AN APPROACH TO THE DESIGN SOFTWARE AUTOMATION TESTING ENVIROMENT

Srđan Nogo, Zoran Škrkar

Faculty of Electrical Engineering in East Sarajevo
e-mails: srdjan.nogo@gmail.com; zoran.skrkar@gmail.com
Bosnia and Herzegovina

Abstract: This paper investigates technical possibilities for development of an environment for practical application of automated software testing. Such an environment, combined with today software tools, enables software automated test run with no human intervention. We will investigate design software automated testing environment upon which we will create and perform several methods of automated software testing. The theoretical model will be applied to a practical model for implementation automated test environment with usage programming language C# combined with Selenium web driver for C#.

Key words: Selenium web driver, Automation software testing, C#.

1. INTRODUCTION

Question ”What is software testing?“ brings a whole list of answers. One option is to say that software testing is looking for errors in the software and it showed up along with appearance of very first practical application software. We can also say that systematical approach to testing is a way to find possible drawbacks of software and therefore improve its quality. According to Glenford et al. in [1], “Software testing is a process, or a serius of processes, designed to make sure computer code does what it was designated to do and, conversely, that it does not do anything unintended”. From this statement we can conclude that the main objective of testing is to find bugs in computer code and to fix them which improve quality of software.

Nowadays this testing process has improved its importance and requires very serious and systematical approach in order to find drawbacks within the program.

For example, Srinivas and Jagruthi in [2], give an assessment that the process of testing consumes 40-50 % of development cycle time and also consumes more effort for softwares which require more reliability. From this statement we can see that a significant number of quality assurance team working hours is allocated for program code testing of final and initial versions of program solutions.

Automated testing has significant importance in case of expensive and robust program systems that are in an initial stage of production and in which a large amount of money has been invested, as well as a large number of engineer working hours. Generally, program testing improves its stability, end user satisfacion and decrease of expenses, but the main reason is that it is more optimal to eliminate an error if detected earlier (earlier, better). Program code for automated testing has become inevitable part of software maintenance process, especially because todays systems are mostly web and cloud system oriented and therefore possible errors could not only cause material damage in real time but also endanger end user data security.

This paper is structured as follows: After Introduction, where we answer the question „What is software testing?“, comes Section II which describes methodology for program code automated testing and where we use programming language C# combined with Selenium web driver for C#. Section Metodology which was developed, as the usage is concerned, is a starting basis for the proposed framework for automated testing outlined in Section III. Section III, presents a good practice case for creation and utilization of various methods for automated smoke testing with generation of test results, which can be sent to responsible persons via e-mail. Section IV, contains the main focus of this survey paper with technical details required for implementation of conceptual framework for Automation Testing, which we can compare with similar framework solutions from this area. Practical application of automated testing described in Section IV provide to us a visual means to confirm our summary and conclusions outlined in Section V.

2. METHODOLOGY

During series of program code testing, which require repetitive and redundant tasks along with ommitance of human and manual involvement, the optimal way is to use automation testing tool. According to Ieshin et al. in [3], use of automation test tool for program code testing increases the test execution speed and software become more reliable, repeatable, programmable, comprehensive, and reusable. Automation testing covers all the problems of manual testing and reveal all complex Obstacles attached with it.

At the beginning of the testing, prior to choosing automated software testing tool, it is necessary to clarify list of requirements for program code testing. Kaur and Gupta in [4], state that ”If we do not have a list of requirements, we may waste time for downloading, installing and evaluating tools that only meet some of requirements, or may not meet any of them”.

List of requirements is required particularly by those system parts which have a large number of elements and can be placed in cathegory of complex parts of the program code. If we look through the eyes of test engineer, it is optimal to create scripts that will test part of the system and click on 250 buttons at once. During this action automated testing of each separate element for availability and optimal functioning is done, instead of manual approach to each element individually through 250 repeated steps by test engineer.

Kankanamge in [5], claims that “Test automation frees up the tester's time to do more effective and exploratory tests which are crucial for the success of quality assurance”. Providing of such a quality assurance optimal level from the user's point of view is very important, as it is guaranteed that he can run the application in any available web browser, using any of its versions. Developer team in the development process, as well as test engineers in production environment, spend most of the time removing bugs concerning tune up the application to each web browser, particularly to their older versions and which can be used by the user. In the previous section of this paragraph, we have listed some preliminary research in this field that concerns automated software tests. Based on this, in the next sections we will design software automated testing environment in combination with web tools that enable the setting of automatic tests without limiting parameters such as time and human factor. Presentation of this method of automated testing environment without limiting parameters represents the achievement of the main goal of the automation process in the field of software testing

As a tool for automated program code testing in this paper we will use programming language C# combined with web driver for C#. In that programming language we will create and perform automated smoke testing with generation of test results, which can be sent to responsible person via e-mail. The test itself, which can collect the data, can be implemented in the manner of storing the data in the database, which is the best option, or by writing them locally to a file.

