UDC 004.054

TEST-DRIVEN DEVELOPMENT - DEVELOPMENT THROUGH TESTING: ADVANTAGES AND DISADVANTAGES

N. GURTOVENKO, O. GOLUBEVA
Polotsk State University, Belarus

This article is devoted to the TDD development method. Its advantages and disadvantages are analyzed.

Introduction. Software testing is a procedure that allows you to confirm or deny the efficiency of the code and the correctness of its operation. During testing, the input data is transmitted to the application and the execution of a certain command is requested, after which the results are checked for compliance with the standard, if the result is as expected, the test is considered passed. This procedure can be automated, in this case, checking the operability and correct operation of the application in comparison with manual testing is much faster, more complete and actually more often.

To create a successful IT project, an important step is the selection of a development methodology that contains its own approach to the process of creating programs in the form of steps, tasks, actions. There are many such methods, each has its own advantages and disadvantages, and the choice mainly depends on the task. One of them is Test-Driven Development method – development through testing. Methods of counteracting attacks on DW systems.

The main part. TDD is a software development process that requires a programmer to write an automated test case for the required functionality (initially faulty), then add the minimum code necessary to pass the test and, finally, reorganize the code and make sure that the automated tests still pass. In traditional methods, code is first written, and then test cases are developed to test the code. In TDD, test cases (usually called unit tests) to satisfy the requirement are written before the implementation begins.

TDD consists of a "red – green – refactor" cycle

Stage Red. A test is written that represents the necessary behavior of the system, then a stub method is created so that you can quickly assemble the project. The test compiles, but does not return the desired result. There is a need to write a program.

Stage Green. Minimal code is written that would allow the test to execute correctly. In reality, this means that code is written without structure, without design, without any design patterns. Now there is a need to give the code a "beautiful look".

Refactoring phase. The external structure of the code changes, without changing its external behavior. This division into methods, adding elements of a design pattern, creating additional classes, etc. The sequence of steps in a cycle is very important. The principle of the Test First method implies that only code is written that is absolutely necessary for all tests to pass successfully.

The TDD operation algorithm can be described as follows:

- Add a test for new (not yet implemented) functionality or for playing an existing bug.

When developing through testing, adding each new feature to the program begins with writing a test. Inevitably, this test will not pass because the corresponding code has not been written yet. (If the written test passes, it means that either the proposed "new" functionality already exists or the test has flaws.) To write a test, the developer must clearly understand the requirements for the new feature. For this, possible usage scenarios and user stories are considered. New requirements may also entail changes to existing tests. This distinguishes development through testing from techniques when tests are written after the code has already been written: it makes the developer focus on requirements before writing code - a subtle but important difference.

- Run all tests and make sure the new test fails.

At this point, verify that the tests just written do not pass. This stage also checks the tests themselves: a written test can always take place and, accordingly, be useless. New tests should not pass for obvious reasons. This will increase the confidence (although it will not guarantee completely) that the test really tests what it was designed for.

- Write a code that will pass the test:

o Run the tests and make sure that they all were successful: passing the new test confirms the implementation of the new functionality or correcting the existing error, and passing the rest allows you to make sure that the previously implemented functionality is still working correctly.

o To do refactoring and optimization - it is advisable to carry out improvement of maintainability and speed of performance already after it turned out to achieve verifiable performance.

- Restart tests and make sure they still pass successfully.
- Repeat cycle.

The described cycle is repeated, realizing more and more new functionality. Steps should be made small, from 1 to 10 changes between test runs. If the new code does not satisfy the new tests or the old tests stop passing, the programmer should return to debugging. When using third-party libraries, you should not make such small changes that literally test the third-party library itself, and not the code that uses it, unless there is a suspicion that the library contains errors.

Methodology advantages:

- Reduced debugging time. If you do not use the TDD methodology, it will take less than pure time to write code. However, the time spent on tests pays off when debugging code and catching errors - and this part of the development process is always present, because it is almost impossible to write code without a single error, and this is normal. Thanks to TDD, there is no need to guess where the error is located, the tested code is easier to maintain.
- Convenience of changeability. Coverage with tests helps to avoid the situation when, when changing the code in one place, an error occurs in a completely different part of the code. In the end, this allows you to safely change or refactor the code, because if something goes wrong, then the tests will immediately handle the emergency situation.
- Tests, especially if they are written in an expressive style, can serve as the technical documentation of the project, which describes how the code should work. Such documentation is very useful for new developers in the project who, in order to get involved in the work, must first deal with the code. The test in this case serves as a concrete example of the use of code. Full coverage of the code with tests provides a huge practical documentation of the code that reflects the real state of the system.
- Modularity. One of the principles of development through testing provides that each function performs a certain, small part of the work, because it is simply impossible to test the "omnipotent" method, which performs a dozen functions in several threads. All this forces the program to be divided into modules so that all branches of the code can be tested.
- Full tests. There is a big difference between writing regular tests and TDD tests. When tests are written after implementation, it is likely to receive defective tests due to the fact that not all scenarios of the method can be taken into account. For example, it may be considered that the written method works because it passes the test, but in fact it may turn out that not all conditions are covered in the method with tests. Using TDD prevents this from happening because the condition cannot appear in code without a test.
- TDD is the best way to ensure that tests actually cover all requirements, not just code. In addition, TDD helps you avoid falling into the "trap": adding functionality that the client does not really need, but would be nice to have.

Disadvantages TDD:

- The ability to apply TDD is not always available. There are tasks that cannot be solved only with the help of tests. For example, these are tasks in the field of data security and interactions between processes. Sometimes it's difficult to immediately imagine how the module will look. In addition, it is impossible to solve with the help of TDD the development of databases, compilers and interpreters of programming languages, it is impossible to automate testing of the graphical interface and distributed objects.
- Initial requirements cannot always be understood and correctly interpreted. As a result, you can get an error in the test, in the code and in understanding. The danger of the situation lies in the fact that it seems that everything works correctly, because the tests pass the green stage. On all this, you can lose a lot of time.
- The need for test support. The base of the code with one hundred percent coverage of tests is almost doubled. And besides, this entire database needs to be documented, maintained, and refactored.

Conclusion. This methodology allows us to achieve the creation of an application suitable for automatic testing and a very good coverage of the code with tests, since TK is translated into the language of automatic tests, that is, everything that the program should do is checked. TDD also often simplifies software implementation: since redundancy is eliminated - if a component passes the test, then it is considered ready. If the existing tests pass, but the component does not work as expected, this means that the tests do not yet reflect all the requirements and this is an occasion to add new tests.

The architecture of software products developed in this way is usually better (in applications that are suitable for automatic testing, the responsibility between the components is usually very well distributed, and complex procedures that are performed are decomposed into many simple ones). The stability of the application

developed through testing is also higher due to the fact that all the main functional capabilities of the program are covered by tests and their performance is constantly checked. Accompanying projects where everything or almost everything is tested is very high - developers may not be afraid to make changes to the code, if something goes wrong, the results of automatic testing will inform about this.

REFERENCES

1. Imprium.ru [Electronic resource]. Access mode : https://imprium.ru/articles/test-based-development. – Access date : 09/20/2019.
2. Proglib.io. Programming site [Electronic resource]. Access mode : https://proglib.io/p/test-driven-development/. –Access date : 09/20/2019.
3. Itvdn.com. Programming site [Electronic resource]. Access mode : https://itvdn.com/en/video/test-driven-development. – Access date : 09/20/2019.