Proceedings of the International Scientific Conference "Development through Research and Innovation - 2021",
IInd Edition, August 27, 2021, Chisinau, Republic of Moldova,
e-ISBN 978-9975-155-54-0, DOI 10.5281/zenodo.5732859

THE ROLE OF UNIT TESTING IN TRAINING

Dimitrichka Zheleva Nikolaeva, PhD
e-mail: dima.nikolaeva@abv.bg

Tehnical University of Varna
1 Studentska Street, 9000 Varna, Bulgaria
Web page: tu-varna.bg

Abstract: *The main priority of any modern software company is to improve the quality of the software. This can be achieved by preventing software defects, i.e. Software Testing (ST) applied by well-trained programmers. This puts before each university the task of professional training of students to master theoretical and practical aspects related to various techniques, strategies and methods in the field of ST. The ultimate goal is the creation, implementation, analysis and subsequent maintenance of the software.*
Key words*: Software Development Life Cycle, Unit Testing, Design Patterns, Anti-Patterns, SQL Server, Software Testing*

JEL CLASSIFICATION: O3

1. INTRODUCTION

The topic of the Software Testing (ST) is relevant. Currently, the funds that a company spends on testing reach 40% of the company's total Internet Technology (IT) budget. A number of authors comment on the need to create, implement and automate software testing in their works (Dustin, 2009), (Nelson, 2006), (Helfen, 2007), (Dustin, 2002), (Phillip, 2010), (Dustin, 2001). However, it is related to some problems, which the authors of (Ammann, 2008) comment on. For example, as early as 1990 year, Beizer noted that: "half of the work spent on developing a work program is spent on testing activities." In 2002 year, Hailpern and Santhanam commented that: "debugging, testing, and verification activities vary between 50% ÷ 75% of the total development costs." In 2008 year, Redmond Developer News wrote that: "those developers spend about 20% of their time designing and coding, and the rest of the time is spent fixing application problems". Everything described is proof of the need to learn, creating and implementing strategies and technologies for automated software testing, to improve software performance and quality. This can only be achieved with experienced, well-known to the standards certified programmers who undergo a training course at the university to acquire the necessiry knowledge and skills in the field of ST.

2. SOFTWARE TESTING - DEFINITION. SOFTWARE DEVELOPMENT LIFE CYCLE. CLASSIFICATION

Software testing (ST) is a phase of the Software Development Life Cycle (SDLC).

Table 1. Classification of Life Cycle Models (Maneva, 2001)

<table>
<tr><td rowspan="10">Full</td><td rowspan="7">One-dimensional</td><td rowspan="6">Chronological</td><td rowspan="3">Standard</td><td>Fight</td></tr>
<tr><td>Metzger</td></tr>
<tr><td>Freeman</td></tr>
<tr><td rowspan="2">Modified</td><td>Cascading</td></tr>
<tr><td>Prototype</td></tr>
<tr><td>Branched</td><td>Fox</td></tr>
<tr><td colspan="3">Functional (Hamilton-Celdin)</td></tr>
<tr><td rowspan="3">Multidimensional</td><td colspan="3">2D (Gunther)</td></tr>
<tr><td colspan="2" rowspan="2">Three-dimensional (Peter-Trip)</td><td>Evolutionary</td></tr>
<tr><td>Spiral</td></tr>
<tr><td>Partial</td><td colspan="4">(Appleton) for multiple use</td></tr>
</table>

e-ISBN 978-9975-155-54-0, DOI 10.5281/zenodo.5732859

ST includes processes related to research, evaluation and establishment of the completeness and quality o computer software. ST guarantees the compliance of the software product in relation to regulatory, business, technical, functional and user requirements. (Maneva, 2001) The purpose of testing processes related to software research and verification may be in relation to: Functionalities / business requirements - checking the full version of the software; Creation of software for errors - identification of technical errors; Assess usability, performance, security, localization, compatibility and installation and review others. The software is considered complete or usable only if it has passed each test. ST starts with a requirements collection phase and reaches the implementation of the software. ST depends directly on the model used. Certain SDLC are listed in Table 1. Upon detailed examination of the life cycle models (Maneva, 2001), (Sommerville, 2011) it is noticed that the testing phase is present in each of them, directly depending on the object and objectives of testing. In Table 2 an attempt is made to summarize them by registering 8 classification groups.

