**TDD/BDD/ATDD** are software development *techniques* that can be used in any methodology although aspects of all three are often part of a team's agile approach.

* **TDD** is Test-Driven Development: the idea is that unit tests are written *first*, then enough code is written to make the tests pass. The pure TDD cycle is to write one failing unit test, then enough code to pass the test. Then a second failing unit test, then enough new code to pass both tests. And so forth.
* **BDD** is Behavior-Driven Development: this technique operates at a slightly higher level than TDD while still following the basic principle of writing the test, then coding to pass the test. BDD is usually the lowest level that will use the *Given-When-Then* pattern to describe the test (e.g. "Given that I have logged in, When I click the My Orders link Then I will be directed to the Order List page"). It can be difficult to distinguish between BDD and ATDD - the difference here is subtle.
* **ATDD** - is Acceptance-Test-Driven Development: this and BDD are often, in my experience, used interchangeably, particularly if the acceptance test is expressed in the Given-When-Then pattern (such as: "Given that I am a logged in user, When I go to My Orders Then I will see a list of all the orders I have made in the system, ordered from the most recent to the oldest.")

### TDD

TDD or Test-Driven Development is a process for when you write and run your tests. Following it makes it possible to have a very high test-coverage. Test-coverage refers to the percentage of your code that is tested automatically, so a higher number is better. TDD also reduces the likelihood of having bugs in your tests, which can otherwise be difficult to track down.

The TDD process consists of the following steps:

1. Start by writing a test
2. Run the test and any other tests. At this point, your newly added test should fail. If it doesn’t fail here, it might not be testing the right thing and thus has a bug in it.
3. Write the minimum amount of code required to make the test pass
4. Run the tests to check the new test passes
5. Optionally refactor your code
6. Repeat from 1

It can take some effort to learn well, but spending the time can pay off big. TDD projects often get a code-coverage of 90-100%, which means maintaining the code and adding new features is easy. This is because you have a large set of tests, so you can trust your code and changes work, and didn’t break any other code either.

Some think you must use the “xUnit style” testing tools to use the TDD process. This is not the case – TDD works great with unit tests, but you can apply it to other testing methods as well. It also does not require any specific tool or syntax.

The most difficult thing about TDD for many developers is the fact you have to write your tests before writing code. Check here for my [5 step method to make TDD easy](https://codeutopia.net/blog/2016/10/10/5-step-method-to-make-test-driven-development-and-unit-testing-easy/)

### BDD

BDD – Behavior-Driven Development – is perhaps the biggest source of confusion. When applied to automated testing, BDD is a set of best practices for writing great tests. BDD can, and should be, used together with TDD and unit testing methods.

One of the key things BDD addresses is implementation detail in unit tests. A common problem with poor unit tests is they rely too much on how the tested function is implemented. This means if you update the function, even without changing the inputs and outputs, you must also update the test. This is a problem because it makes doing changes tedious.

**Example ;**

### The characteristics of a good story

Using the example from the article [Introducing BDD](https://dannorth.net/introducing-bdd), let’s look at the requirements for withdrawing cash from an ATM:

|  |
| --- |
| **Story: Account Holder withdraws cash**   As an Account Holder  I want to withdraw cash from an ATM  So that I can get money when the bank is closed    **Scenario 1**: Account has sufficient funds  Given the account balance is \$100   And the card is valid   And the machine contains enough money  When the Account Holder requests \$20  Then the ATM should dispense \$20   And the account balance should be \$80   And the card should be returned    **Scenario 2**: Account has insufficient funds  Given the account balance is \$10   And the card is valid   And the machine contains enough money  When the Account Holder requests \$20  Then the ATM should not dispense any money   And the ATM should say there are insufficient funds   And the account balance should be \$20   And the card should be returned    **Scenario 3**: Card has been disabled  Given the card is disabled  When the Account Holder requests \$20  Then the ATM should retain the card  And the ATM should say the card has been retained    **Scenario 4**: The ATM has insufficient funds  ... |

As you can see, there are a number of scenarios to consider, some related to the account balance, others about the card and yet others about the ATM itself. Let’s dissect the story to determine whether it is any good.