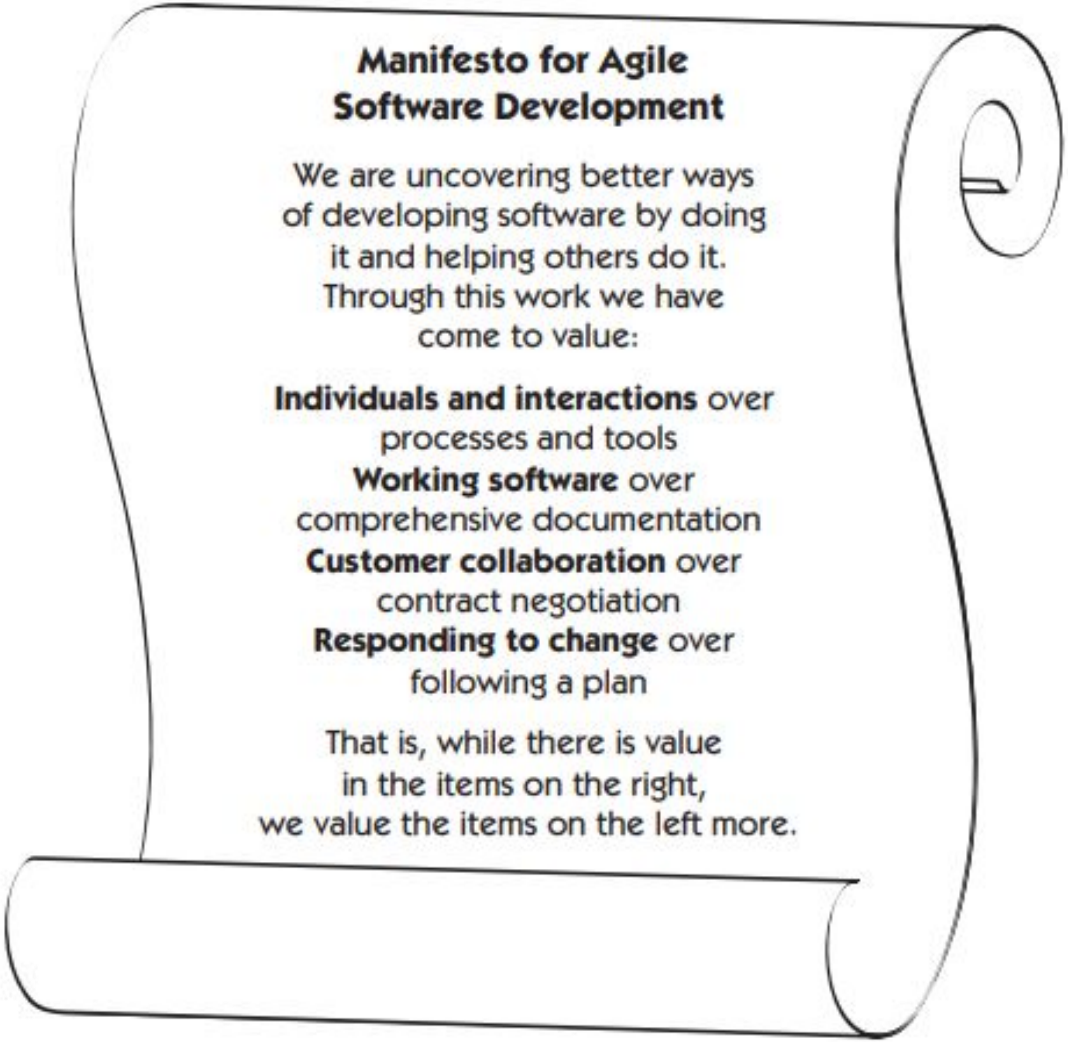
Agile Testing

Agile Manifesto

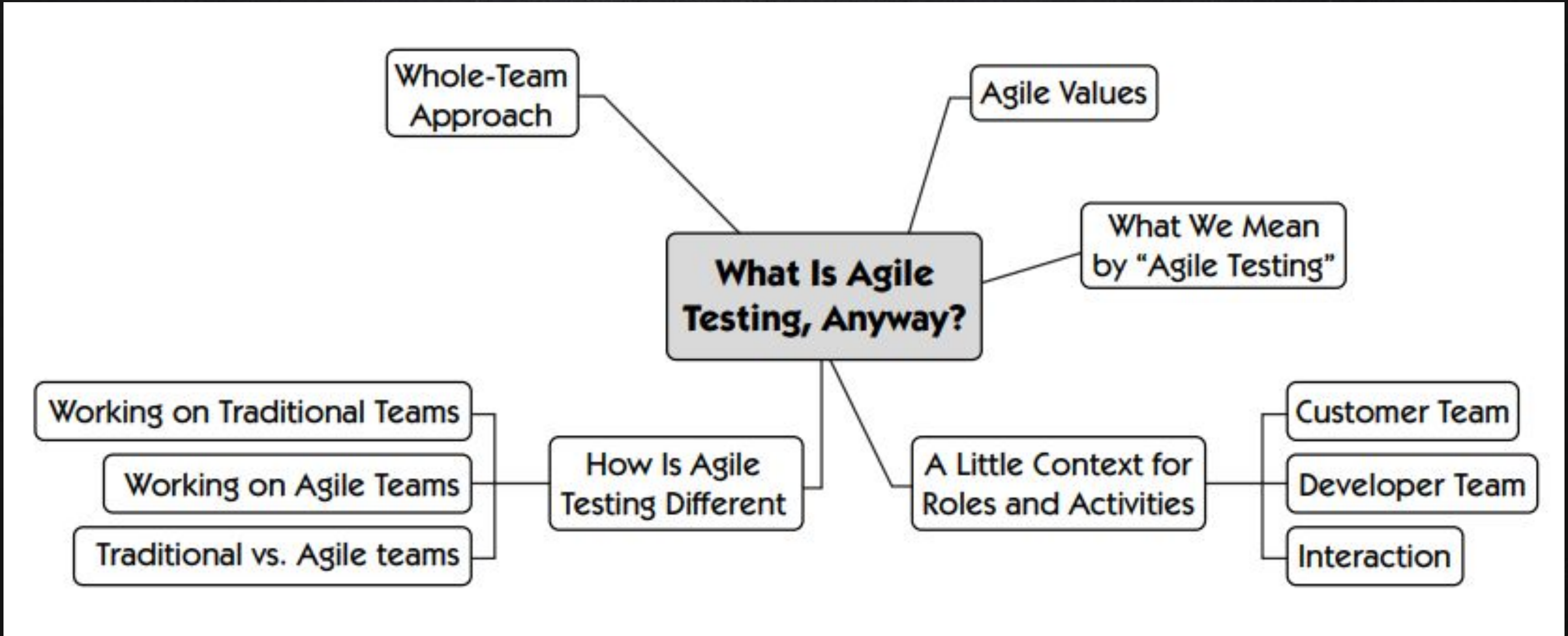


Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.