Table 2. Generalized Classification Based On Literature Sources

Classification № 1 (Maneva, 2001)	**Classification № 5: Machine learning in software testing-framework dimensions (Noorian, 2011)**
1.1.According to the selected test data and expected results 1.2.According to the level of testing 1.3.Depending on whether or not the internal structure of the software is ignored 1.4.According to the purpose 1.5.Specific types of testing 1.6.According to the submitted value of the input data	5.1.Testing Category 5.2.ML Category **Classification № 6: Software testing (Jacob, 2016)** 6.1.Unit Testing 6.1.1.Black Box Testing 6.1.2.While Box Testing 6.2.Integration Testing 6.3.System Testing
Classification № 2: According to testing methods (Kiran, 2016), (Kalin, 2010)	**Classification № 7: Static and Dynamic Testing (Functionize, 2018)**
Classification № 3: According to the level of testing (tutorialspoin, 2021) 3.1.Functionally 3.2.Non-functional **Classification №4** (Georgi, 2020) 4.1. Functional and Non-functional Tests 4.2. Black Box Testing (BBT) Techniques 4.3. White Box Testing (WBT) Techniques 4.4. Strategy for conducting BBT (Black Box Testing) 4.5. Strategy for conducting WBT	7.1.Static 7.1.1.Review 7.1.2.Static Analysis 7.2.Dynamic 7.2.1.Functional Testing 7.2.2.Non-Functional Testing **Classification № 8:** Manual and automated testing (SDA, 2020)

3. MANUAL AND AUTOMATED TESTING. STANDARDIZATION AND CERTIFICATION

3.1. Manual and Automated testing

According to classification 8 of Table 2 STs are divided into Manual and Automated. Manual testing is testing without the use of an automated tool or script. The tester is the end user. The stages for manual testing are: modular, integration, system and user acceptance testing. Automated testing, also known as Test Automation, is performed by a tester who writes scripts and uses other software to test the product. Test Automation is used to restart test scenarios that have been run manually, quickly, and repeatedly. The tools used in this test are (SDA, 2020), (Dustin, 2014), (myservername, 2021): HP Quick Test Professional; Selenium; IBM rational function tester; SilkTest; TestComplete; Testing everywhere; WinRunner; LoadRunner; Visual Studio Test Professional; WATIR. The main advantage of automation over manual testing is resource saving. Early start of the testing phase reduces the time for processing and production of error-free software delivered to the end customer. By reducing manual testing efforts, by increasing testing coverage (e.g., memory leak detection under specific conditions, parallelism test, performance test, etc.), development tools will also be reduced.

3.2. Standardization and certification

In connection with dealing with the above problems, software development companies often create and develop software testing standards themselves. In (IEEE, 2013) some known standards for software improvement are presented. For example, ATRT Display Automation (Dustin, 2014) specializes in automated software testing, including recording / recording and playback. The software allows them to automate the actions of the test engineer. To perform actions during testing, a tool is used that captures actions and information from the screen, which are based in an automated test script. During the test playback, the latest results are compared with the base results, using VNC technology - for remote connection to the tested system. The creation and implementation of software testing is done by experienced programmers. The staff is created by the university, where the precise selection of disciplines in the ST direction leads to the creation of well-trained staff for the practice. A number of other institutions offering certification on the basis of acquired experience and practice also provide an opportunity for raising the qualification of programmers. Some of the certificates that are issued as a result of proving competencies when taking an exam in the field of software testing are listed in (myservername, 2021). This ensures that experienced, certified professionals familiar with the standards and able to apply them will be preferred in the labor market.

