Automated Code Testing System for Bug Prevention in Web-based User Interfaces

Dragoș SMADA, Carmen ROTUNĂ, Radu BONCEA, Ionuț PETRE
The National Institute for Research & Development in Informatics
dragos.smada@ici.ro, carmen.rotuna@rotld.ro, radu@rotld.ro, ionut.petre@ici.ro

Automation in testing user interfaces is a prerequisite for overcoming the major weaknesses of manual testing, such as time consumption, not being able to reproduce the sequence that generates a bug or the tendency to repeat only the successful steps. Continuous testing represents an important step in the agile software development cycle because any features and changes added to the code need to be checked before their propagation to production environment. Manually testing is a resource and time-consuming process thus the solution would be to make the entire workflow from committing a change to publishing a new release completely automated. The solution proposed within this paper is a framework for automated code testing and bug prevention that relies on Selenium, a framework supporting also headless testing, integrated with a Continuous Integration (CI) server such as Jenkins.
Keywords: *UI Testing, Web-Based Interface, Bug Prevention, Automated Testing, Bad Code Denial, Agile, Continuous Integration, Headless Automated Testing*

1 Introduction

At present, easy access to information and communication technologies represents one of the premises of good functioning in modern society [7]. Software producers are frequently working improving the applications in the attempt to keep up with the pace imposed by the modern society needs.

In recent years software development shifted from the traditional style towards Agile development mainly caused by the need to accelerate the launch of software applications on the market.

Traditional development style implies an accurate but time costly planning, development and major releases in terms of software products. With Agile development, the software is produced in short cycles, and frequent releases are preferred. No matter of the chosen scenario, tests are required for ensuring a reliable release of the software that meets all the envisioned business and technical requirements.

Validation is defined as: "Confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled [11].

Testing for Validation should confirm that the software contains the feature set and operates according to the requirements established before development began. In practice, the actual cost of software testing is determined by how much it costs to reduce uncertainty of the software quality to the appropriate amount for that application [4].

Manual software testing obviously requires human resources, interface analysis and evaluation. A thorough manual testing is usually performed in long periods, but due to the increased pressure from the management, the testers are forced to release the applications more quickly, a fact that often affects the quality of the application. Therefore, the producers turned to solutions for performing automated testing, which can be viewed as the automated version of manual testing.

In practice there is still a lack of knowledge in the subject of automated testing efforts and pay-off. In a survey of over 700 test professionals, 70-percent of respondents stated they believe that automation for software testing is a high payoff endeavor; however, they were not sure why that was or how automation fit with their project [2]. This shows there is an initial optimism approaching automated testing but a lack of deeper understanding in order to proceed in a certain business case.

DOI: 10.12948/issn14531305/22.3.2018.03

2 Continuous Software Delivery

Continuous software delivery is a software engineering approach in which developer teams produce software in short cycles, ensuring that an application can be released safely at any time. This approach describes different aspects of iterative software applications development such as continuous integration, continuous delivery, continuous testing and continuous deployment [3]. It must be noted that the concept of *continuous delivery* is not similar with *continuous deployment*, concept which implies that the updates are automatically deployed to the production environment. In continuous delivery the team takes the necessary measures to ensure the updates can be deployed to production but may choose not to do it, usually due to business reasons. To implement and work in continuous deployment, one must be doing continuous delivery.

Continuous integration refers to the process of permanently adding new commits to source code. Each team member submits work as soon as it's finished and in this way each developer knows immediately if their code will meet minimum standards and they can immediately fix bugs.

Continuous delivery is based on continuous integration and each commit is automatically tested at the time it is pushed. In addition to the automation component and integration testing, a continuous delivery system will include functional tests, regression tests, and possibly other tests, such as pre-generated acceptance tests. After passing the automated tests, the code changes are sent to a standby environment.

