupted testing sessions with a fixed, predefined length
* Document activities, notes, and also takeaways from the face-to-face brief in a session report
  1. **Agile Testing Quadrants**

The following are the four quadrants of Agile testing, as introduced by Brian Marick. It enlists the different types of software testing that fit the various stages of the development process.



***Figure 1.8:***  ***Agile Testing Quadrant***

***Agile Testing Quadrant 1: Unit Level, Technology Facing***

**Unit testing:** is an Agile Testing process for checking the quality and efficiency of individual user stories, i.e., for a specific feature built by the developers. In short, it checks the user story from the design and technical perspective. The developers themselves can also run unit tests.

For example, if the code for the authentication and login user story is ready, the unit testing will be run to check if the login works as per the expectations.

**Component testing:** in agile is a testing process where the individual objects or parts of the user story are tested individually. It can be done in isolation or in conjunction with other components that make up a user story.

For example, if we again refer to the authentication and login user story, checking the different components on the screen, i.e., options like login with email, Facebook, or even your phone number will be the various components that need to be tested.

***Agile Testing Quadrant 2: System Level, Business Facing***

**Functional testing:** is a type of black-box testing where the software system is tested to check whether it adheres to the specified functional requirements. The loopholes in performance are identified and reported to the developers. This test focuses on examining the basic functionality, usability, and accessibility of the application.

**Story tests**: are run to check if all the user stories are accommodated as a part of the software. When the stories are created initially, the corresponding story test cases are also developed alongside that, in turn, act as story acceptance tests. Each of the user stories adds a new test to the story test stack.

**Prototype & Simulation Testing:** these tests are mainly conducted to check the design and the UX flow of the software. Any defects on the visible side of the application are identified and reported so that the prototypes can be reworked before the MVP is launched. These types of tests, if successful, also help attract seed funding for the product.

**Pair Testing:** is a practice where two people work simultaneously and in the same space to test the software. This type of testing can be conducted by a tester-tester pair or even a developer-tester pair. Pair testing is foolproof and offers speedy results.

***Agile Testing Quadrant 3: System or User Acceptance Level, Business Facing***

**Exploratory testing**: is a type of testing where the test cases are not created in advance. Testing is conducted during the ongoing development process. However, the testers generally brainstorm about what tests they will perform before the process. These tests are mainly based on the experience, learning, and creativity of testers.

**Usability testing:** is a type of testing where a set of end-users step in to test how easy it is to use and navigate the user interface. These users are assigned tasks that they need to perform to see how the application works. However, testers might step into the user’s shoes too to evaluate the product from their perspective.

**User acceptance testing**: is conducted by the end-users or clients to check whether their requirements have been met, and pain points have been addressed. This type of test checks the wholeness of the software and its ability to perform as intended.

**Alpha & Beta Testing:** the tester conducts alpha testing before the software product gets launched in the market. All the bugs and inconsistencies are identified and fixed before the actual launch.

In beta testing, the product is released to minimal end users who test and provide feedback. Such tests help mitigate product failure risks. In Agile Testing, alpha and beta testing should be conducted with every release.

***Agile testing Quadrant 4: System or Operational Acceptance Level, Technology Facing***

**Performance tests:** are conducted to test the speed, responsiveness, intuitiveness, and stability of the software product. In Agile Testing, performance testing can be checked at the end of each sprint that delivers a user story. This ensures a good speed of testing and timely delivery of high-performing products or fragments of the product.