What Is Agile Testing, Anyway?

Whole-Team Approach

Agile Values

What We Mean by "Agile Testing"

How Is Agile Testing Different

Working on Traditional Teams

Working on Agile Teams

Traditional vs. Agile teams

A Little Context for Roles and Activities

Customer Team

Developer Team

Interaction

✘ “Agile” is a buzzword

✘ It’s loaded with different meanings that apply in different circumstances.

✘ One way to define “agile development” is to look at the Agile Manifesto.

✘ Using the values from the Manifesto to guide us, we strive to deliver small chunks of business value in extremely short release cycles.

WHAT DO WE MEAN BY “AGILE TESTING”?

✘ You might have noticed that we use the term “tester” to describe a person whose main activities revolve around testing and quality assurance.

✘ You’ll also see that we often use the word “programmer” to describe a person whose main activities revolve around writing production code.

✘ We don’t intend that these terms sound narrow or insignificant.

✘ Programmers do more than turn a specification into a program.

✘ We don’t call them “developers,” because everyone involved in delivering software is a developer.

✘ Testers do more than perform “testing tasks.”

✘ Each agile team member is focused on delivering a high-quality product that provides business value.

✘ Agile testers work to ensure that their team delivers the quality their customers need. We use the terms “programmer” and “tester” for convenience.

- ✘ Several core practices used by agile teams relate to testing.
- ✘ Agile programmers use test-driven development (TDD), also called test-driven design, to write quality production code.
- ✘ With TDD, the programmer
  - ○ writes a test for a tiny bit of functionality, sees it fail,
  - ○ writes the code that makes it pass, and
  - ○ then moves on to the next tiny bit of functionality.
- ✘ Programmers also write code integration tests to make sure the small units of code work together as intended.
- ✘ This essential practice has been adopted by many teams, even those that don’t call themselves “agile,” because it’s just a smart way to think through your software design and prevent defects.

✘ This isn’t about unit-level or component-level testing, but these types of tests are critical to a successful project.

✘ Brian Marick [2003] describes these types of tests as “supporting the team,” helping the programmers know what code to write next.

✘ Brian also coined the term “technology-facing tests,” tests that fall into the programmer’s domain and are described using programmer terms and jargon.

✘ Working harder and longer doesn’t help when your task is impossible to achieve.

✘ Agile development acknowledges the reality that we only have so many good productive hours in a day or week, and that we can’t plan away the inevitability of change.

✘ Agile development encourages us to solve our problems as a team.

✘ Business people, programmers, testers, analysts—everyone involved in software development—decides together how best to improve their product.

✘ As testers, we’re working together with a team of people who all feel responsible for delivering the best possible quality, and who are all focused on testing.

✘ “agile testing” usually is about business facing tests, tests that define the business experts’ desired features and functionality.

✘ It includes just about everything beyond unit and component level testing: functional, system, load, performance, security, stress, usability, exploratory, end-to-end, and user acceptance.

✘ All these types of tests might be appropriate to any given project, whether it’s an agile project or one using more traditional methodologies.

- ✘ Agile testing doesn’t just mean testing on an agile project.
- ✘ Some testing approaches, such as exploratory testing, are inherently agile, whether it’s done an agile project or not.
- ✘ Testing an application with a plan to learn about it as you go, and letting that information guide your testing, is in line with valuing working software and responding to change.

- ✘ Exploratory Testing is a type of software testing where Test cases are not created in advance but testers check system on the fly.
- ✘ They may note down ideas about what to test before test execution. The focus of exploratory testing is more on testing as a “thinking” activity.

ROLES AND ACTIVITIES ON AN AGILE TEAM

Customer Team

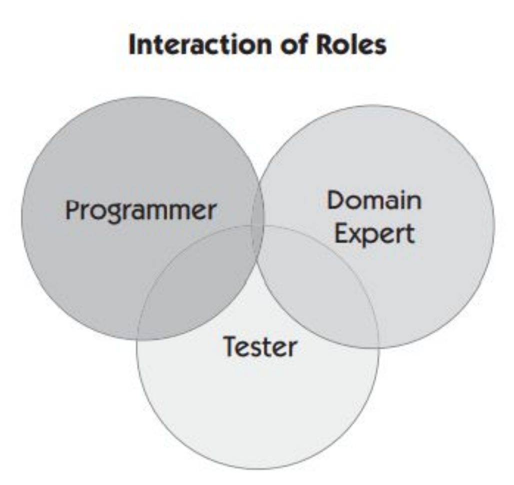
- The customer team includes business experts, product owners, domain experts, product managers, business analysts, subject matter experts—everyone on the "business" side of a project.
- The customer team writes the stories or feature sets that the developer team delivers.
- They provide the examples that will drive coding in the form of business-facing tests.
- They communicate and collaborate with the developer team throughout each iteration, answering questions, drawing examples on the whiteboard, and reviewing finished stories or parts of stories.
- Testers are integral members of the customer team, helping elicit requirements and examples and helping the customers express their requirements as tests.

Developer Team

- Everyone involved with delivering code is a developer, and is part of the developer team.
- Agile principles encourage team members to take on multiple activities; any team member can take on any type of task.
- Many agile practitioners discourage specialized roles on teams and encourage all team members to transfer their skills to others as much as possible.
- Programmers, system administrators, architects, database administrators, technical writers, security specialists, and people who wear more than one of these hats might be part of the team, physically or virtually.
- Testers are also on the developer team, because testing is a central component of agile software development.
- Testers advocate for quality on behalf of the customer and assist the development team in delivering the maximum business value.