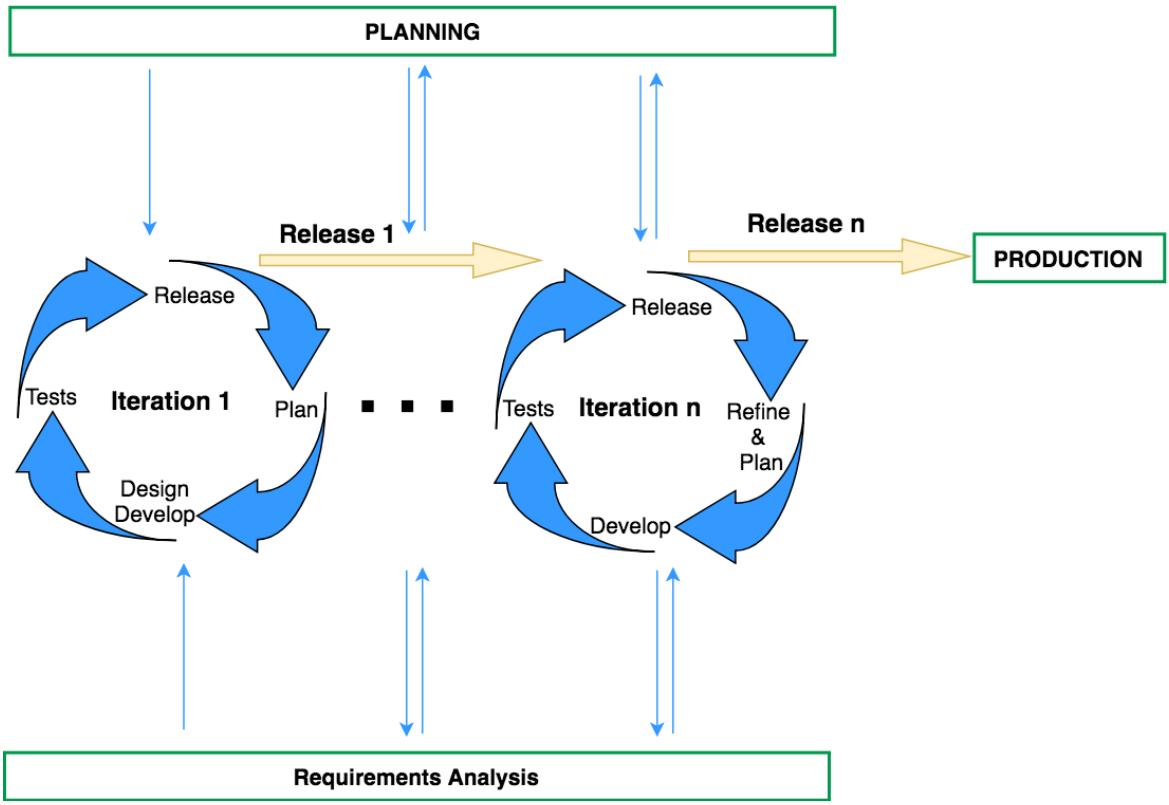

Fig. 1. Test Driven Development

Continuous deployment adds more automation to the software development process. After passing all the automated delivery tests, each commit is deployed into production as soon as it is available [8].

2.1 Agile development

Agile emerged in the 1990s from different lightweight software approaches as a response to some project managers' dislike of

DOI: 10.12948/issn14531305/22.3.2018.03

the rigid, linear Waterfall methodology. It focuses on flexibility, continuous improvement, and speed [15].
Through this approach, software is developed in short cycles, thereby ensuring reliability for timely releases. This results in building, testing and releasing the software faster and more frequently. The approach has proven to reduce cost, time and the risk of delivering critical changes to production, thereby allowing incremental updates to the production system [5].
Agile is an umbrella concept that includes other methodologies such as Scrum, Extreme Programming, Kanban, Crystal etc.
The main phases in the Agile development cycle are *Planning, Performing requirements analysis, Product Design, Development* and *Testing.* The phases are not consecutive, they are flexible and can be done in parallel as the design and requirements often change during product development and testing.
In Agile there is continuous feedback and frequent face-to-face interactions, the project team and stakeholders understand and prioritize the right requirements. Agile teams use user-story backlogs to manage the requirements. Before starting an iteration, the team agrees with the requirements they should meet for the next delivery. This collaborative approach ensures that the most important features are prioritized. Requirements are continually updated throughout the project as new information is presented.

