TDD: Software is developed iteratively.

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

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.

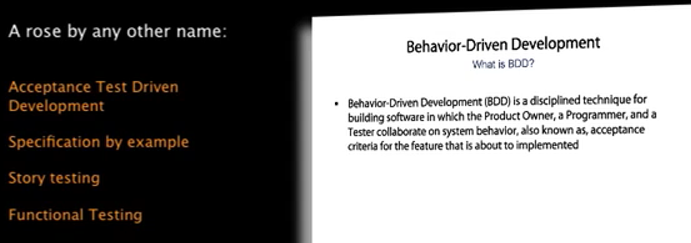
TDD works great with unit tests, but you can apply it to other testing methods as well. It also does not require any specific tool or syntax.

The most difficult thing about TDD for many developers is the fact you have to write your tests before writing code

TDD mainly uses automated unit tests to give developers direction on how to design the software. This approach follows a set process. First, create a test that will make the code fail. This helps to ensure that the test is calling the correct code and not working by accident. Then, teams will improve that code to make the test pass. The final stage of this method involves refactoring and removing duplication. It's expected that teams will repeat this cycle throughout the project's lifecycle, making it a perfect complement to Agile and [automation testing](https://www.utest.com/courses/test-automation-overview) efforts

Test-driven development gives you the when

BDD: It is a collection of tools and methodologies based on TDD

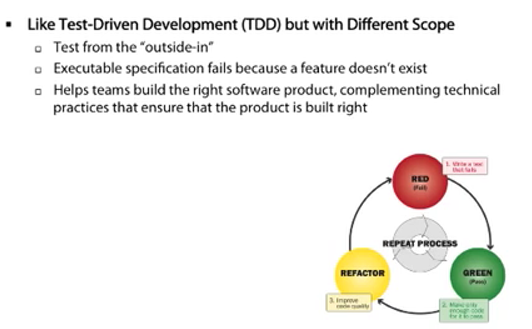


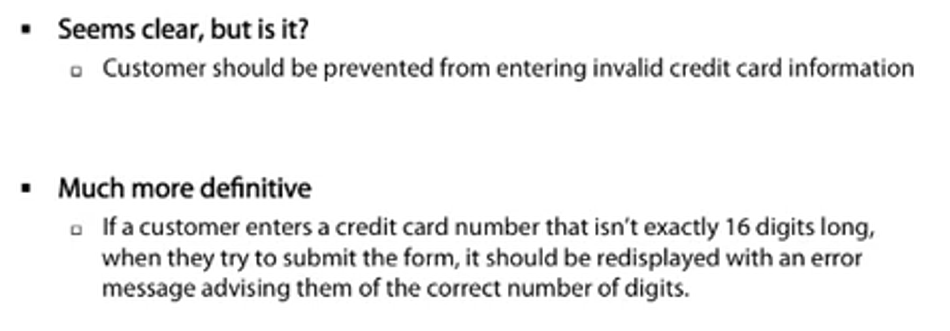
Given< >

When<>

Then<>

Fed to automated tool like cucumber





In Behavior-Driven development there is an approach called **Outside-In Development** where we go the other way round:

We write our scenario first, and in order to make it work we write examples for single components. The application code comes last. This is our development cycle:

1. Specify a scenario
2. Run the scenario and watch it fail 2.1. Specify the behavior of a component in an example to fix the failure 2.2. Run the component specification and watch it fail 2.3. Implement the minimum functionality necessary to make the example work
3. Run the feature again. If it still fails, continue with 2.1.
4. The scenario works! Write another scenario until the feature is complete. Developing software Outside-In has numerous advantages. Watch the screencast to find out more!

BDD helps mitigate issues caused by TDD and is driven by behavior and specification.BDD is telling the code exactly what you want it to do. This is a major difference, as BDD directly impacts the actual design of the software, while TDD is more focused on the testing.

Behavior-driven development gives you the how

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.

The software tends to be better designed [with BDD], that is, loosely coupled and easily maintainable, because the developer is free to make design decisions and refactor at any time with confidence that the software is still working

Although you can use each individually, you should combine them for best results as they complement each other very nicely