3. FRAMEWORK FOR AUTOMATED TESTING

Afroz et al. in [6], state that, “It is important that Web applications be dependable, but recent reports indicate that in practice they often are not“. For example, one study of Web application integrity found that 29 of 40 leading e-commerce sites [7], and 28 of 41 government sites [8], exhibited some type of failure when exercised by a “first-time user”. Based on the results of this research, in order to avoid failure, it is necessary to decide for the optimal type of test, or to select the framework for automatic testing.

When we get the first version of the application, by the developer team, testing the program code is done using a smoke test. As we see in Figure 1, a flow diagram of smoke testing is shown.

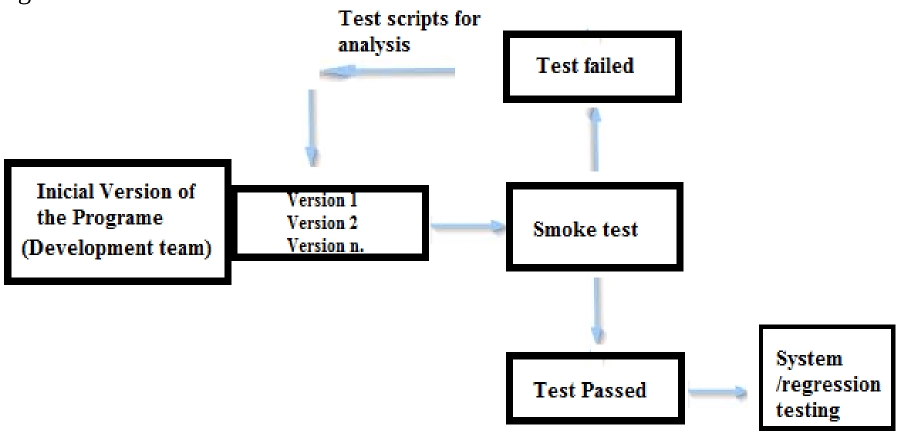

Test scripts for analysis
Test failed
Inicial Version of the Programe (Development team)
Version 1
Version 2
Version n.
Smoke test
Test Passed
System /regression testing

Fig. 1. Smoke test diagram

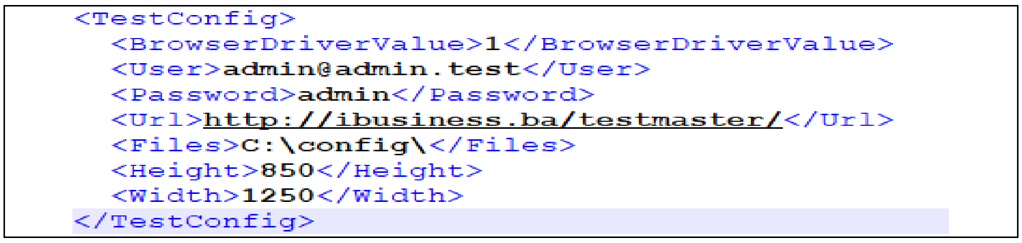
In the initial versions of the program whose version is still unstable for production use in full capacity, we use a smoke test. Smoke test is a type of test that is performed before all other types of tests, which can be automated as needed, in order to eliminate all possible shortcomings of the program code of the application in a systematic and automated manner. Once a new version of the application is released, a smoke test is performed to determine whether the system is available at all, i.e. whether it gives the expected response. If the smoke test passes anticipated, then it is only allowed to proceed to a further testing process, i.e. systemic / regression testing. System / regression testing includes tests that are performed to determine that no errors or defects have occurred after programmers install a new version of the code on the system or part of the system that has already been completed and shipped to the client. Also, this testing is done to verify that the old defects have not returned with the installation of a new code. If the smoke test does not pass, it is considered that the system is blocked and out of use. The result of an unsuccessful test is returned in the form of test scripts for analysis by the developer team. After the analysis, a redesign of the program code is made by the developer team in order to bring the application into the state to pass the smoke test. In a framework for the practical application of the smoke of automatic testing in a specific production system outlined in Section IV, checks are carried out on whether the basic parts of the system work properly, after installing a new version of the program. Our goal is to check the stability of the system. Smoke test is a test that is a kind of control for the entire system.

