**Difference Between TDD and BDD**

| **Sr No.** | **TDD** | **BDD** |
| --- | --- | --- |
| 1 | This is driven by the developers. | This is driven by developers, QAs, product owners, customers and business analysts. |
| 2 | This is mostly focused on the coding implementation of the functionalities of the application. | This is mostly focused on the business scenarios of the product. |
| 3 | This is mainly used for unit testing. | This is mainly for making developers, testers, product owners, customers and business analysts agree on functional requirements of the application. |
| 4 | The popularly used tools are JDave, SpecFlow and so on. | The popularly used tools are Cucumber, Gherkin, BeanSpec and so on. |
| 5 | TDD is known as Test Driven Development. | BDD is known as Behavior Driven Framework |
| 6 | The designing of test cases is the starting point of TDD. | The designing of scenarios is the starting point of BDD |
| 7 | TDD is used for projects involving third party tools and APIs. | BDD is used for projects involving the end user’s interaction |
| 8 | TDD requires team members having technical knowledge. | BDD does not require team members with technical knowledge. |
| 9 | The probability of having defects in the application is less. | The probability of having defects in the application is more compared to TDD. |
| 10 | Requires coordination among the developers. | Requires coordination among the developers. |
| 11 | Test cases are written with the help of any programming language. | Test cases are written in plain English. |

* **What is TDD?**

TDD is Test Driven Development. This means writing a test that fails because the specified functionality doesn't exist, then writing the simplest code that can make the test pass, then refactoring to remove duplication, etc. You repeat this Red-Green-Refactor loop over and over until you have a complete feature.

Behavior driven development should be focused on the business behaviour, your code is implementing: **the “why” behind the code**. It supports a team-centric (especially cross-functional) workflow. Behavior Driven testing is an extension of TDD. Like in TDD in BDD also we write tests first and the add application code.

**Advantages of TDD:**

* **You only write code that’s needed –**to prevent writing production code when all of your test’s pass. If your project needs another feature, you would like a test to drive the implementation of the feature. The code you write is the simplest code possible. So, all the code ending up within the product is really needed to implement the features.
* **More modular design –**In TDD, you consider one microfeature at a time. And as you write the test first, the code automatically becomes easy to check. Code that’s easy to check features a clear interface. This leads to a modular design for your application.
* **Easier to maintain –**Because the different parts of your application are decoupled from one another and have clear interfaces, the code becomes easier to take care of, you’ll exchange the implementation of a microfeature with a far better implementation without affecting another module.
* **Easier to refactor –**Every feature is thoroughly tested. you do not get to be afraid to form drastic changes because if all the tests still pass, everything is ok. Now, is extremely important because you, as a developer, improve your skills each and each day.
* **High test coverage –**There’s a test for each feature. This leads to a high-test coverage It develops gain confidence in your code.
* **Tests document the code –**The test code shows you ways your code is supposed to be used. As such, it documents your code. The test code is a sample code that shows what the code does and the way the interface has got to be used.
* **Less debugging –**How often have you ever wasted each day to seek out a nasty bug? How often have you copied a mistake message from Xcode and looked for it on the web.

**Drawbacks of TDD:**

* **No silver bullet –**Tests help to seek out bugs, but they cannot find bugs that you simply introduce within the test code and in implementation code. If you haven’t understood the matter you would like to unravel, writing tests most likely doesn’t help.
* **slow process –**If you begin TDD, you’ll get the sensation that you simply need an extended duration of your time for straightforward implementations. you would like to believe the interfaces, write the test code, and run the tests before you’ll finally start writing the code.
* **All the members of a team got to do it –**As TDD influences the planning of code, it’s recommended that either all the members of a team use TDD or nobody in the least.
* **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 then make the tests pass. So, actually, this disadvantage is that the same as before when writing code that apparently takes an extended time takes a long time.
* **What is BDD?**

BDD is Behavior Driven Development. This means creating an executable specification that fails because the feature doesn't exist, then writing the simplest code that can make the spec pass. You repeat this until a release candidate is ready to ship.

**Advantages Of BDD:**

**Reduce waste**: BDD is focused on discovering and developing features that brings high business value, so any feature that does not bring value will be ignored.

**Reduce costs**: Because of the reduced waste, it means that the development process will be focused just on the features that matters, so any additional cost is reduced.

**Easier and safer changes**: The requirements changes will be easier to implement because of the living documentation and safer because of the executable specification, which will act as automated acceptance and unit tests.

**Faster releases**: The releases will become more frequent as the large number of automated tests will eliminate some of the manual testing done by the testers. The testers will have instead more time to focus on other more complicated tests and scenarios.

**Drawbacks of BDD:**

**BDD requires high communication and collaboration:** BDD practices are very focused on discussions and collaboration between all parties involved. In case the business owners are unwilling or unable to engage in conversions and give feedback in time, then it will be hard to take advantage of BDD.

**BDD works best in an Agile or iterative context**: One of the BDD principles assume that it is very hard to know all the requirements from the beginning and that we don’t have to define them all in the first phase of a project, but instead the knowledge of the stakeholders will evolve during the lifetime of a project. That is why it is considered that BDD is well suited to be used in conjunction with Agile or iterative methodologies.

**BDD does not work not well in an enclosed environment**: In some cases, for example large projects developed by corporations, which delegates the work to multiple remote teams, will be harder for them to enable the high collaboration between teams.

**Poorly written tests can lead to higher test-maintenance costs**: For some complex applications, it will require experience to design and write the automated acceptance tests. In case the application will acquire many poorly written tests, they will become hard to maintain.