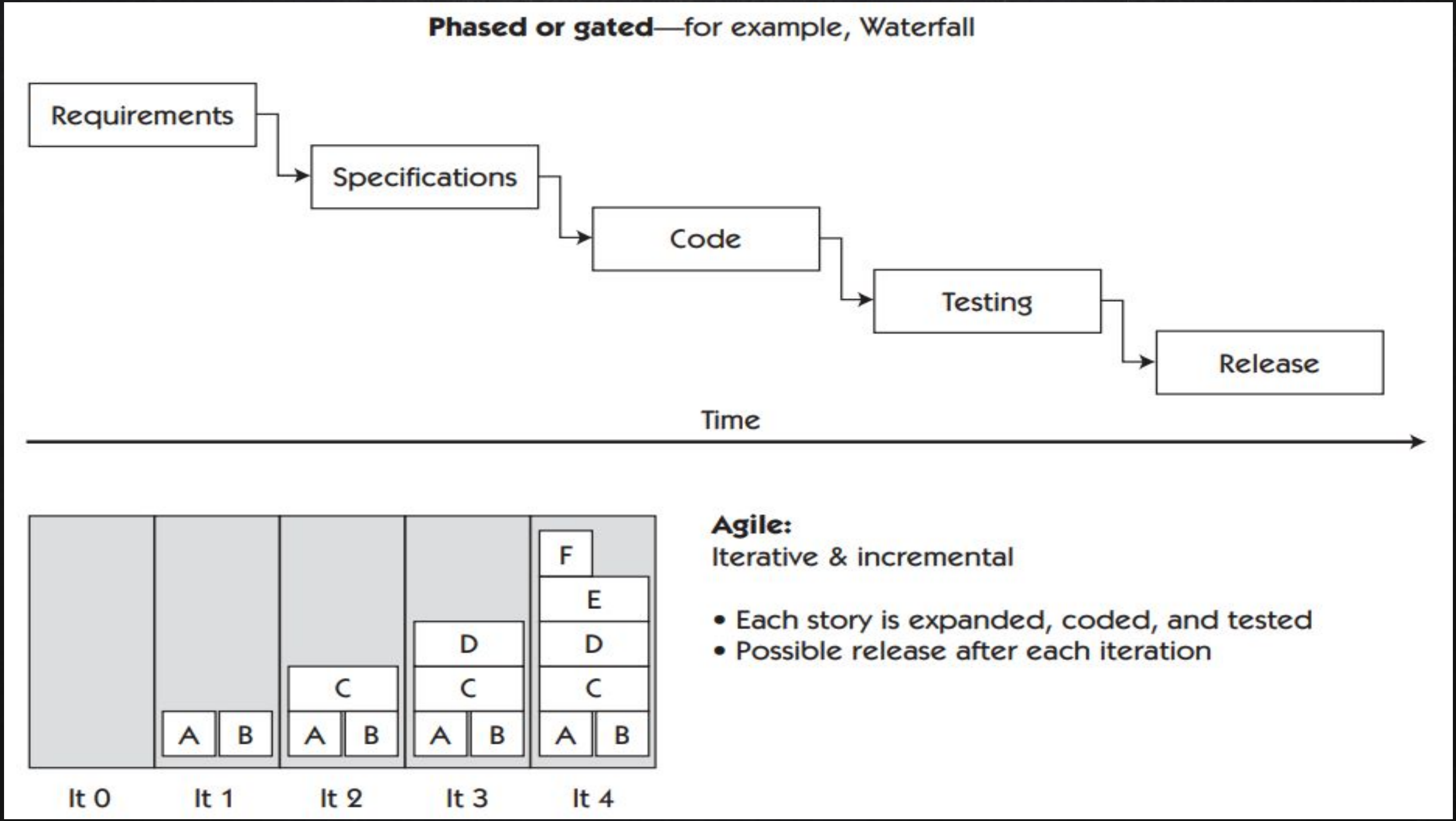
Interaction between Customer and Developer Teams

- ✘ Ideally, they're just one team with a common goal to deliver value to the organization.
- ✘ Agile projects progress in iterations, which are small development cycles that typically last from one to four weeks.
- ✘ The customer team, with input from the developers, will prioritize stories to be developed, and the developer team will determine how much work they can take on. They'll work together to define requirements with tests and examples, and write the code that makes the tests pass.
- ✘ Testers have a foot in each world, understanding the customer viewpoint as well as the complexities of the technical implementation



- Some agile teams may not have any members who define themselves as “testers.”
- However, they all need someone to help the customer team write business facing tests for the iterations’ stories, make sure the tests pass, and make sure that adequate regression tests are automated.
- Even if a team does have testers, the entire agile team is responsible for these testing tasks.
- Experience with agile teams has shown that testing skills and experience are vital to project success and that testers do add value to agile teams.

Traditional vs. Agile Testing



Phased or gated—for example, Waterfall

Agile:
Iterative & incremental

- Each story is expanded, coded, and tested
- Possible release after each iteration

✘ In the phased approach diagram, it is clear that testing happens at the end, right before release.

✘ The diagram is idealistic, because it gives the impression there is as much time for testing as there is for coding.

✘ In many projects, this is not the case. The testing gets “squished” because coding takes longer than expected, and because teams get into a code-and-fix cycle at the end.

✘ Agile is iterative and incremental. This means that the testers test each increment of coding as soon as it is finished.

✘ An iteration might be as short as one week, or as long as a month.

✘ The team builds and tests a little bit of code, making sure it works correctly, and then moves on to next piece that needs to be built.

✘ Programmers never get ahead of the testers, because a story is not “done” until it has been tested.

✘ Rather than creating tests from a requirements document that was created by business analysts before anyone ever thought of writing a line of code, someone will need to write tests that illustrate the requirements for each story days or hours before coding begins.

✘ This is often a collaborative effort between a business or domain expert and a tester, analyst, or some other development team member.

✘ Detailed functional test cases, ideally based on examples provided by business experts, flesh out the requirements.