Kaur and Gupta in [4], state that, "The selection of particular automated testing tool is based on the type of application we are testing and the cost associated with the tool". The application we test for a smoke test is available at http://ibusiness.ba/testmaster and is designed to have about 2 million users' unique visits annually.

Given the type of application as well as the trends in automated testing of web applications, as a tool for automated testing (written in open source technology), it was necessary for us to choose one such tool. In the presented practical work, the tests were created with the help of a Selenium web driver for C#. Because of its ease of use, this test tool can be used by a developer, who is part of a development team, or by person who will testing aplications.

4. PRACTICAL APPLICATION OF AUTOMATIC TESTING IN THE CONCRETE PRODUCTION SYSTEM

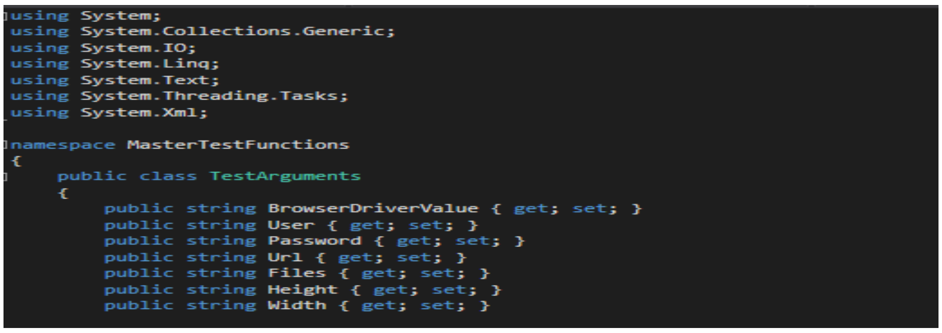
By the term, "creation of automated tests", we mean activities that are specific programming of the code with certain test functions as well as their calling at a particular moment. In this particular example, we automatically test the application, using a smoke test to determine whether the system is available at all, i.e. it gives the expected response to the user. In order to avoid as much as possible the so-called hard coding, in which, when certain parameters or values are repeated from the method to method, the file in which the parameters are configured is used. Code in XML format of configuration file used in the test is shown in Figure 2.



```
<TestConfig>
  <BrowserDriverValue>1</BrowserDriverValue>
  <User>admin@admin.test</User>
  <Password>admin</Password>
  <Url>http://ibusiness.ba/testmaster/</Url>
  <Files>C:\config\</Files>
  <Height>850</Height>
  <Width>1250</Width>
</TestConfig>
```

Fig. 2. Code of used configuration file

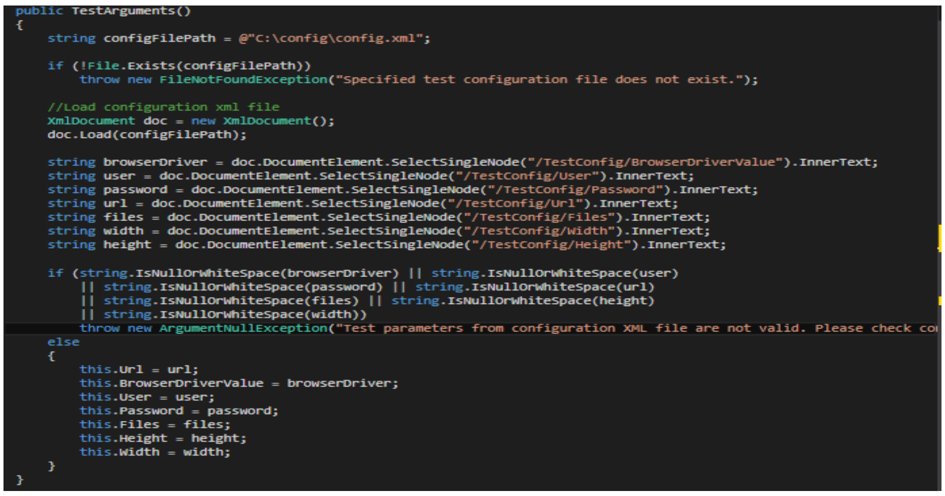
A complete description of computing Extensible Markup Language (XML) as a markup language can be found in [9]. After we have created a file with configuration parameters, it is necessary to create a class in the code, which will be used to call the parameters as needed. We named the class, "TestArguments" as we see in Figure 3, and keyword after the access modifier public is called the return type Class TestArguments. It is written to return the parameter for, BrowserDriverValue, User, Password, Url, Files, Height and Width.



```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;

namespace MasterTestFunctions
{
    public class TestArguments
    {
        public string BrowserDriverValue { get; set; }
        public string User { get; set; }
        public string Password { get; set; }
        public string Url { get; set; }
        public string Files { get; set; }
        public string Height { get; set; }
        public string Width { get; set; }
```