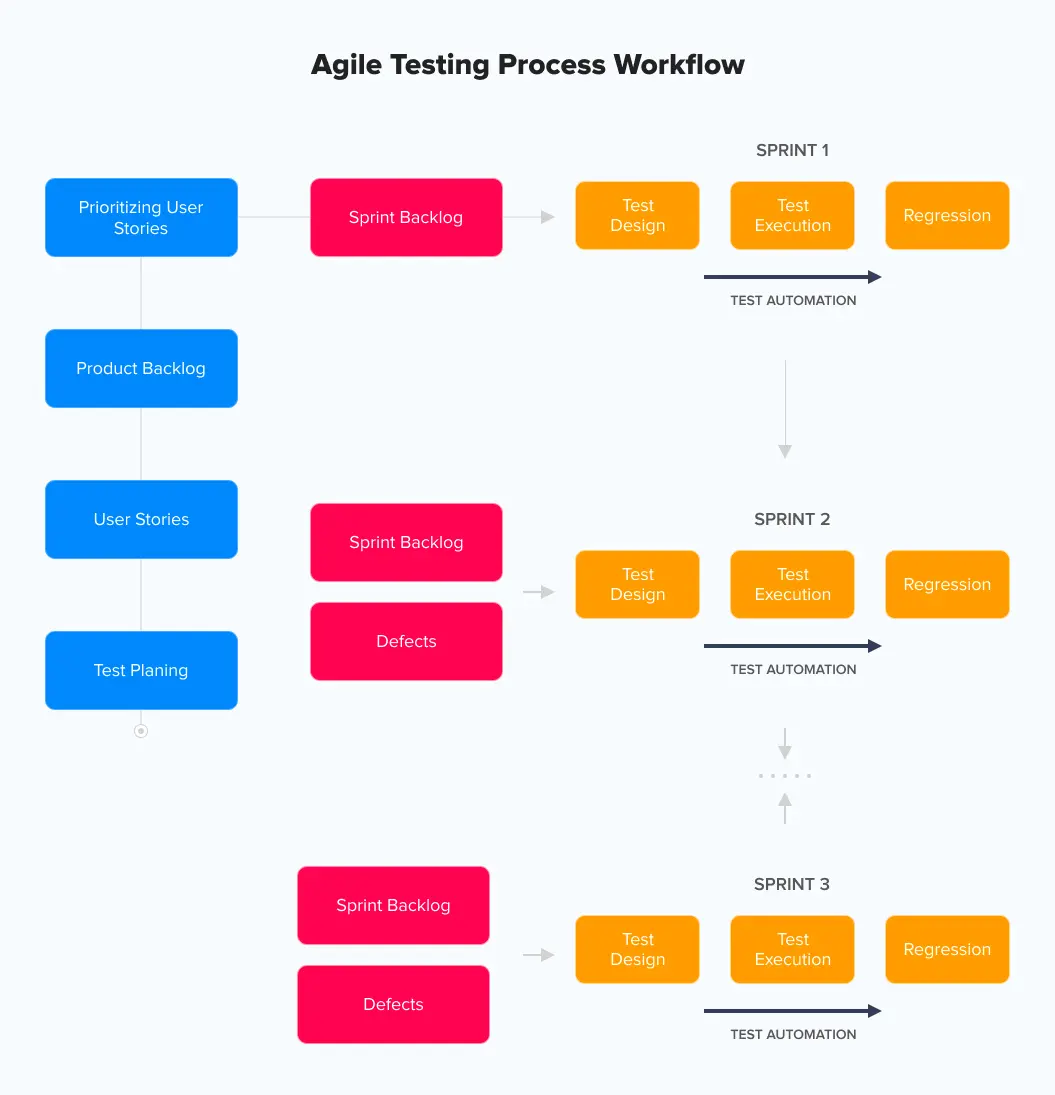
**Load testing:** is conducted to check the working of the software under specific load conditions. This type of testing is generally done to check the application’s performance under normal and peak conditions. Automated Agile Testing tools are used for load testing, and should be performed at the end of every sprint to check the performance of every user story.

**Security testing:** is a type of test that checks the robustness of the software system and its level of protection against intruders. Security as a part of Agile Testing should be conducted using automated tests weekly or monthly.

* 1. **Agile testing Life-cycle**

The Agile testing lifecycle is a set of processes and best practices that the team adopts to deliver iterative quality feedback to the development team. It helps detect and plug any functional and non-functional mismatches, at the early stages, between the project requirements and the project delivery.