✘ Testers will conduct manual exploratory testing to find important bugs that defined test cases might miss. Testers might pair with other developers to automate and execute test cases as coding on each story proceeds.

✘ Automated functional tests are added to the regression test suite. When tests demonstrating minimum functionality are complete, the team can consider the story finished.

WHAT'S AN AGILE TESTER?

✘ An agile tester is a professional tester who embraces change, collaborates well with both technical and business people, and understands the concept of using tests to document requirements and drive development.

✘ Agile testers tend to have good technical skills, know how to collaborate with others to automate tests, and are also experienced exploratory testers.

✘ They're willing to learn what customers do so that they can better understand the customers' software requirements.

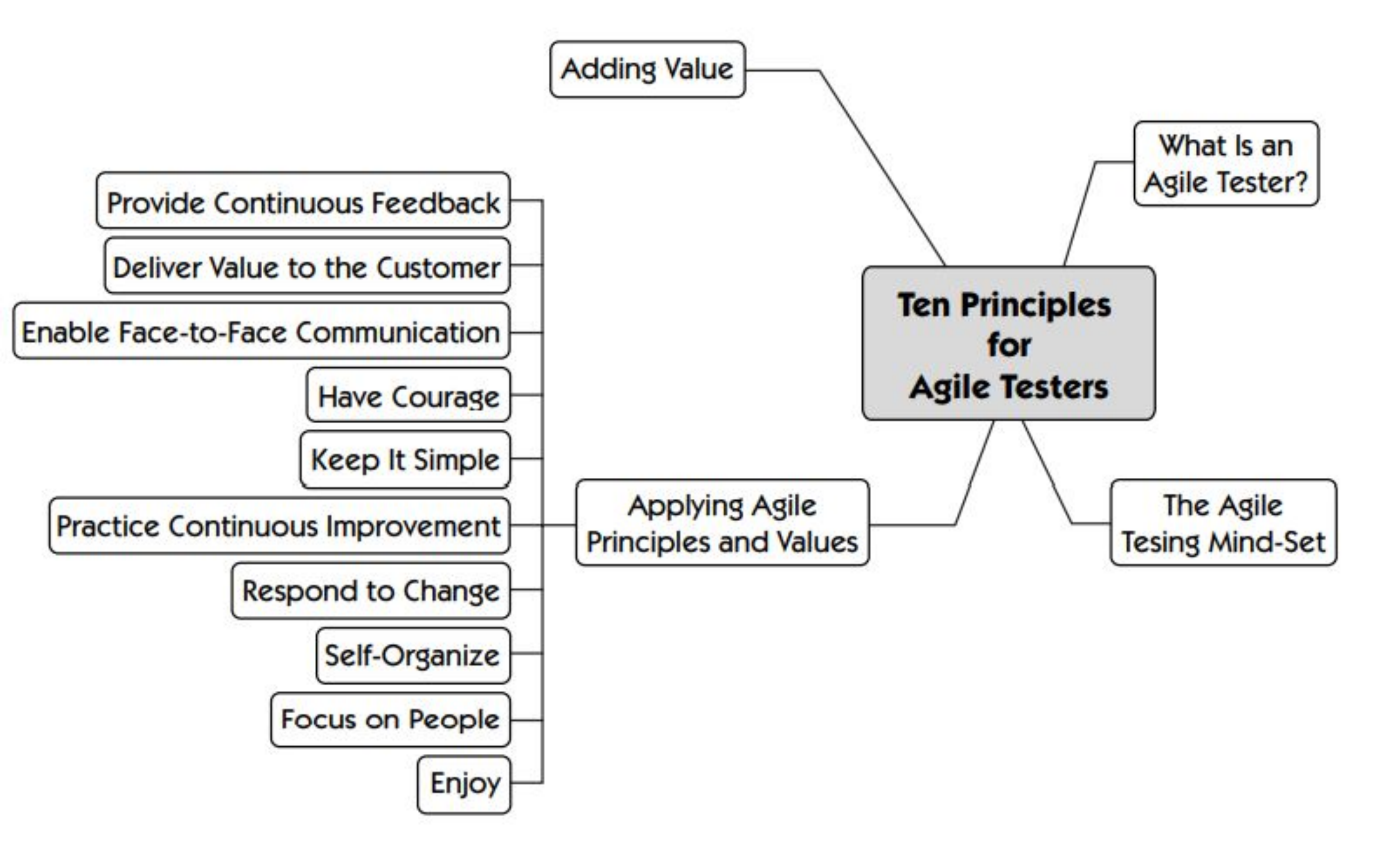
✘ Who's an agile tester? He/She is a team member who drives agile testing.

✘ A developer becomes test-infected and branches out beyond unit testing.

✘ An exploratory tester, accustomed to working in an agile manner, is attracted to the idea of an agile team.

✘ Professionals in other roles, such as business or functional analysts, might share the same traits and do much of the same work.

TEN PRINCIPLES FOR AGILE TESTERS



Adding Value
What Is an Agile Tester?
Provide Continuous Feedback
Deliver Value to the Customer
Enable Face-to-Face Communication
Have Courage
Keep It Simple
Practice Continuous Improvement
Respond to Change
Self-Organize
Focus on People
Enjoy
Ten Principles for Agile Testers
Applying Agile Principles and Values
The Agile Tesing Mind-Set

- The principles we think are important for an agile tester are:
    - Provide continuous feedback.
    - Deliver value to the customer.
    - Enable face-to-face communication.
    - Have courage.
    - Keep it simple.
    - Practice continuous improvement.
    - Respond to change.
    - Self-organize.
    - Focus on people.
    - Enjoy.

VALUE

- These principles bring business value.
- In agile development, the whole team takes responsibility for delivering high-quality software that delights customers and makes the business more profitable.
- This, in turn, brings new advantages for the business.
- Agile testers not only think about the system from the viewpoint of stakeholders who will live with the solution
- They also have a grasp of technical constraints and implementation details that face the development team. Programmers focus on making things work. If they're coding to the right requirements, customers will be happy.
- Unfortunately, customers aren't generally good at articulating their requirements.
- Driving development with the wrong tests won't deliver the desired outcome. Agile testers ask questions of both customers and developers early and often, and help shape the answers into the right tests.