Fig. 3. Define attributes in configuration file

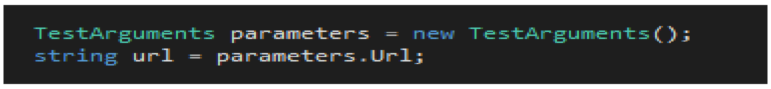
Then, the config file is read from code and the parameter values are read one at a time, and these values are assigned to local variables. Class that passes values from the config file to local variables is given in Figure 4.



```
public TestArguments()
{
    string configFilePath = @"C:\config\config.xml";

    if (!File.Exists(configFilePath))
        throw new FileNotFoundException("Specified test configuration file does not exist.");

    //Load configuration xml file
    XmlDocument doc = new XmlDocument();
    doc.Load(configFilePath);

    string browserDriver = doc.DocumentElement.SelectSingleNode("/TestConfig/BrowserDriverValue").InnerText;
    string user = doc.DocumentElement.SelectSingleNode("/TestConfig/User").InnerText;
    string password = doc.DocumentElement.SelectSingleNode("/TestConfig/Password").InnerText;
    string url = doc.DocumentElement.SelectSingleNode("/TestConfig/Url").InnerText;
    string files = doc.DocumentElement.SelectSingleNode("/TestConfig/Files").InnerText;
    string width = doc.DocumentElement.SelectSingleNode("/TestConfig/Width").InnerText;
    string height = doc.DocumentElement.SelectSingleNode("/TestConfig/Height").InnerText;

    if (string.IsNullOrWhiteSpace(browserDriver) || string.IsNullOrWhiteSpace(user)
        || string.IsNullOrWhiteSpace(password) || string.IsNullOrWhiteSpace(url)
        || string.IsNullOrWhiteSpace(files) || string.IsNullOrWhiteSpace(height)
        || string.IsNullOrWhiteSpace(width))
        throw new ArgumentNullException("Test parameters from configuration XML file are not valid. Please check co
    else
    {
        this.Url = url;
        this.BrowserDriverValue = browserDriver;
        this.User = user;
        this.Password = password;
        this.Files = files;
        this.Height = height;
        this.Width = width;
    }
}
```

Fig. 4. Appearance of a class that passes values from the config file to local variables

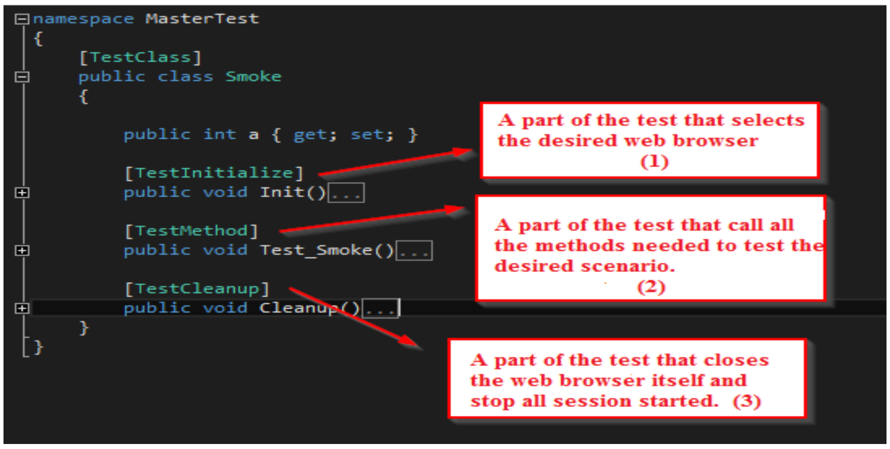
Defined variables on this way, can be invoked later anywhere from the code, wherever there is a need for one of the values from the file for configuring the parameters and it is given in Figure 5.



```
TestArguments parameters = new TestArguments();
string url = parameters.Url;
```

Fig. 5. Calling url from config file.

