

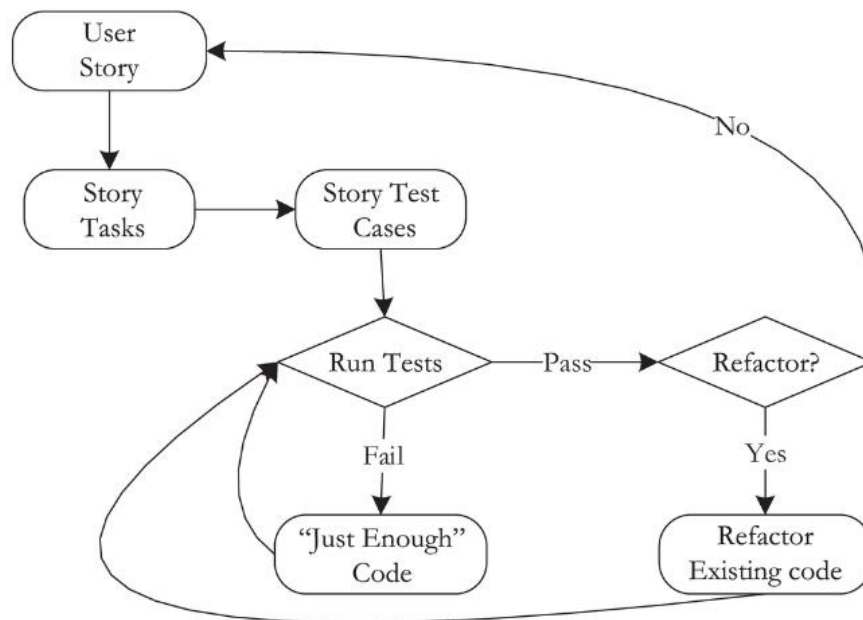
## Tutorial on Test Driven Development (TDD)

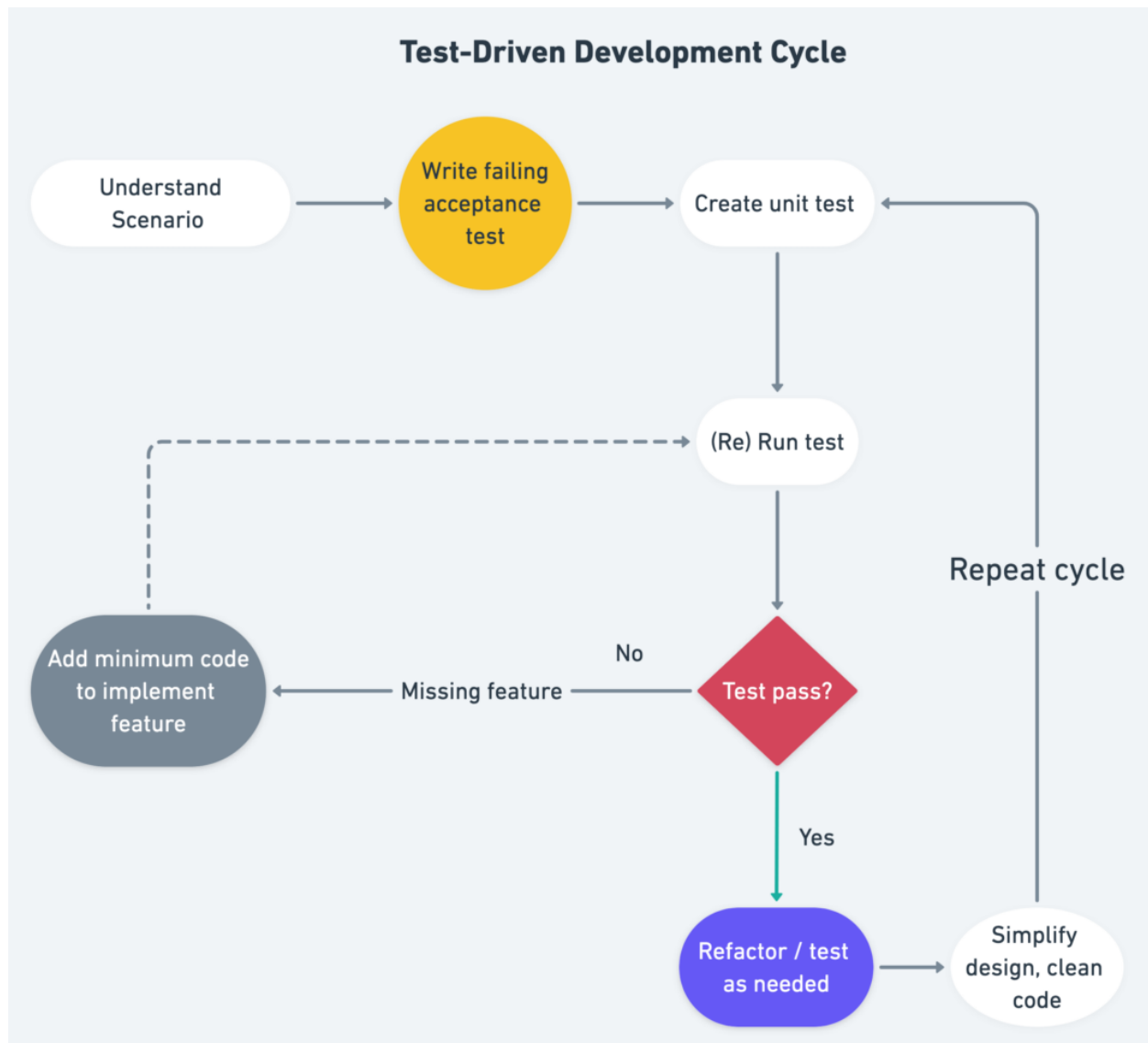
### What is Test Driven Development (TDD)?