The stages of the Agile testing lifecycle are based on the Agile testing pyramid. Here is how the Agile testing teams apply best practices and methodologies to deliver a quality product at speed and scale.



***Figure 1.9:***  ***Agile Testing Life Cycle Process***

***Sprint Planning***

A sprint is a pre-decided time period within which the team members working on the project needs to complete a user story.

Before a sprint kick starts, the product owner, developers, and testers discuss what needs to be accomplished in each of the sprints and the consecutive sprint goal. The daily standups are also a must among the agile teams throughout the Product Development process.

***Building a Test Case Design***

The testing teamwork maintains a cadence with the development team, i.e., while the development team is building a user story, the testing team develops test-case designs. A test case design enlists how you will go about setting up the different test cases. These designs ensure that the quality tests stick to the laid down process.

The documented test cases are then handed over to the development team for review and also so that both the teams involved can decide the test cases that can be automated.

Testing

As soon as the user story gets completed on the development side, the testing team steps into quality check the software. The developers and testers work in tandem to execute testing in an agile environment.

This is done to identify early-stage defects so that the development team can fix the minor ones there and then, and the rest can be fixed in the following sprints on a priority basis. As for the automated test cases, they are run across the development process daily.

***Ensure Product Stability***

Agile is associated with iterative development, i.e., new requirements can be accommodated at any stage of the development process without adding to the complexities.

The testing team’s responsibility is to determine when to stop the requirement flow and ensure product stability.

***Manual & Automated Regression Testing***

The manual and automated test cases are run to check the quality of the code after new user stories are added to the agile development process.

These tests ensure that the new requirement additions do not disturb the software’s overall structure and functioning.

# Chapter 2

# AGILE TESTING IN CASE STUDY

Suppose your company is a App Development Company. How you apply Agile Testing Life Cycle in Development and How you prepare Test Cases and How you perform Agile Testing for Error Reporting and Analysis.

1. **How you apply Agile Testing Life Cycle in Development**

Suppose the company is applying Scrum framework for software development , Scrum advocates the Whole Team Approach, in the sense that every team member has to take part in every project activity. Testers also participate in all the project and development activities contributing their expertise in testing.

The whole team works together on Test Strategy, Test Planning, Test Specification, Test Execution, Test Evaluation, and Test Results Reporting.

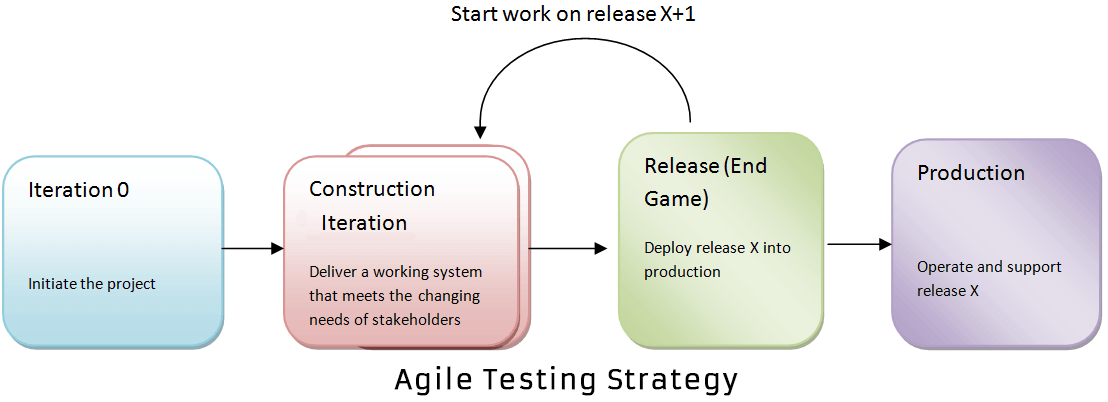
In a real project, testers perform the following activities during the various stages of Scrum:

* ***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:



***Figure 2.1:***  ***Agile Testing Strategy***

(a) 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

a) Establishing a business case for the project

b) Establish the boundary conditions and the project scope

c) Outline the key requirements and use cases that will drive the design trade-offs

d) Outline one or more candidate architectures

e) Identifying the risk

f) Cost estimation and prepare a preliminary project

(b) 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.