Defining the parameters on this way and calling them in the manner shown in Figure 5. is an approach called the parameterization of the constants, which avoids the access to direct code input by the programmer. As we described in chapter 2, *(Methodology)* after the basic step when we define a list of test requests, next step, which must be defined in the testing process, which is to accept the methodology of the test itself when creating the tests. For the purposes of concrete testing, we will assume that each test consists of three parts, Figure 6.



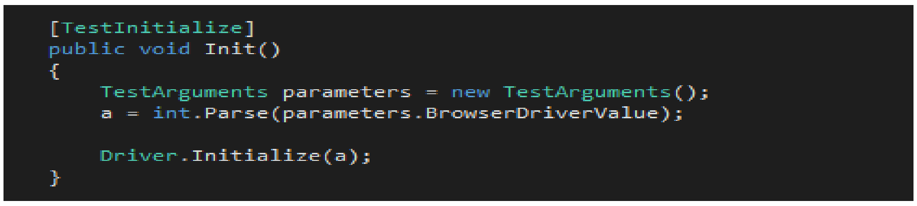
```
namespace MasterTest
{
    [TestClass]
    public class Smoke
    {
        public int a { get; set; }

        [TestInitialize]
        public void Init()...

        [TestMethod]
        public void Test_Smoke()...

        [TestCleanup]
        public void Cleanup()...
    }
}
```

A part of the test that selects the desired web browser (1)
A part of the test that call all the methods needed to test the desired scenario. (2)
A part of the test that closes the web browser itself and stop all session started. (3)

Fig. 6. Test code structure

In the first part for initialization, it’s caling diferent types of web browsers such as Internet Expoler, Google Chrome, Mozilla Firefox, etc. (1). The second part of the test consists of the methods that are required to test the desired scenario and are referred to in this section (2). The third part consists of the functionality in which the web browser itself closes and interrupts the running browser session (3). In each section, the use of the parameters from the config file is expressed in the manner as previously explained and shown in Figure 2. Each of these parts of the code shown in Figure 6 has its purpose and the exact methods that are called in that part.

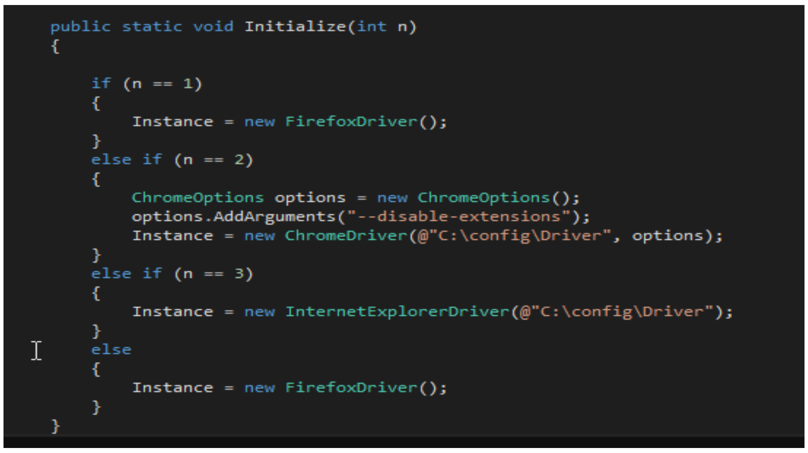
Based on the above, we begin the Test by executing the code where the browser is called, by transmitting the previously created method to the parameter whose value depends on which type of web browser as shown in Figure 7.



```
[TestInitialize]
public void Init()
{
    TestArguments parameters = new TestArguments();
    a = int.Parse(parameters.BrowserDriverValue);

    Driver.Initialize(a);
}
```

