# Test Driven Development (TDD):

If we look for the meaning of TDD, it is basically a programming practice where the developers bypass the duplication of code by writing new code only when the automated test fails. TDD’s fundamental aim is clearer, bug-free and simple code. The automated tests are run even before starting the actual development of the product and hence, people also refer it as **Test First Development.**

Here, we begin by drafting and establishing tests for single one of the functionalities of the product. This will later validate and specify the outcome of the code. But, since the tests are written before the development process, there is always a higher probability of failure. Therefore, developers need to constantly apply changes to the code i.e. refactoring and continue to do so until the test is passed.

The general steps to perform a TDD test is as follows:

1. Add a test
2. Run all tests and see if the new one fails
3. Write some code
4. Run tests
5. Refactor code
6. Repeat

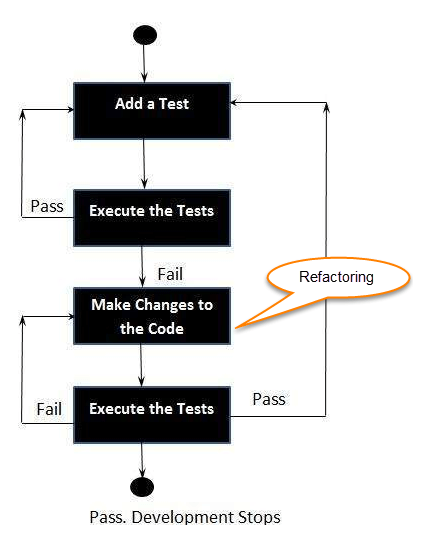


Figure : Steps of TDD test

A common misconception about automated tests is that it is time consuming. But in reality, automated tests help the developers along with the team to build the product much more quickly and with a higher quality. The followings are some of the benefits of TDD which might help to clear out the doubts and misconceptions of people.

* It asks us to acknowledge our desired outcome of the code.
* Quick feedback.
* Increases time efficiency via refactoring.
* Less loss of time in the debugger.
* Creation of a thorough specification.
* Encourage evolution in design and comply with varying understanding of the problem.
* Forces radical simplification of code.
* Final unit tests are simple and can be utilized as documentation for the code.
* Fewer bugs and enhanced quality.
* Cost efficient development along with increased productivity of the developers.
* Leads to more flexible, modularized and extensible code which further leads to cleaner interfaces, relaxed coupling, and smaller but highly focused classes.
* Grants ability to painlessly update the software in case of new business requirements or other unexpected variables.

# Behavior Driven Development (BDD):

As for the definition, BDD is a union and refinement of processes of Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD) which utilizes samples in order to represent the characteristics/behavior of the system that are noted in a readable and recognizable language for all the people associated with development. One of the target of BDD is to offer a shared procedure and tools to promote communication among business analysts and software developers during the development phase in order to hand over a product carrying business value. Next up, it looks up into what a system should do and not do on its implementation method. It also sheds light on the visibility and readability. Last but not the least, it ensure that the product not only functions well but also meets the expectations of the client.

Furthermore, it expands ATDD and TDD with accompanying strategies:

* Apply the "Five Why's" rule to each proposed client story, so its motivation is obviously identified with business results.
* To limit waste, implement thinking “from the outside in”, which means implementing just those practices that contribute more precisely to the same business results.
* Depict practices in a solitary documentation which is directly available to domain specialists, developers and testers, in order to improve communication.
* Apply these procedures right down to the most reduced degrees of deliberation of the product, giving specific consideration to the dispersion of behavior, with the goal that evolution stays modest.

## BDD Workflow:

The workflow if BDD can be essentially divided into three main parts:

1. **Define:** At first, the domain specialist or the business analyst drafts behaviors into a feature file which is typically written as standard sentences and is presents it to the team.
2. **Create:** With respect to the behaviors listed in the feature file, developers prepare a code known as “glue code”, which pictures behaviors to certain administration. An example is as follows:

***Given*** I am the user

***When*** I enter my login credentials in the login page

***Then*** I should be able to access the website

1. **Verify:** Next, the testers compare the glue code against the feature file in order to ensure that the application code correlates to the behaviors. This process is executed with the help of a BDD tool such as HipTest, Cucumber, etc. For the output, the feature file is compiled and pulled off to confirm the functionality.

Moving on to the advantages of using BDD, here are some of the expected benefits of it:

* It grants more clear-cut guidance on arranging the discussion between domain specialists, developers and testers.
* Documentations starting in the BDD method, specifically the **given-when-then** canvas, are nearer to ordinary day to day language and have a trivial learning curve contrasted with those of tools, for example, Fit/FitNesse.
* Tools focusing on a BDD approach for the most part bear the automatic generation of technical and end user documentation from BDD “specifications”.

## Tool used for BDD:

Out the varieties of tools available in the market, to name a few, HipTest, Easy B, J Behave, etc., we have used **Cuke Test**. The tool is most importantly free and it aids in drafting tests that are simple to understand for people even without Technical knowledge. Additionally, it helps to keep the test case scenarios and the requirement statement in the same place. Similarly, it is confined only for Web-driven automation, so it was also the selling point for us. The whole process of tracking the implementation is also easy with it as it displays the behavior sets that have been implemented and in working status.