|  |  |
| --- | --- |
| **BDD** | **TDD** |
| Behavior Driven Development is a development technique which focuses more on a software application’s behavior. | Test Driven Development is a development technique which focuses more on the implementation of a feature of a software application/product. |
| In BDD the participants are Developers, Customer, QAs. | In TDD the participants are developers. |
| Its main focus is on system requirements. | Its main focus is on unit test. |
| In BDD the starting point is a scenario. | In TDD the starting point is a test case. |
| It is a team methodology. | It is a development practice. |
| Here language used to write behavior/scenarios is simple English language. | Here language is used is similar to the one used for feature development like programming language. |
| In BDD collaboration is required between all the stakeholders. | In TDD collaboration is required only between the developers. |
| It is a good approach for project development which is driven by user actions. | It is a good approach for projects which involve API and third-party tools. |
| Some of the tools used are  Cucumber, Dave, JBehave, Spec Flow,  Concordian, BeanSpec etc. | Some of the tools used are  JBehave, JDave, Cucumber, Spec Flow, BeanSpec, FitNesse etc. |

**TDD**

**Test Driven Development (TDD) is a programming practice that instructs developers to write new code only if an automated test has failed. Test Driven Development (TDD) is software development approach in which test cases are developed to specify and validate what the code will do.**

**Advantages of TDD (Test Driven Development)**

**1. We have to write the code that is needed:** It provides a feature of writing the only production code that passes all the test cases. If we need to add some new feature to our project, then we just have to like a test case that drives the new feature to the project. We have to write code that is as simple as possible. So we have to write that code which is really needed for implementing the new features.

**2. More modular design**: In TDD (Test Driven Development), we consider only one micro feature at a time. We have to write the test case first; after that, the code is automatically easier to check. It is the modular design for the application.

**3. Easier to maintain**: In TDD (Test Driven Development), all the parts of the code are decoupled from each other, so it is very easy to maintain the whole project. We have to exchange the implementation of micro features without affecting the whole project. We have to write the entire application and test the program again. When we are done with the project, then we have to test the program again.

**4. Easier to refactor**: Every feature is tested by the TDD (Test Driven development). We do not have to worry about the project; if all the test case passes successfully, then the code is correct. But the important thing is, as a developer, we have to upgrade our skills daily. After improving the skill, if we open the project after a long day, then we have some ideas to improve our project. But if our memory is not fresh, then it is very dangerous to make any changes to the project. With the help of the test suite, we can make any type of changes without the fear of breaking the application.

**5. High test coverage**: In TDD (Test Driven development), there is a test for each feature. It gives high confidence to the user because it has high test coverage.

**6. Test document the code**: The test code shows us in which way our code will be used. It also documents our code. The test code is a type of sample code that shows the user what to do and what not to do.

Disadvantages of TDD (Test Driven Development)

**1. No silver bullet**: Tests help to seek out bugs, but they cannot find bugs you simply introduce within the test and implementation code. If you haven't understood the matter, you would like to unravel it, and writing tests most likely doesn't help.

**2. Slow process**: If you begin TDD, you'll get the sensation that you simply need extended time for straightforward implementations. You would like to believe the interfaces, write the test code, and run the tests before you finally start writing the code.

**3. All the members of a team got to do it**: As TDD influences the planning of code, it's recommended that all team members use TDD or nobody the least. Additionally, to the present, it's sometimes difficult to justify TDD to the management because they often have the sensation that implementing the latest features takes longer if developers write code that will not find them within the product half the time. It helps if the entire team agrees on the importance of unit tests.

**4. Tests got to be maintained when requirements change**: Probably, the strongest argument against TDD is that the tests need to be maintained because the code has got to. Whenever requirements change, you would like to vary the code and tests. But you're working with TDD. This suggests that you simply got to change the tests first and then make the tests pass. So, this disadvantage is the same as before when writing code that takes an extended time takes a long time.