Fig. 7. Code for calling web browser

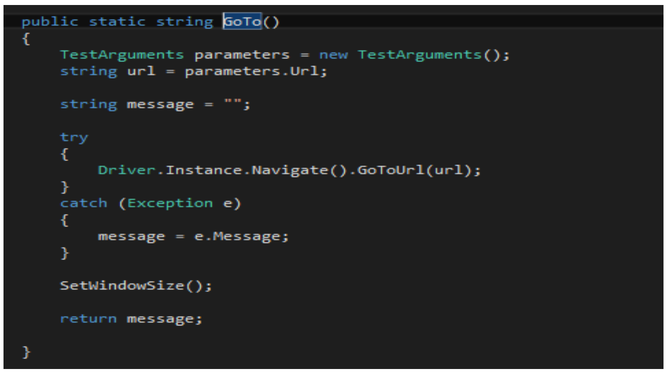
If the file for configuring the parameters in the "BrowserDriverValue" node is entered the number 1 Mozila Firerox will be called, if the number 2 is entered it’s a number for Google Chrome, and if the number 3 is entered, the Internet Explorer call will be made as shown in Figure 8.



```
public static void Initialize(int n)
{

    if (n == 1)
    {
        Instance = new FirefoxDriver();
    }
    else if (n == 2)
    {
        ChromeOptions options = new ChromeOptions();
        options.AddArguments("--disable-extensions");
        Instance = new ChromeDriver(@"C:\config\Driver", options);
    }
    else if (n == 3)
    {
        Instance = new InternetExplorerDriver(@"C:\config\Driver");
    }
    else
    {
        Instance = new FirefoxDriver();
    }
}
```

Fig. 8. Calling the appropriate browser based on the parameter passed

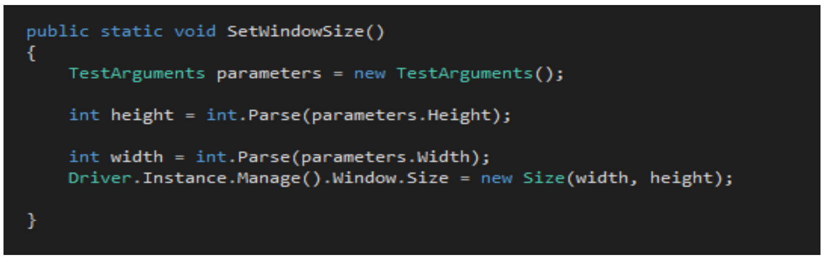
After opening the selected browser based on our selection, the next step is to open the Uniform Resource Locator (URL) that wants to be tested. In order to do this, we need to have certain user authorizations and connect to the URL as well as to define all the methods that are required to be performed in our test script. Also for this purpose we created a GoTo Statement that transfers the program control directly to a labeled statement in the form of the parameter which calling location parameter defined in the config file as shown in Fig. 9.



```
public static string GoTo()
{
    TestArguments parameters = new TestArguments();
    string url = parameters.Url;

    string message = "";

    try
    {
        Driver.Instance.Navigate().GoToUrl(url);
    }
    catch (Exception e)
    {
        message = e.Message;
    }

    SetWindowSize();

    return message;

}
```

Fig. 9. Usage GoTo statement in test

In the second part of the test (2), which calls a particular link, there is a parameter, which sets the size of the browser window to a fixed and always the same value. We need this because it is necessary, among other things, to test the position of the elements in the table relative to the edge of the web browser window. If there was a change in position, the test would "spot it", and by introducing the method for defining the window size, we enabled the testing of the position of the elements on different screens of different sizes, because the size of the browser window is always the same as shown in Figure 10.

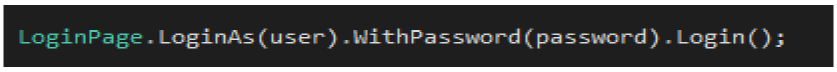


```
public static void SetWindowSize()
{
    TestArguments parameters = new TestArguments();

    int height = int.Parse(parameters.Height);

    int width = int.Parse(parameters.Width);
    Driver.Instance.Manage().Window.Size = new Size(width, height);

}
```

Fig. 10. Method that sets the size of the browser window

As you can see the methods used, values have been invoked in an already defined way from the parameter configuration file. In this way, we have, among other things, made it possible to define all the necessary parameters in one place only, and if there is an error in the input, the engineer tester can find the error in as short a time interval as possible.

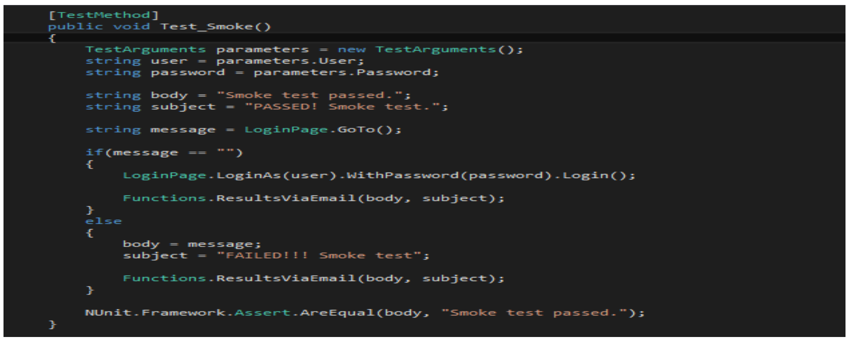
After we have opened the desired location, the next step is to log on to the system using the passed authority, we do this in the following way using the code line as shown in Figure 11.



```
LoginPage.LoginAs(user).WithPassword(password).Login();
```

Fig. 11. Login methods.