4. FACTORS FOR SUCCESSFUL ST. AUTOMATED TEST TOOLS

4.1. Factors for a successful ST

The success of a software test depends on a number of factors, the most important of which are: Teamwork and involvement of testers in each stage of software development; Performing tests throughout the life cycle, not just by the QA team; Joint work of testers and developers, i.e. DevOps Shift-Left Practice; Implementing a flexible testing process, by automating the workflow; Availability of experienced staff; Application of functional testing; Workflow testing using different approaches, such as: dividing the tests into small fragments; application of regression testing; automation of software testing, applying various technologies, such as: Open source automation tools to be installed in the system, such as the cloud-based LambdaTest platform. It is among the leading tools for test automation for 2021. (Arsie, 2019)

4.2. Automated Test Tools

There are a variety of tools that are used to automate tests. For 2021 year, according to (myservername, 2021), among the first 10 instruments are: LambdaTest; TestComplete; QMetry Automation Studio; TestProject; Catalon Studio; testigma; Worksoft; QUALIBRATE; TWENTY ONE - Autonomous connection of testing and production; basis. For 2021 year, there are also test management tools according to (myservername, 2021) among the top 10 best tools are: Marshmallow Scale; PractiTest; GetZephyr; Collab's test; TestFLO for JIRA; XQual; Xray - Control of test edges; TestRail; Quality; Jira (RTM) Test Requirements and Management. Another classification for 2021 year according to (guru, 2021) indicates that Best Software Testing Services: Testio; QAlifed; Capgemini.

e-ISBN 978-9975-155-54-0, DOI 10.5281/zenodo.5732859

5. UNIT TESTING CONCEPTS, ADVANTAGES, FEATURES

5.1. Unit testing concepts

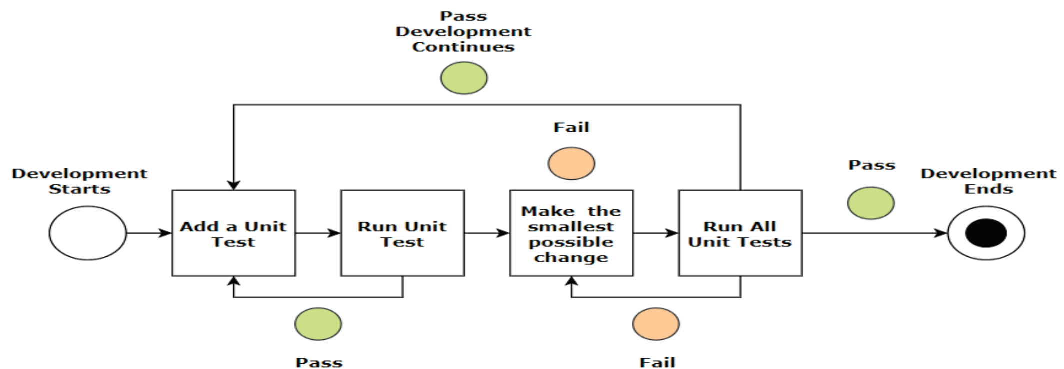

Figure 1. Unit test workflow diagram

To ensure the quality of the software, a number of traditional forms of testing are used, automated or manual forms are used to validate the behavior of the developed software. There are also various tests for loading the system, as well as tests with the participation of the user, which is a guarantee that the system works as the customer expects. The Unit test, unlike all the tests listed, focuses on a lower level. This type of testing belongs to the white box testing, which is based on the internal structure of the system. It is a functional test whose main purpose is to test the smallest "unit" of code. The Unit test is usually written in the same programming language as the source code of the application itself and is written to verify / test this code directly. In fact, the Unit test is generally code that tests another code. (manning, 2021) From the graph of Figure.1. the workflow of the Unit test can be traced.

5.2. Advantages of the Unit test (Khorikov, 2020)

- Saves money to compensate for the time spent on debugging at a later stage of system development;
- Allow restart, after a change has been made, which corresponds to the correctness of the data that meet the requirements;
- Storage of the test in the source near the code, convenient for checking and facilitating the synchronization of the main code and the test; belongs to the regression tests, which allows re-testing of part of the code, with added new functionality;
- Provides quality control when correcting errors in program code;
- Unit test can be created and run in Visual Studio development environment.

5.3. Features when creating a Unit test:

- Each Unit test adheres to the so-called model AAA (Arrange-Act-Assert). This is a model for structuring tests. According to it, the Unit test is divided into three parts - Arrangement (Setting), Action and Validation (Verification), and each of these parts is a step leading to the next. Step 1 is Stacking sets the input values of the test. Step 2 is Action, prompts the main function to be tested. Step 3 is Validation. The last third step confirms that the output of the function is what is expected. These parts are actually objects.
- In order to identify classes in the Unit test, the [TestClass] attribute must be added, through which the Unit methods are recognized.
- The Unit test method must be public, non-static, not accept parameters and not return a value. The TestMethod attribute must be added to distinguish the test from the regular method.