Test-Driven Development (TDD) is a software development process, which includes test-first development. It means that the developer **first writes a fully automated test case before writing the production code** to fulfil that test and refactoring.

In TDD, code is not considered done, until all tests pass. TDD is a testing approach used in eXtreme Programming (XP). XP is a variation of Agile based Software Development Life Cycle.

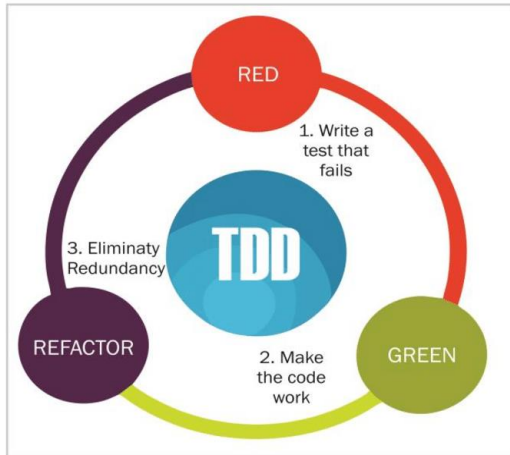
According to TDD, at first the developer writes automated test cases for the functions specified in User Stories. Obviously, those test cases will be failed initially. Because, still there is no written production code. Next, developers produce the minimum amount of code to pass that test. This time, test cases need to be passed. When the production code passes the test, that means the requirements or User Stories are satisfied by developers. After that, developer can refactor the code to make it better. Following is the TDD lifecycle.





### TDD Cycle - Red-Green-Refactor (RGR) Cycle:

TDD is often described as a Red, Green, Refactor cycle. Let us see what it is.



### Red Phase:

In the red phase, you have to write a test on a requirement that you are about to implement. You have to write the test first without any implementation of actual code.

### Green Phase:

In the green phase, you have to write the actual implementation of the requirement or enough code which can pass the test that you wrote before. In this phase, you only write the implementation of one test. Only for one test, because in TDD you have to

follow the cycle, which are making a test, run and fails. Then make the implementation for the test, run and succeed. Then make another test for the same implementation, run and fails.

### Refactor Phase:

In this phase, you have to refactor your implementation which is you needed. You may need to refactor your code, because you need to add more requirements. or you have been wrote another test for the same function that you wrote for your previous test. You may also want to remove code duplication in your implementation. Refactor both new and old code to make it well structured.

As a summary, in TDD cycle, following steps are generally followed:

- (1) Add a test
- (2) Run all tests and see if the new one fails ----- **Red Phase**
- (3) Write some code
- (4) Run tests ----- **Green Phase**
- (5) Refactor code ----- **Refactor Phase**
- (6) Repeat

[Sometimes, you may need to repeat only step 5, until you get a better code.]

### An Example of TDD using JUnit:

**Using TDD, Develop a simple system to calculate course costs for students of a college.**

Base costs of the courses are given in the following table.

**Length** has represented the duration of course and **cost** has represented the cost of the course. All cost amount are represented in USD [US Dollar].

Length	cost
3 weeks	500
4 weeks	700
6 weeks	1100
8 weeks	1500
12 weeks	2000

Other factors affecting costs are as follows:

Residency: Students, who have opted for residency within the college dormitory, will pay additional 10% for the courses they will enroll.

School Loyalty: Per year of enrollment, \$50 will be off.

### **Solution:**

(1) From the above example, at first think about features or requirements, which we need to implement.

In this example, features/ requirements can be represented as following user stories:

- (a) As a student, I want to know about the base cost of each course.
- (b) As a student, I want to know about the discount cost of each course, when I enroll in the college more than one year.
- (c) As a student, I want to know about the additional cost of each course, when I take residency in college dormitory.
- (d) As a student, I want to know about the total cost of each course, so that the total cost will include base cost, additional cost and the discount cost.


[You can also think about more features/requirements]

(2) Next, create test cases based on the above-mentioned requirements. In this step, think about interfaces to access the requirements. You do not need to think about actual implementation code.

In this example,

- (i) The test case to test the feature/requirement (a) is

```
@Test
public void baseCostTest()
{
    int expected=1500;
    int actual=cal.baseCost(8);
    assertEquals(expected,actual);
}
```



Interface to  
access the  
feature/  
requirement

Obliviously, before add the *baseCostTest()* method, we need to create two classes *Student* and *Cost*.

(3) Next, we will run the test case. Obliviously, the test case will fail. Because, actual code is not implemented yet.

(4) Then, we need to implement the feature/requirement, so that the test case can be passed. Following code is added in the *Cost* class to implement the feature/requirement.

```

public int baseCost(int weeks) {
    int cost=0;
    switch(weeks)
    {
        case 3:
            cost=500;
            break;
        case 4:
            cost=700;
            break;
        case 6:
            cost=1100;
            break;
        case 8:
            cost=1500;
            break;
        case 12:
            cost=2000;
            break;
    }
    return cost;
}

```

(5) Now, we again run the test case, this time the test case will pass.

(6) Next, you can refactor the code for better structure.

(7) Next, add another test case and repeat all the steps.

Now, let assume new feature/requirements are added.

New feature/requirement is “Support towards many students per class”

- For each student, over the 20<sup>th</sup> one (21+students), give a 2% discount from the total cost. For example:
  - For 21 students, 2% discount
  - For 22 students, 4% discount
  - For 23 students, 6% discount
  - And So on.

From this new requirement, we can get the following user stories.

(e) As a Student, I want to know the cost of the course, if student numbers in the course exceeds 20.

We need to again follow TDD cycle to add this feature.