We also take user and password as parameters from the parameter configuration file. In the process of creating a fremwork environment for software testing, we notice common steps for all types of automated testing. These steps are as we presented the methods above opening the browser, calling the site and logging onto the system as the minimum of every possible defect in the software. As is shown in Figure 12.

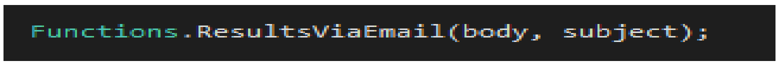


```
[TestMethod]
public void Test_Smoke()
{
    TestArguments parameters = new TestArguments();
    string user = parameters.User;
    string password = parameters.Password;

    string body = "Smoke test passed.";
    string subject = "PASSED! Smoke test.";

    string message = LoginPage.GoTo();

    if(message == "")
    {
        LoginPage.LoginAs(user).WithPassword(password).Login();

        Functions.ResultsViaEmail(body, subject);
    }
    else
    {
        body = message;
        subject = "FAILED!!! Smoke test";

        Functions.ResultsViaEmail(body, subject);
    }

    NUnit.Framework.Assert.AreEqual(body, "Smoke test passed.");
}
```

Fig. 12. Smoke test method

There is a method showing a smoke test whose primary goal is to log on to the system. If logging on the system passes the message "Smoke test passed" will be generated successfully, and if the logging does not pass the answer is negative.

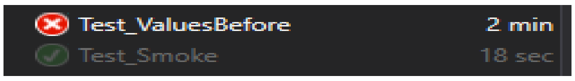
The last third step (3) of each test is to close the web browser and to end the browser session. After this action, it is crucial to send a message with the test results, as shown in the smoke test code in Figure 12. Using the ResultsViaEmail method where the results arrive at the pre-defined e-mail as shown in Figure 13.



```
Functions.ResultsViaEmail(body, subject);
```

Fig. 13. The method used to send test results to email.

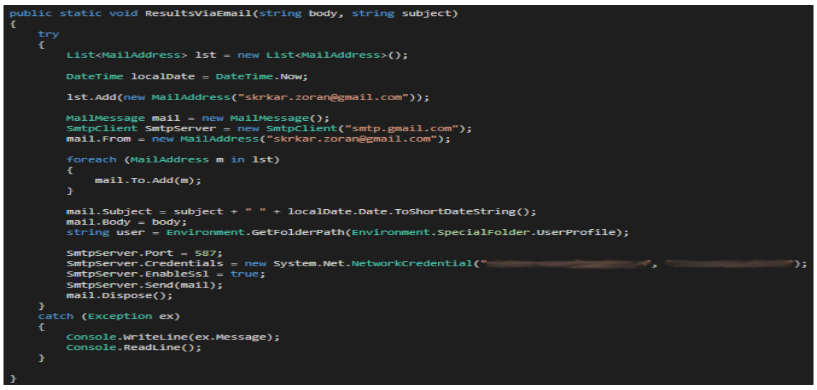
The method uses the body and subject parameters that are tested in the test depending on whether the test has passed or failed to configure and forward this function. An integral part of each test is also a step in which to compare the "correct score" with the test results, and if the test passes the test will have a positive result in the environment in which it is run, i.e. The result of the test will be colored green as shown in Figure 14. If the test fails, the result will be shaded in red. This part of the test is an additional method of displaying test results in addition to the mentioned ability to send an e-mail message.



```
Test_ValuesBefore    2 min
Test_Smoke           18 sec
```