- Agile testers take a much more integrated, team-oriented approach than testers on traditional waterfall projects.
- They adapt their skills and experience to the team and project.
- A tester who views programmers as adversaries, or sits and waits for work to come to her, or expects to spend more time planning than doing, is likely to cling to skills she learned on traditional projects and won’t last long on an agile team.
- During story estimating and planning sessions, agile testers look at each feature from multiple perspectives: business, end user, production support, and programmer.
- They consider the problems faced by the business and how the software might address them.
- They raise questions that flush out assumptions made by the customer and developer teams.

✘ At the start of each iteration, they help to make sure the customer provides clear requirements and examples, and they help the development team turn those into tests.

✘ The tests drive development, and test results provide feedback on the team’s progress.

✘ Testers help to raise issues so that no testing is overlooked; it’s more than functional testing.

✘ Customers don’t always know that they should mention their performance and reliability needs or security concerns, but testers think to ask about those.

✘ Testers also keep the testing approach and tools as simple and lightweight as possible.

✘ By the end of the iteration, testers verify that the minimum testing was completed.

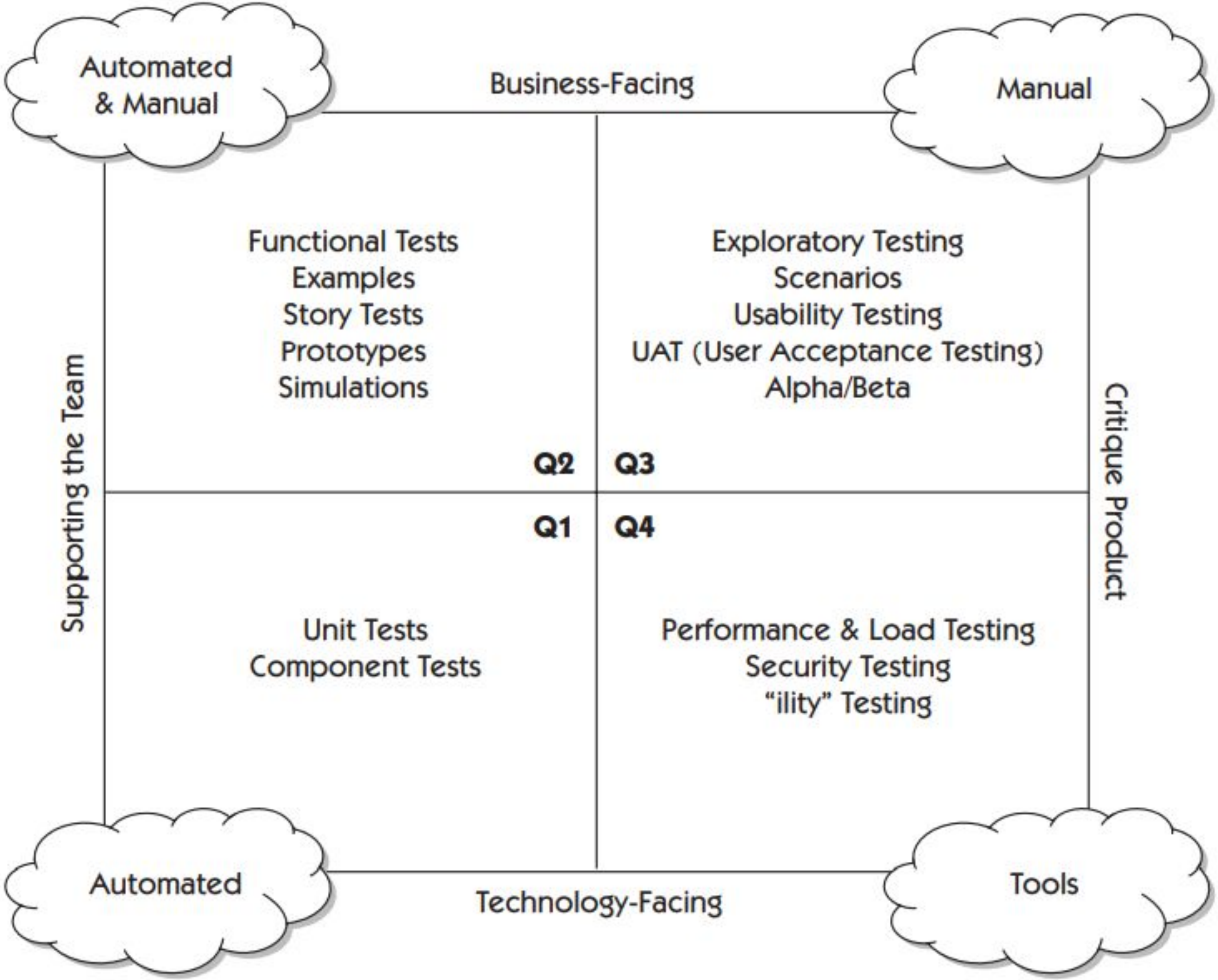
THE AGILE TESTING QUADRANTS

- Software quality has many dimensions, each requiring a different testing approach.
  - How do we know all the different types of tests we need to do?
  - How do we know when we’re “done” testing?
  - Who does which tests and how?
- Agile Testing Quadrants is used to make sure your team covers all needed categories of testing.

Why do we test?

- We test for a lot of reasons: to find bugs, to make sure the code is reliable, and sometimes just to see if the code’s usable.
- We do different types of testing to accomplish different goals.
- Software product quality has many components.

Agile Testing Quadrants



Automated & Manual

Business-Facing

Manual

Supporting the Team

Critique Product

Functional Tests
Examples
Story Tests
Prototypes
Simulations

Q2

Exploratory Testing
Scenarios
Usability Testing
UAT (User Acceptance Testing)
Alpha/Beta

Q3

Q1

Unit Tests
Component Tests

Q4

Performance & Load Testing
Security Testing
“ility” Testing

Automated

Technology-Facing

Tools

- Diagram of the agile testing quadrants shows how each of the four quadrants reflects the different reasons we test.
- On one axis, divides the matrix into tests that support the team and tests that critique the product.
- The other axis divides them into business-facing and technology-facing tests.
- Agile development starts with customer tests, which tell the team what to code.
- The timing of the various types of tests depends on the risks of each project, the customers' goals for the product, whether the team is working with legacy code or on a greenfield project, and when resources are available to do the testing.