(c) 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.

(d) Production

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

* ***Sprint Planning***

Testers are involved in Sprint planning meetings and especially add value in the following activities:

* Planning the testing for the release
* Participating in the detailed risk analysis of user stories
* Creating acceptance tests for the user stories
* Defining the necessary test levels, testers should decide how many hours (Effort Estimation) it should take to finish testing for each of selected user stories.
* Breaking down user stories into tasks (particularly testing tasks)
* Estimating testing effort associated with the user stories and all testing tasks
* Identifying functional and non-functional aspects of the system to be tested
* Supporting and participating in test automation at multiple levels of testing
* ***Sprint***

Testers are involved during Sprint and especially add value in the following activities:

* Testers support developers in unit testing
* Test user-story when completed. Test execution is performed in a lab where both tester and developer work hand in hand. Defect are logged in Defect Management tool which are tracked on a daily basis. Defects can be conferred and analyzed during the scrum meeting. Defects are retested as soon as it is resolved and deployed for testing
* Testers attends all daily standup meeting to speak up
* Testers can bring any backlog item that cannot be completed in the current sprint and put to the next sprint
* Testers is responsible for developing automation scripts, schedules automation testing with Continuous Integration (CI) system. Automation receives the importance due to short delivery timelines. Test Automation can be accomplished by utilizing various open source or paid tools available in the market. This proves effective in ensuring that everything that needs to be tested was covered. Sufficient Test coverage can be achieved with a close communication with the team.
* Review CI automation results and send Reports to the stakeholders
* Executing non-functional testing for approved user stories
* Coordinate with customer and product owner to define acceptance criteria for Acceptance Tests
* At the end of the sprint, the tester also does acceptance testing(UAT) in some case and confirms testing completeness for the current sprint

1. **How you prepare Test Cases.**

The preparation of writing TestCases will be based on the User Story and Acceptance Criteria of the project. Methods and techniques will be applied to design a complete and efficient TestCase.

Following are the typical design techniques in TestCases design:

Deriving test cases directly from a requirement specification or black box test design technique. The Techniques include:

* Boundary Value Analysis (BVA)
* Equivalence Partitioning (EP)
* Decision Table Testing
* State Transition Diagrams
* Use Case Testing

Deriving test cases directly from the structure of a component or system:

* Statement Coverage
* Branch Coverage
* Path Coverage
* LCSAJ Testing

Deriving test cases based on tester's experience on similar systems or testers intuition:

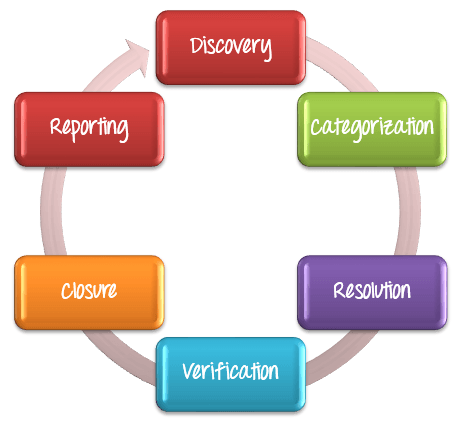
* Error Guessing
* Exploratory Testing

1. **How you perform Agile Testing for Error Reporting and Analysis**

* ***Defect Management Process***

Defect Management is a systematic process to identify and fix bugs. A defect management cycle contains the following stages

1. Discovery of Defect,
2. Defect Categorization
3. Fixing of Defect by developers
4. Verification by Testers,
5. Defect Closure
6. Defect Reports at the end of project