Fig. 14. Test results

The method for sending the test results using the e-mail message is shown in Fig.15.



```
public static void ResultsViaEmail(string body, string subject)
{
    try
    {
        List<MailAddress> lst = new List<MailAddress>();

        DateTime localDate = DateTime.Now;

        lst.Add(new MailAddress("skrkar.zoran@gmail.com"));

        MailMessage mail = new MailMessage();
        SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
        mail.From = new MailAddress("skrkar.zoran@gmail.com");

        foreach (MailAddress m in lst)
        {
            mail.To.Add(m);
        }

        mail.Subject = subject + " " + localDate.Date.ToShortDateString();
        mail.Body = body;
        string user = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);

        SmtpServer.Port = 587;
        SmtpServer.Credentials = new System.Net.NetworkCredential("[redacted]", "[redacted]");
        SmtpServer.EnableSsl = true;
        SmtpServer.Send(mail);
        mail.Dispose();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }

}
```

Fig.15. Method for sending email

Based on the above-mentioned methods and results of concrete smoke testing with the help of which we automatically test the application, we have determined that the system is available, i.e. that a response is given to the user. The obtained test results are displayed on the display tester of the engineer or sent to e-mail in real time. The obtained results can be entered into the database in order to monitor the history of the tests carried out and the eventual application shortcomings after the software automated testing process.

5. CONCLUSION

This paper outlines a framework for automated testing envirement as a basis for this kind of solution. Based on this framework, we have established an appropriate automated testing environment for application testing, using a smoke test that has determined that the system is available for users with access privileges.

We have proposed a solution that seems to be the most optimal from the standpoint of ease of use. This Smoke Testing Framework can be applied to most other types of automated tests because they are based on the same principle using software tools such as Selenium and can be used by a programmer who is part of a development team or a person who is part of testing team.

A model for practical automated testing, outlined in Chapter IV, meets the framework requirements for future upgrading of the automated software testing method. Finally, this research work can be extended with additional modules and software tools, which allow you to set to automatically run tests without the presence of a human being, which is the achievement of the goal of the automation process.

REFERENCES

[1] Glenford J. Myers, Corey Sandler, Tom Badgett, *The Art of Software Testing*, 3rd edition, ISBN: 978-1-118-03196-4, Nov 2011.

[2] Srinivas Nidhra and Jagruthi Dondeti, "Black box and White box techniques-A Literature review.", *International Journal of Embedded Systems and Applications, Vol 2*, No. 2, June 2012 (https://www.ijraset.com/fileserve.php?FID=4529).

[3] A. Ieshin, M. Gerenko, and V. Dmitriev, *Test Automation- Flexible Way*, IEEE, 978-1-4244-5665-9, 2009 (https://ieeexplore.ieee.org/document/5501151/).

[4] Harpreet Kaur , Dr.Gagan Gupta, Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete, *Int. Journal of Engineering Research and Applications*, ISSN : 2248-9622, Vol. 3, Issue 5, Sep-Oct 2013, pp.1739-1743

[5] Kankanamge, C. (2012). Web Services Testing with SoapUI. Birmingham: *Packt Publishing.APA* (American Psychological Assoc.)

[6] Dr. S. M. Afroz, N. Elezabeth Rani and N. Indira Priyadarshini, Web Application– A Study on Comparing Software Testing Tools, *International Journal of Computer Science and Telecommunications*, 2011 (http://www.ijcst.org/Volume2/Issue3/p1_2_3.pdf)

[7] Business Internet Group of San Francisco, *The BIG-SF Report on Government Web Application, Integrity* (http://www.tealeaf.com/downloads/news/analyst_report/BIG-SF_Report_Gov2003-05).

[8] Business Internet Group of San Francisco, *The Black Friday Report on Web Application Integrity* (http://www.tealeaf.com/ downloads/news/analyst_report/BIG).

[9] Bray T, Paoli J, Sperberg-McQueen CM, Maler E, Yergeau F, Cowan J. *Extensible Markup Language (XML) 1.1* (2nd Edition); 2006 (http://www.w3.org/TR/xml11/).

Information about the authors:

Prof. **Srđan Nogo** has been a full professor in computer science at Department for computer sciences and information technology, Faculty of Electrical Engineering at University of East Sarajevo since 2013. He received his Ph.D. (Tech.) degree in 2013 at Faculty of Electrical Engineering and his M.Sc. (Tech.) degree in 2007 at same faculty. His primary research interests include analysis of standards, guidelines and best practices in the area of software development, data exchange, web services and database solutions.

Mr. Zoran Škrkar is currently a M.Sc. student at the Faculty of Electrical Engineering at University of East Sarajevo. He received his Bachelor degree in Computer Science in 2010. Faculty of Electrical Engineering His primary research interests include Computer Security, Quality Analysis, Quality Control, and solutions for Automatic software testing.

Manuscript received on 14 April 2018