Tests that Support the Team

- The quadrants on the left include tests that support the team as it develops the product.
- This concept of testing to help the programmers is new to many testers and is the biggest difference between testing on a traditional project and testing on an agile project.
- The testing done in Quadrants 1 and 2 are more requirements specification and design aids than what we typically think of as testing.

Quadrant 1

- The lower left quadrant represents test-driven development, which is a core agile development practice.
- Unit tests verify functionality of a small subset of the system, such as an object or method.
- Component tests verify the behavior of a larger part of the system, such as a group of classes that provide some service.
- Both types of tests are usually automated with a member of the xUnit family of test automation tools.
- This are referred as programmer tests, developer-facing tests, or technology-facing tests.
- They enable the programmers to measure what Kent Beck has called the internal quality of their code [Beck, 1999].

- A major purpose of Quadrant 1 tests is test-driven development (TDD) or test-driven design.
- The process of writing tests first helps programmers design their code well.
- These tests let the programmers confidently write code to deliver a story's features without worrying about making unintended changes to the system.
- They can verify that their design and architecture decisions are appropriate.
- Unit and component tests are automated and written in the same programming language as the application.
- A business expert probably couldn't understand them by reading them directly, but these tests aren't intended for customer use.
- In fact, internal quality isn't negotiated with the customer; it's defined by the programmers.
- Programmer tests are normally part of an automated process that runs with every code check-in, giving the team instant, continual feedback about their internal quality.

Quadrant 2

✘ The tests in Quadrant 2 also support the work of the development team, but at a higher level.

✘ These business-facing tests, also called customer-facing tests and customer tests, define external quality and the features that the customers want.

✘ With agile development, these tests are derived from examples provided by the customer team. They describe the details of each story.

✘ Business-facing tests run at a functional level, each one verifying a business satisfaction condition. They're written in a way business experts can easily understand using the business domain language.

✘ In fact, the business experts use these tests to define the external quality of the product and usually help to write them.

✘ It's possible this quadrant could duplicate some of the tests that were done at the unit level; however, the Quadrant 2 tests are oriented toward illustrating and confirming desired system behavior at a higher level.

✘ Most of the business-facing tests that support the development team also need to be automated.

✘ One of the most important purposes of tests in these two quadrants is to provide information quickly and enable fast troubleshooting.

✘ They must be run frequently in order to give the team early feedback in case any behavior changes unexpectedly.

✘ All of these tests should be run as part of an automated continuous integration, build, and test process.

✘ There is another group of tests that belongs in this quadrant as well.

✘ User interaction experts use mock-ups and wireframes to help validate proposed GUI (graphical user interface) designs with customers and to communicate those designs to the developers before they start to code them.

✘ The tests in this group are tests that help support the team to get the product built right but are not automated.

- tests in Quadrants 1 and 2 are written to help the team deliver the business value requested by the customers.
- They verify that the business logic and the user interfaces behave according to the examples provided by the customers.
- There are other aspects to software quality, some of which the customers don’t think about without help from the technical team.
- We need different tests to answer following types of questions.
  - Is the product competitive?
  - Is the user interface as intuitive as it needs to be?
  - Is the application secure?
  - Are the users happy with how the user interface works?

Tests that Critique the Product

- ✘ If you’ve been in a customer role and had to express your requirements for a software feature, you know how hard it can be to know exactly what you want until you see it.
- ✘ Even if you’re confident about how the feature should work, it can be hard to describe it so that programmers fully understand it.
- ✘ The word “critique” isn’t intended in a negative sense. A critique can include both praise and suggestions for improvement.
- ✘ Appraising a software product involves both art and science.
- ✘ We review the software in a constructive manner, with the goal of learning how we can improve it.
- ✘ As we learn, we can feed new requirements and tests or examples back to the process that supports the team and guide development.

Quadrant 3

- ✘ Business-facing examples help the team design the desired product, but <u>at least some of our examples will probably be wrong.</u>
- ✘ The business experts <u>might overlook functionality, or not get it quite right if it isn’t their field of expertise.</u>
- ✘ The team might simply misunderstand some examples.
- ✘ Even when the programmers write code that makes the business-facing tests pass, <u>they might not be delivering what the customer really wants.</u>
- ✘ That is where the tests to critique the product in the third and fourth quadrants come into play.
- ✘ Quadrant 3 classifies the business-facing tests that exercise the working software to see if it doesn’t quite meet expectations or won’t stand up to the competition.

✘ When we do business-facing tests to critique the product, we try to emulate the way a real user would work the application.

✘ This is manual testing that only a human can do. We might use some automated scripts to help us set up the data we need, but we have to use our senses, our brains, and our intuition to check whether the development team has delivered the business value required by the customers.

✘ User Acceptance Testing (UAT) gives customers a chance to give new features a good workout and see what changes they may want in the future, and it’s a good way to gather new story ideas.

✘ Usability testing gives Knowledge of how people use systems is an advantage when testing usability. Focus groups might be brought in, studied as they use the application, and interviewed in order to gather their reactions. Usability testing can also include navigation from page to page or even something as simple as the tabbing order.

✘ Exploratory testing is central to this quadrant. During exploratory testing sessions, the tester simultaneously designs and performs tests, using critical thinking to analyze the results. This offers a much better opportunity to learn about the application than scripted tests.

Quadrant 4

- ✘ The types of tests that fall into the fourth quadrant are just as critical.
- ✘ These tests are technology-facing, and we discuss them in technical rather than business terms.
- ✘ Technology-facing tests are intended to critique product characteristics such as performance, robustness, and security.
- ✘ For example, programmers might be able to leverage unit tests into performance tests with a multi-threaded engine.
- ✘ Agile's emphasis on having customers write and prioritize stories.
- ✘ Nontechnical customer team members often assume that the developers will take care of concerns such as speed and security, and that the programmers are intent on producing only the functionality prioritized by the customers.
- ✘ If all non-functional attributes are known before we start coding, it's easier to design and code with that in mind.