***Figure 2.2:***  ***Defect Management Process***

* ***Defect Life Cycle Explained***

1. Tester finds the defect
2. Status assigned to defect- New
3. A defect is forwarded to Project Manager for analyze
4. Project Manager decides whether a defect is valid
5. Here the defect is not valid- a status is given "Rejected."
6. So, project manager assigns a status rejected. If the defect is not rejected then the next step is to check whether it is in scope. Suppose we have another function- email functionality for the same application, and you find a problem with that. But it is not a part of the current release when such defects are assigned as a postponed or deferred status.
7. Next, the manager verifies whether a similar defect was raised earlier. If yes defect is assigned a status duplicate.
8. If no the defect is assigned to the developer who starts fixing the code. During this stage, the defect is assigned a status in- progress.
9. Once the code is fixed. A defect is assigned a status fixed
10. Next, the tester will re-test the code. In case, the Test Case passes the defect is closed. If the test cases fail again, the defect is re-opened and assigned to the developer.
11. Consider a situation where during the 1st release of Flight Reservation a defect was found in Fax order that was fixed and assigned a status closed. During the second upgrade release the same defect again re-surfaced. In such cases, a closed defect will be re-opened.

To perform bugs checking, analysis, and reporting, testers use the following methods to perform:

***Test Driven Development (TDD) Method***

In the Test Driven Development (TDD) method, the code is developed based on the Testfirst approach directed by Automated Test Cases. A test case is written first to fail, code is developed based on that to ensure that the test passes. Method is repeated, refactoring is done through the development of code.

TDD can be understood with the help of the following steps:

Step 1: Write a Test case to reflect the expected behavior of the functionality of the code that needs to be written.

Step 2: Run the test. The test fails as the code is still not developed.

Step 3: Develop code based on the test case.

Step 4: Run the test again. This time, the test has to pass as the functionality is coded. Repeat Step (3) and Step (4) till the test passes.

Step 5: Refactor the code.

Step 6: Run the test again to ensure it passes.

Repeat Step 1 => Step 6 adding test cases to add functionality. The added tests and the earlier tests are run every time to ensure the code is running as expected. To make this process fast, tests are automated.

***Acceptance Test Driven Development Method***

In the Acceptance Test Driven Development (ATDD) method, the code is developed based on the test-first approach directed by Acceptance Test Cases. The focus is on the acceptance criteria and the Acceptance Test Cases written by the testers during User Story Creation in collaboration with the customer, end users and relevant stakeholders.

Step 1: Write Acceptance Test Cases along with user stories in collaboration with the customer and users.

Step 2: Define the associated acceptance criteria.

Step 3: Develop code based on the acceptance tests and acceptance criteria.

Step 4: Run the acceptance tests to ensure that the code is running as expected.

Step 5: Automate the acceptance tests. Repeat Step 3 – Step 5 until all the user stories in the iteration are implemented.

Step 6: Automate the regression tests.

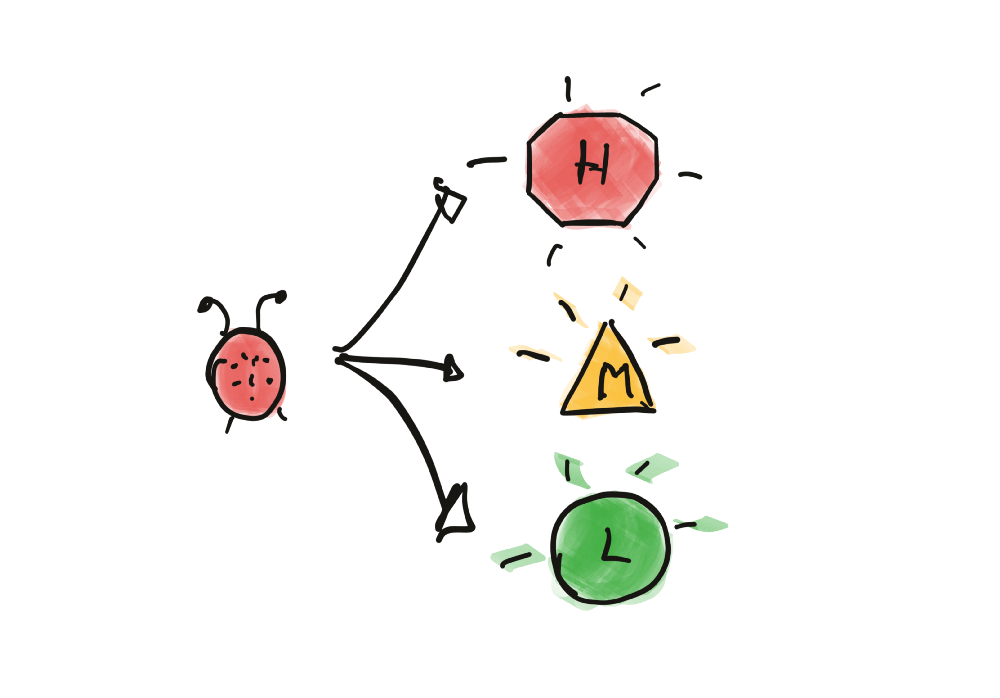
Step 7: Run the automated Regression Tests to ensure Continuous Regression.