- A Unit test is successful or unsuccessful according to the thrown exception, if no exception is created; the test is successful, only the ExpectedException attribute makes an exception.
- The Assert.AreEqual method in the third part of the AAA model is used to compare between two values - the one expected by the programmer and the one generated by calling the created method. If they do not match, an exception is thrown that indicates the test failed. When starting the project, the result appears in a window. It can be in three variants successful, marked with a green mark, unsuccessful, marked with a red, unconvincing, marked with a question mark.
- The management and conduct of tests is done through Test Manager and Test View. The test view allows quick selection to run a test, with the option to group by name, project, type, class name and other criteria. The test manager offers the same features as Test View, but with additional options for displaying tests. You can organize a list of tests, filter tests and more.
- If it is necessary to configure a resource (i.e. connection to a database, log file, shared object) it may be necessary to clean up the actions of the tests, which is reduced to closing a shared stream or returning a transaction. Unit Test Framework offers attributes to identify such methods, as they are grouped into three levels: Test, Class, and Assembly, these levels determine the scope and execution time of the methods. Details of these attributes are provided in Table 3.
-

Table 3. Unit Test of Framework (manning, 2021)

Attributes	Frequency and Scope
TestInitialize, TestCleanup	Executed before (Initialize) or after (Cleanup) any of the class`s unit tests are run
ClassInitialize, ClassCleanup	Executed a single time before or after any of the tests in the current class are run
AssemblyInitialize, AssemblyCleanup	Executed a single time before or after any of the tests in any of the class`s unit tests are run

- Methods with the specified attributes do not have to appear in the test, but more than one attribute is not allowed in this context.
- In addition to the methods listed in the Unit tests, the following are also used:
- The method Assert.AreEqual and Assert.AreNotEqual, Assert.AreSame and Assert.AreNotSame, Assert.IsTrue and Assert.IsFalse, Assert.IsNull and Assert.IsNotNull, Assert.IsInstanceOfType and Assert.OsN
    - classes CollectionAssert, StringAssert, TestContext;
    - PrivateObject to access non-public instance members;
    - PrivateType for accessing non-public static members

6. UNIT TESTING IN TRAINING

One of the main sections studied in the disciplines of Software Technologies and Technologies for Software Production at the Technical University in Varna is Software Testing. In order to apply the acquired theoretical knowledge in these disciplines in the field of ST, in practice, in the laboratory, exercises were created 5 Unit tests, oriented to one of the most common problems in software development, namely for: mathematical methods; access to a private variable; to work with databases, specifically with SQL server; Design Patterns and Anti-Patterns.

Test №1 was performed in 5 variants, each of which is described in 4 main steps:

- Step 1: Generate code (interface / class / method) to be tested;
- Step 2: Create a Unit test on the generated code from Step 1;
- Step 3: Execution and visualization of the result of the performed Unit test;
- Step 4: Analysis and conclusions from the Unit test.

Tests №2 and №4 are presented only with Unit test. Tests №3 and №5 are implemented by basic code and Unit test.

Table 4. Unit tests

Test 1: Unit test for mathematical methods. Create a Unit test using the AAA model to test a method of Adding two real numbers

Option 1

Step1:
1.Create a Calculator class.
2.Add Sum method to implement mathematical operation Addition.

```
public class Calculator
{
   public double Sum(double num1, double num2)
   {
      return num1 + num2;
   }
}
```

Step2:
1.Create a project from the Test / Unit Test Project menu
2.Add class CalculatorTests to the Unit test, following the AAA template.

```
public class CalculatorTests
{
   [Fact]
   public void Sum_numbers()
   {
      // Arrange
      double num1 = 1;
      double num2 = 2;
      var calculator = new Calculator();
      // Act
      double result = calculator.Sum(num1, num2);
      // Assert
      Assert.Equal(3, result);
   }
}
```

Option 2

Step1:
1.Create a dll file using Class Library.
2.Create a Calculator class.
3.No Sum method to implement mathematical addition operation.

Step2:
1.Create a project from the menu Test / UnitTestProject1. Create a CalculatorTests class.
2.Add the created dll file from the menu References / Add Reference.
3.Observance of AAA, when observing the Unit test.