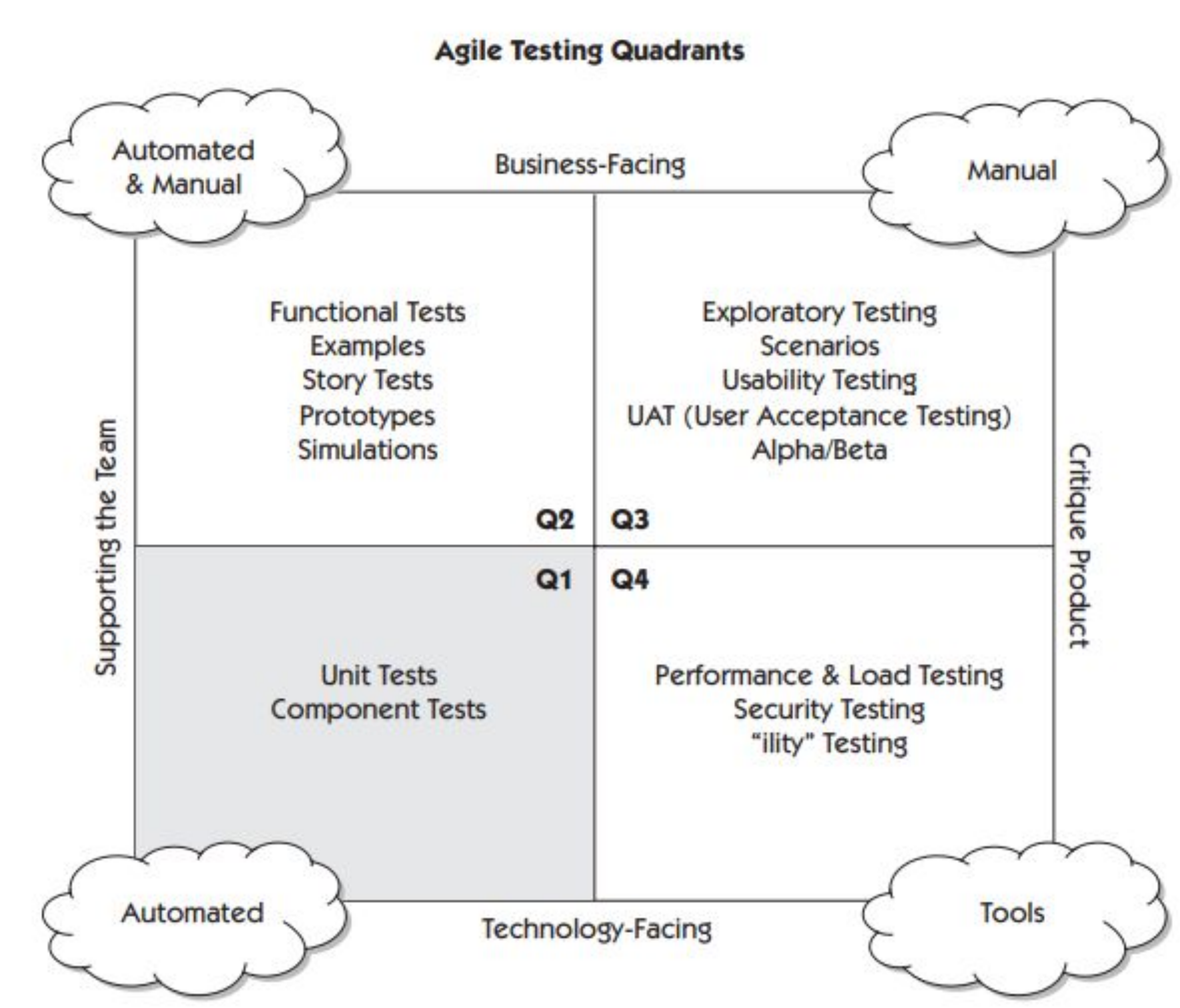
Using Tests that Critique the Product

- ✘ The information produced during testing to review the product should be fed back into the left side of our matrix and used to create new tests to drive future development.
- ✘ <u>For example, if the server fails under a normal load, new stories and tests to drive a more scalable architecture will be needed.</u>
- ✘ The short iterations of agile development give your team a chance to learn and experiment with the different testing quadrants.
- ✘ If the iteration demo reveals that the team misunderstood the customer's requirements, maybe you're not doing a good enough job of writing customer tests to guide development.
- ✘ If the team puts off needed refactoring, maybe the unit and component tests aren't providing enough coverage.
- ✘ Use the agile testing quadrants to help make sure all necessary testing is done at the right time.

Sharing Responsibilities

- ✘ Our product teams need a wide range of expertise to cover all of the agile testing quadrants.
- ✘ Programmers should write the technology-facing tests that support programming, but they might need help at different times from testers, database designers, system administrators, and configuration specialists.
- ✘ Testers take primary charge of the business-facing tests in tandem/ along with the customers, but programmers participate in designing and automating tests, while usability and other experts might be called in as needed.
- ✘ The fourth quadrant, with technology-facing tests that critique the product, may require more specialists.
- ✘ No matter what resources have to be brought in from outside the development team, the team is still responsible for getting all four quadrants of testing done.
- ✘ A successful team is one where everybody participates in the crafting of the product and that everyone shares the team's internal pain when things go wrong.

TECHNOLOGY-FACING TESTS THAT SUPPORT THE TEAM



Agile Testing Quadrants

Automated & Manual

Business-Facing

Manual

Supporting the Team

Critique Product

Q2

Functional Tests
Examples
Story Tests
Prototypes
Simulations

Q3

Exploratory Testing
Scenarios
Usability Testing
UAT (User Acceptance Testing)
Alpha/Beta

Q1

Unit Tests
Component Tests

Q4

Performance & Load Testing
Security Testing
"ility" Testing

Automated

Technology-Facing

Tools

✘ We discuss Quadrant 1 first because the technology-facing tests that support the team form the foundation of agile development and testing.

✘ Quadrant 1 is about much more than testing. The unit and component tests we talk about in Quadrant 1 aren't the first tests written for each story, but they help guide design and development.