* ***Agile Bugs classification***

We can classify the bugs by the moment when they were identified: in production, in development (user stories in the current sprint).

The bugs reported in production need to have assigned a priority:

* High priority bug: in this category, we consider the bugs that the core functionalities of the product. For example, for a car software product, an issue that doesn’t allow the user to drive its car will be a high priority bug.
* Medium priority bug: there is a workaround available. For example, I am opening a car with the key instead of pressing the open button of the key. Unlocking the vehicle is a core function of the car, but since there is a workaround available, the bug will be a medium priority.
* Low priority bug: the issue is rather cosmetic or comfort, with no critical functionality.



***Figure 2.3:***  ***Agile Bugs Prioritisation***

Defining the classification of the bugs helps for the fast sorting of the defects. The product owner can decide quickly which ones get on the teams’ sprint and which bugs to add to the product backlog.

We could have operational teams and product teams. This article concerns the product teams that do both maintenance of the production product but also the new development of the product.

* ***Bug Report***

Bug Report in Software Testing is a detailed document about bugs found in the software application. Bug report contains each detail about bugs like description, date when bug was found, name of tester who found it, name of developer who fixed it, etc. Bug report helps to identify similar bugs in future so it can be avoided.

While reporting the bug to developer, your Bug Report should contain the following information

* **Defect\_ID**: Unique identification number for the defect.
* **Defect Description:** Detailed description of the Defect including information about the module in which Defect was found.
* **Version:** Version of the application in which defect was found.
* **Steps:** Detailed steps along with screenshots with which the developer can reproduce the defects.
* **Date Raised:** Date when the defect is raised
* **Reference:** where in you Provide reference to the documents like . requirements, design, architecture or maybe even screenshots of the error to help understand the defect
* **Detected By:** Name/ID of the tester who raised the defect
* **Status:** Status of the defect , more on this later
* **Fixed by:** Name/ID of the developer who fixed it
* **Date Closed:** Date when the defect is closed
* **Severity** which describes the impact of the defect on the application
* **Priority** which is related to defect fixing urgency. Severity Priority could be High/Medium/Low based on the impact urgency at which the defect should be fixed respectively

# CONCLUSIONS

Software testing is very important in all areas of life. To have a quality products there must be an effective development and quality control process. Each company will select and apply for itself a management image, but the ultimate goal is still to make products that meet the needs of customers. The essay's purpose is to clarify more about the product testing process. How to apply in practice and include what steps. Due to the vast theory of testing and the practical application that requires more in-depth studies, the content is only basic.

Agile is a flexible product development process, applied in a wide variety of manufacturing industries. If applicable, the content of the essay will be studied in more detail.

**REFERENCES**

1. Dr. Anand Nayyar, Instant Approach to Software Testing, 2019
2. Website: https://www.guru99.com
3. Website: https://www.tutorialspoint.com