Option 3 – the ICalculate interface is added to the specified condition in Test 1

Step1:
1.Create a Calculate interface.
2.Create a Calculator class that inherits the ICalculate interface.

```
interface ICalculate
   {
      double ADD(double a, double b);
   }
```

Step2:
1.Create a project from the menu Test / UnitTestProject1. Create a CalculatorTests class.
2.Add a void TestSet () method.
3.Adding method void TestMethodADD ()
4.Compliance with AAA, when complying with the Unit test.

```
[TestClass]
   public class CalculatorTests
   {
      ICalculate calulate;
      [TestInitialize]
      public void TestSet()
      {
         calulate = new Calculate();
      }
      [TestMethod]
      public void TestMethodADD()
      {
         double res = calulate.ADD(1, 1);
         Assert.AreEqual(2, res);
      }
   }
```

Option 4

Step1:
1.Create a dll file using Interface Library - ICalculate.
2.Create a Calculator class that inherits the ICalculate interface.

```
 interface ICalculate
   {
      double ADD(double a, double b);
   }
```

Step2:
1.Create a project from the menu Test / UnitTestProject1. Create a CalculatorTests class.
2.Add the created dll file from the menu References / Add Reference.
3.Add a void TestSet () method.
4.Adding void TestMethodADD () method
5.Compliance with AAA, when complying with the Unit test.

```
[TestClass]
   public class CalculatorTests
   {
      ICalculate calulate;
      [TestInitialize]
      public void TestSet()
      {
         calulate = new Calculate();
      }
      [TestMethod]
      public void TestMethodADD()
      {
         double res = calulate.ADD(1, 1);
         Assert.AreEqual(2, res);
      }
   }
```

Option 5 - Three new methods are added to the specified condition in Test 1: Subtraction, Multiplication and Division.

Step1:
1.Create a Calculator class.
2.Add Sum method to implement mathematical operation Addition.
3.Add Subtraction method to implement mathematical operation Addition.
4.Adding a Divide method to implement a mathematical addition operation.
5.Adding Multiply method for realization of mathematical operation Addition.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace UnitTestProject1
{  public class Calculator
   {    public double Sum(double num1, double num2)
      { return num1 + num2;   }
      public double Subtraction (double num1, double num2)
      { return num1 - num2;   }
      public double Divide(double num1, double num2)
      { return num1 / num2;   }
      public double Multiply(double num1, double num2)
      { return num1 * num2;   }
   }
}
```

Step2:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
namespace UnitTestProject1
{    [TestClass]
     public class CalculatorTests
     { [TestMethod]
       public void Test_SumMethod()
       {
       double num1 = 1; double num2 = 2;
       var calculator = new Calculator();
       double result = calculator.Sum(num1, num2);
Assert.AreEqual(3, result);  }
       [TestMethod]
       public void Test_SubstractMethod()
       {
       double num1 = 1; double num2 = 2;
       var calculator = new Calculator();
       double result = calculator.Substract(num1, num2);
Assert.AreEqual(-1, result);  }
       [TestMethod]
       public void Test_DivideMethod()
       {
       double num1 = 1; double num2 = 2;
       var calculator = new Calculator();
       double result = calculator.Divide(num1, num2);
Assert.AreEqual(0.5, result);  }
       [TestMethod]
       public void Test_MultiplyMethod()
       {
       double num1 = 1; double num2 = 2;
       var calculator = new Calculator();
       double result = calculator.Multiply(num1, num2);
Assert.AreEqual(2, result);  }
      }
}
```

Step3:
1.Visualization of the result of the performed Unit test in the Test Explorer panel.

Step4:
The tested method is presented with a colored icon, test name and execution time. The color of the icon depends on the output of the test. If it is successful, it is green, otherwise it is red. If the color of the test method icon is red, the test method is corrected and executed again.