✘ Without a foundation of test-driven design, automated unit and component tests, and a continuous integration process to run the tests, it's hard to deliver value in a timely manner.

✘ All of the testing in the other quadrants can't make up for inadequacies in this one.

✘ Teams need the right tools and processes to create and execute technology-facing tests that guide development.

The Purpose of Quadrant 1 Tests

- Unit tests and component tests ensure quality by helping the programmer understand exactly what the code needs to do, and by providing guidance in the right design.
- They help the team to focus on the story that's being delivered and to take the simplest approach that will work. Unit tests verify the behavior of parts as small as a single object or method [Meszaros, 2007].
- Component tests help solidify the overall design of a deployable part of the system by testing the interaction between classes or methods.
- Developing unit tests can be an essential design tool when using TDD.
- By building the code in small test-code-test increments, the programmer has a chance to think through the functionality that the customer needs.

- The term test-driven development misleads practitioners who don’t understand that it’s more about design than testing.
- Quadrant 1 activities are all aimed at producing software with the highest possible internal quality.
- When teams practice TDD, they minimize the number of bugs that have to be caught later on. Most unit-level bugs are prevented by writing the test before the code. Thinking through the design by writing the unit test means the system is more likely to meet customer requirements.
- When post-development testing time is occupied with finding and fixing bugs that could have been detected by programmer tests, there’s no time to find the serious issues that might adversely affect the business.
- The more bugs that leak out of our coding process, the slower our delivery will be, and in the end, it is the quality that will suffer.
- A team without these core agile practices is unlikely to benefit much from agile values and principles.

Supporting Infrastructure

- ✘ Solid source code control, configuration management, and continuous integration are essential to getting value from programmer tests that guide development.
- ✘ They enable the team to always know exactly what’s being tested. Continuous integration gives us a way to run tests every time new code is checked in.
- ✘ When a test fails, we know who checked in the change that caused the failure, and that person can quickly fix the problem.
- ✘ Continuous integration saves time and motivates each programmer to run the tests before checking in the new code.
- ✘ A continuous integration and build process delivers a deployable package of code for us to test.
- ✘ Agile projects that lack these core agile practices tend to turn into “mini-waterfalls”
- ✘ The development cycles are shorter, but code is still being thrown “over the wall” to testers who run out of time to test because the code is of poor quality

WHY WRITE AND EXECUTE THESE TESTS?

some reasons to use technology-facing tests that support the team.

- ➔ Lets Us Go Faster and Do More
- ➔ Making Testers' Jobs Easier
- ➔ Designing with Testing in Mind
- ➔ Timely Feedback

WHERE DO TECHNOLOGY-FACING TESTS STOP?

- For example, we have a story to calculate a loan amortization schedule and display it to a user who's in the process of requesting a loan.
  - A unit test for this story would likely test for illegal arguments, such as an annual payment frequency if the business doesn't allow it.
  - There might be a unit test to figure the anticipated loan payment start date given some definition of amount, interest rate, start date, and frequency.
  - Unit-level tests could cover different combinations of payment frequency, amount, interest date, term, and start date in order to prove that the amortization calculation is correct.
- They could cover scenarios such as leap years. When these tests pass, the programmer feels confident about the code
- Each unit test is independent and tests one dimension at a time.

- ✘ The business-facing tests very seldom/ rarely cover only one dimension, because they are tackled from a business point of view.
- ✘ The business-facing tests for this story would define more details for the business rules, the presentation in the user interface, and error handling.
- ✘ They would verify that payment details, such as the principal and interest applied, display correctly in the user interface.
- ✘ They would test validations for each field on the user interface, and specify error handling for situations such as insufficient balance or ineligibility.
- ✘ They could test a scenario where an administrator processes two loan payments on the same day, which might be harder to simulate at the unit level.
- ✘ The business-facing tests cover more complex user scenarios and verify that the end user will have a good experience.

WHAT IF THE TEAM DOESN'T DO THESE TESTS?

- ✘ When we're in a tester role, what can we do to help the development team implement TDD, continuous integration, and other practices that are key to successful development?
- ✘ Our experience over the years has been that if we aren't programmers ourselves, we don't necessarily have much credibility when we urge the programmers to adopt practices such as TDD.
- ✘ If we could sit down and show them how to code test-first, that would be persuasive, but many of us testers don't have that kind of experience.
- ✘ We've also found that evangelizing doesn't work. It's not that hard to convince someone conceptually that TDD is a good idea. It's much trickier to help them get traction actually coding test-first.

  - ○ What Can Testers Do?
  - ○ What Can Managers Do?
  - ○ It's a Team Problem

TOOLKIT

- ✘ Tools can help good people do their best work.
- ✘ Building up the right infrastructure to support technology-facing tests is critical.
- ✘ There's a huge selection of excellent tools available, and they improve all the time. Your team must find the tools that work best for your situation.
  - ○ Source Code Control
  - ○ IDEs
  - ○ Build Tools
  - ○ Build Automation Tools
  - ○ Unit Test Tools

Agile Testing Methods Overview

- ✘ In Agile Testing, the commonly used Testing methods are from the traditional practices and are aligned to the principle – Test Early.
- ✘ The Test Cases are written before the code is written.
- ✘ The emphasis is on defect prevention, detection, and removal running the right test types at the right time and at right level.

  - ○ Test Driven Development (TDD)
  - ○ Acceptance Test Driven Development (ATDD)
  - ○ Behavior Driven Development (BDD)

Test Driven Development

- ✘ 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.
- ✘ The tests can be at unit, integration or system level. Constant communication between testers and developers needs to be ensured.

Acceptance Test Driven Development

- ✘ In the 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.

Behavior Driven Development (BDD)

- ✘ BDD encourages communication between project stakeholders so all members understand each feature, prior to the development process.
- ✘ Behavior Driven Development (BDD) is similar to the Test Driven Development (TDD), and the focus is on testing the code to ensure the expected behavior of the system.
- ✘ In BDD, language like English is used so that it makes sense to the users, testers and developers. It ensures –
  - ○ Continuous communication among the users, testers and developers.
  - ○ Transparency on what is being developed and tested.

Thank You