УДК 681.3 (075)

Stud. Weronika Kuszneruk
Supervisor prof. P. Urbanovich
(the John Paul II Catholic University of Lublin, Poland)

SOME FEATURES OF THE PRACTICAL USE OF THE AGILE TESTING TECHNOLOGIES

Until recently software testing was perceived by many as destructive activity. Information about defects were swept under the rug, and a need to hire a tester was considered an unnecessary expense. This approach, of course, brought with it many failures, which were often costly in consequence.

The evolution of software development has forced a change in approach to the way the software is tested. This led to the development of an Agile Testing methodology [1, 2]. One of the most important assumptions of this approach is that the tester is no longer the only person responsible for the quality of the software produced. Responsibility is shared by the whole project team. Therefore, the team must learn to communicate in order to be able to react better to events happening in the project. Better communication allows to avoid the problem of misunderstanding the requirements, and also enables better verification of customer needs, which can undergo frequent changes. Agile testing and agile software development methodologies are able to respond to such changes because in this approach software development is split into relatively short cycles, in which the client is constantly informed about the progress of work. The result of that is a project that meets the requirements of the end user. As team members' responsibility increases, so does their performance. This is because everyone working with a given software has a real impact on the success of this project, which is clearly visible in the agile approach as opposed to the cascade methodology.

The Agile approach uses the seven principles of testing. These principles are:

1) Testing Shows the Presence of Defects,
2) Early Testing,
3) Exhaustive Testing is Not Possible,
4) Testing is Context-Dependent,
5) Defect Clustering,
6) Pesticide Paradox,
7) Absence of Error.

They are at the core of good software testing practices. According to one of those principles, testing should start as early as possible. From this principle derives a concept of a test pyramid, which is used in agile testing. Such a pyramid is shown in Fig. 1 below. The pyramid shows the hierarchy of test levels. The amount of tests corresponds to a level in a pyramid - tests found on a base of it should be the most numerous. Typically, unit and integration tests are automated, which makes them relatively cheap and rather fast, and therefore at the bottom of the pyramid. The number of remaining tests should be successively smaller, with each level.

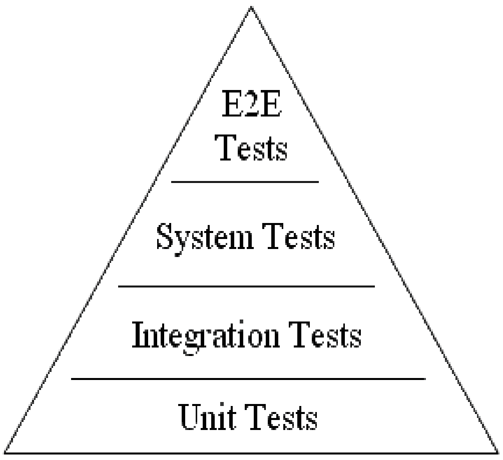

Figure 1 – Test pyramid

Agile testing uses several techniques that complement each other. This is, in turn, TDD, i.e. test driven development, ATDD, i.e. acceptance test driven development, and BDD, i.e. behavior driven development. The above techniques implement the testing principles mentioned above. What these techniques have in common is that they are all designed before actual software code is written. The first of the techniques, the TDD, is presented in Fig. 2. The assumption is that the person writing such a test creates it on the basis of an idea of a given functionality, before the development even starts.

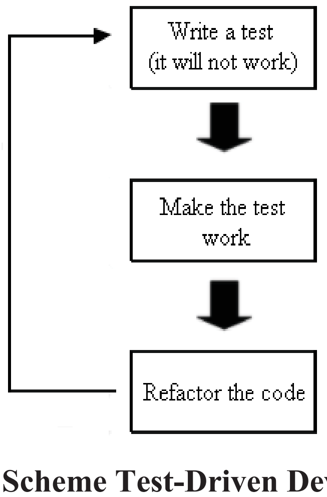

Figure 2 – Scheme Test-Driven Development

Then the tests and the newly created code of a given functionality should be run alternately until the test is successful. In the final stage, the code should be refactored.

TDD is responsible for defining the entry criteria and tests when creating user stories. The use of this technique allows for quick fixing of defects and software validation. BDD, in turn, allows to focus on the behavior of the software itself. The format of such a test is understandable to all team members and stakeholders.

Of course, agile testing is not an ideal solution, it is associated with some problems that directly affect the tester. A good example here is the aforementioned risk of changing customer requirements. This may lead to the need to redo some parts of the previously performed work several times. In the event of a change in requirements, the tester must rebuild the test cases he has created earlier, or perform the tests themselves again. This in turn requires additional time, which generates additional costs.

There are also concerns that a tester working in close contact with the development team, which is currently taking place when working with the use of agile methodologies, would lose objectivity. In my opinion, it is not the testing itself that is to blame, but rather the tester himself. Therefore, I would not consider it in the context of potential disadvantages. Each tester may have a different approach to testing, he tests in a different way using different methods. However, there is nothing wrong with that, because it is this diversity among testers that increases the chances that the software has been tested on various levels.

It should always be remembered that the reliability of the software determines the security of the information system [3, 4].

REFERENCES

1. Roman A., Testowanie i jakość oprogramowania, Warszawa: PWN, 2018.

2. Beck K., Sztuka tworzenia dobrego kodu, Gliwice: Helion, 2014.

3. Ochrona informacji w sieciach komputerowych / pod red. prof. P. Urbanowicza. – Lublin: Wydawnictwo KUL, 2004. – 150 p.

4. Urbanovich, P. P. Zashchitainformatsii: konspekt-lektsiya, ch. 2 = Information Protection, Part 2: BASIC METHODS / P. P. Urbanovich. – Minsk: BGTU, 2019. – 34 p.