Step 3 and Step 4 are the same for all Options from 1 to 5 in Test 1

Test 2: Create a Unit test for access a private variable

Coding:

```
namespace TestLib
{
   public class TestClass
   {
      private int _idClass;
      private string _nameClass;
      public TestClass()
      { }
      public TestClass(int IdClass, string NameClass)
      {
         this._idClass = IdClass;
         this._nameClass = NameClass;
      }
   }
}
```

Unit Testing:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using TestLib;
[TestClass]
   public class UnitTest1
   {
      [TestMethod]
      public void TestMethod()
      {
         //Arrange
         int _idClass = 1;
         string _nameClass = "Eva Planck";
         //Act
         TestClass newTestClass = new TestClass(_idClass,
_nameClass);
Microsoft.VisualStudio.TestTools.UnitTesting.PrivateObject
pObject =
         new
Microsoft.VisualStudio.TestTools.UnitTesting.PrivateObject(
newTestClass);
         // Assert
         Assert.AreEqual<int?>(_idClass,
pObject.GetFieldOrProperty("_idClass") as int?);
         Assert.AreEqual<string>(_nameClass,
pObject.GetFieldOrProperty("_nameClass") as string);
      }
   }
```

Test 3: To create a Unit test SCRIPT, for SQL server DataBase University with the following tables:
University (IDUniversity, Name)
Department Department (IDDepartment, Name, IDUniversity)
Specialty Specialty (IDSpeciality, Name, IDDepartment, Degree, Year, Languages, AnnualFee)
1. To enter data in University, Department and Specialty tables
2. Processing AddToDepartment @ IDSpeciality = 01, @ IDDepartment = 001
3. Filtering records in the Department table by criteria IDDepartment = 001

<u>Unit Testing:</u>

```
//Arrange
INSERT INTO dbo.University
(IDUniversity, Name)
VALUES (5, "TU-Varna");
INSERT INTO dbo.Department
(IDDepartment, Name, IDUniversity)
VALUE (51, "SIT", 5);
INSERT INTO dbo.Speciality (IDSpeciality,
Name, IDDepartment, Degree, Year,
TypeOfTraining, Languages, AnnualFee)
VALUES (511, "SIT", 51, bachelor, 4,
regularly, AEO, 3000);

//Act
EXEC dbo.AddSpecialityToDepartment
@IDSpeciality=01, @IDDepartment=001;

// Assert
SELECT * FROM Department WHERE
IDDepartment=001;
```

Test 4: Create a Unit Anti Patterns test – Liar (manning, 2021)

Unit Testing:

```
[Fact]
public void ReturnEmptyForNegativeInputs()
{
   //Arrange
   var ResultExpected = 200;
   var sut = new Calculator();
   //Act
   var result = sut.Sum(100, 100);
   //Assert
   Assert.Equal(ResultExpected, result);
}
```

Test 5: Create a Unit Design Patterns test

Coding:

```
public class Zoo
{
 public void Birds ()
 {
  Helper.Instance().DoIt();
 }
}
public class Zoo
{
 private readonly IHelper _helper;
          public Zoo()
 {
  _helper = Helper.Instance();
 }
 public Zoo(IHelper helper)
 {
  _helper = helper;
 }
 public void Birds()
 {
  _helper.DoIt();
 }
}
```

Unit Testing:

```
using Xunit;
using Moq;
public class ZooTests
{
   [Fact]
   public void Birds_Invokes_Helper()
   {
      var mock = new Mock<IHelper>();
      mock.Setup(x => x.DoIt());
      var zoo = new Zoo(mock.Object);
      zoo.Birds();
      mock.VerifyAll();
   }
}
```

DPs provide a solution to a specific programming problem, in a specific context, that can be used in many other different situations. (Paul, 2012)

As a result of insufficient experience or knowledge in solving a certain type of problems or using a well-established template in the wrong context, opposites of Software Design Patterns (SDPs) arise, etc. Anti-Patterns. Like any other program code, Anti-Patterns are tested. (manning, 2021) Anti-Pattern Unit tests are: Loudmouth, Greedy Catcher, Sequencer, Enumerato, Liar and others. The Liar is a single test that works and does not fail. Unfortunately, he does not test what he claims to test. What is characteristic of it is that its name is misleading because it bears the name of a certain class / method, but in reality it tests another class / method. The actual Liar gives a false sense of security. For example, if you test a method called ReturnEmptyForNegativeInputs designed to test negative values, it tests only positive values and the statement checks the result of the sum and the test is successful, although there is a discrepancy in what the test describes in its name. Ie the test is correct, although it tries to prove a completely different statement. There are two ways to correct this Unit test of the Lear Anti-Patterns: Updating the test name to a name that corresponds to the performance; Changing the performance of the test to match the name of the test. The conclusion is that the Liar is one of the most harmful TDD Anti-Patterns. It gives a false sense of security because it lies behind the test. Therefore, it is difficult to find the error in the code itself. To avoid this problem when creating / updating modular tests, you should always check that the test performance matches its name.

7. CONCLUSION

The article discussed the main problems in software development and pointed out the need for training in the field of software testing. Chapter 2 described ST as a phase of the software life cycle that is present in every software model. Two classifications were presented: of software models and of ST according to the literature. Chapters three and four described the advantages of automatic over manual testing and the possibilities for a successful ST, as well as automated testing tools. Chapter 5 discussed the concepts, benefits, and characteristics of ST. And in the last sixth experimental chapter were included 5 single tests, focused on one of the most common problems in software development, namely: mathematical methods; access to a private variable; to work with databases, in particular with SQL server; Design Patterns and Anti-Patterns. Test №1 was performed in 5 variants, each of which is described in 4 main steps: Code generation (interface / class / method) for testing; Create a Unit test of the generated code from Step 1; Execution and visualization of the result of the performed Unit test; Analysis and conclusions from the Unit test. Tests №2 and №4 are presented only with Unit test. Tests №3 and №5 are performed using a master code and a Unit test.

In conclusion, it can be said that by mastering the theory and realizing the practical tasks, students increase their competence in the field of software testing, and in particular to one of the most common types of testing, namely Unit test.

ACKNOWLEDGMENT

This paper is supported by the National Scientific Program "Information and Communication Technologies for a Single Digital Market in Science, Education and Security (ICTinSES)" (grant agreement DO1-205/23.11.18), financed by the Ministry of Education and Science.

REFERENCES

Arsie, O. (2019). Organizational Best Practices for Software Testing Success

Dustin, E. (2002). Effective software testing: 50 specific ways to improve your testing

Dustin, E. (2014). Creating an Automated Software Testing Center of Excellence

Dustin, E., Garrett, T., & Gauf, B. (2009). Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality

Dustin, E., Rashka, J., & McDiarmid, D. (2001).Quality Web Systems: Performance, Security, and Usability

Functionize.(July 17, 2018).Types of Software Testing. https://www.functionize.com/blog/types-of-software-testing/

Georgi, Ch., May, St. (2020). http://edesign-bg.com/courses/software-quality-2019-20/QA-upr-za-03-2019-2020.pdf

Guru. (2021). https://www.guru99.com/software-testing-service-providers.html

Helfen, M., Lauer, M., & Trauthwein, H.M. (2007). Testing SAP Solutions

IEEE. (2013). https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29119:-1:ed-1:v1:en

Jacob, P., & Prasanna, M. (2016).A Comparative analysis on Black box testing strategies. 2016 International Conference on Information Science (ICIS), 1-6

Kalin, V. (Jul. 03, 2010). https://www.slideshare.net/kalin4y/ss-4672183

Kiran, Th. (Aug. 07, 2016). https://www.slideshare.net/kiranthejа1/testing-ppt-64769727

Khorikov, Vl. (2020). Unit Testing Principles, Practices, and Patterns

Maneva, N., Eskenazi, Avr. (2001). Software technologies manning. (2021). https://livebook.manning.com/book/unit-testing/chapter-3/

Myservername. (2021). https://bg.myservername.com/top-20-best-automation-testing-tools-2021#Top_20_BEST_Automation_Testing_Tools_Compared

Nelson, L., Wysopal, C., & Dustin, E. (2006).The Art of Software Security Testing: Identifying Software Security Flaws

Noorian, M., Bagheri, E., & Du, W. (2011). Machine Learning-based Software Testing: Towards a Classification Framework. SEKE

Paul, J. (2012).Design Patterns in C#

Phillip L., Fevzi B., Jerry G., Greg K., Keith M., W. Eric W., & Dianxiang X.(2010).Software Test Automation

Software development academy. (19.05.2020). What is the difference between manual and automated testing? https://sdacademy.dev/what-is-the-difference-between-manual-testing-and-automated-testing/

Sommerville, J. (2011). Software Engineering.

tutorialspoin. (2021). https://www.tutorialspoint.com/software_testing/software_testing_levels.htm