2.2 Kanban

It is a visual frame used to implement Agile that shows what it should produce, when to produce it and how much it produces. This encourages small incremental changes to an existing system and does not require a specific configuration or procedure. Kanban board is used during development - which is a tool for implementing the Kanban project method. Traditionally, this tool was a physical plate, with magnets, plastic chips, or notes on a white board to represent work items, but now more and more project management software tools have created Kanban online panels.

2.3 SCRUM

Scrum is an agile methodology for managing and planning software projects. A framework within which people can address and solve complex and adaptive problems [1]. The Scrum team consists of a Product Owner, the Development Team, and a Scrum Master. Scrum Teams are self - organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team [6]. The Development Team usually consists in few members, yet not smaller than three people.
The functionalities, bug-fixes and improvements are defined and tracked in Product Backlog. The development process occurs iteratively, each iteration is called Sprint and has 2 to 4 weeks. At the beginning of each Sprint, the team holds a meeting where the items in the Backlog are organized and tasks are allocated to developers. Usually team members are requesting tasks by themselves, based on their project experience and programming knowledge. During the Sprint the team meets for briefing sessions and tasks can be re-allocated to ensure that Sprint can end successfully.
At the end of the Sprint the team holds another meeting, the review is performed on each assignment and any unfinished tasks are moved in the next sprint.

2.4 eXtreme Programming - XP

eXtreme Programming is a type of software development designed to improve the quality and ability to respond to changing customer requirements. There are systems whose functionality is expected to change every few months but in many software environments dynamically changing requirements is the only constant. In an XP team the developers, the managers and customers as well, work all together asking questions, negotiating scope and schedules, and creating functional tests. [http://www.extremeprogramming.org/]
The XP principles include feedback, assuming simplicity and adopting change. XP iterations last one or two weeks long compared to Scrum teams which work in iterations lasting 2 to 4

DOI: 10.12948/issn14531305/22.3.2018.03

weeks. The XP teams are open to changing the content of their iteration if the work hasn't started yet on a particular feature, thus a new feature prioritized by the customer can be added to the existing sprint and the team will start working on it. XP recommends engineering practices, specifically techniques like test-driven development, the focus on automated testing, pair programming, simple design, refactoring, continuous integration and so on.

3 Automation of Software Testing

In the Continuous software development cycle, testing is a prerequisite before propagating changes in the production environment. Automation is required for overcoming the major weaknesses of manual testing, such as time consumption, not being able to reproduce the sequence that generates an error, low coverage caused by the tendency to repeat actions, etc. Automation process relies on strategies, tools and artefacts that augment or reduce the need for manual or human involvement or interaction in unskilled, repetitive or redundant tasks [12]. The process of automating the software testing is similar to a software development process. A big difference consists in the test assertion document which must be created before starting the development. When it comes to a software there are several types of tests that can be automated [12]:

- *Functional tests* – checking the operations behavior
- *Regression tests* – checking the system behavior
- *Stress tests* – simulating maximum loads to determine the capability
- *Performance tests* – check if the system is adequate and meets the expectations
- *Loading tests* - determining the points at which the capacity and performance of the system become degraded to the situation that hardware or software upgrades would be required

In the automation process, one of the goals is to run tests without user assistance.

Continuous testing does not eliminate manual testing from the continuous delivery model. Using continuous testing, the team will constantly test the up-to-date version of the code available. Continuous testing still involves manual exploration tests and user acceptance tests of the new modules before implementing the corresponding automated tests. This testing approach differs from traditional testing as the software is expected to change over time, regardless of a defined launch schedule.

4 Use case of automated testing for web platform

The use case presented in the article represents the testing automation of a complex web application used by the operators at the *ICI Bucharest - Romanian Top Level Domain Registry*. Operators' authentication in the app is performed by username and password, with users' roles and access levels being already defined.

The development team is composed of six members working on Scrum methodology, including the Scrum master. Every Sprint lasts for 2 weeks. For development, organize and discussions, the team uses Atlassian Stash, a Git repository management solution for enterprise teams. It allows everyone to easily collaborate on Git repositories.

4.1 Application and environment

The system functions over a middleware architecture, meaning that it provides means to connect the various software blocks into an application where these can exchange information with relatively easy-to-use mechanisms. Middleware deals with component communication modes and can be used in a wide range of domains. The middleware provides a set of commands through an API for running specific tasks. The web applications interact with middleware through API calls and are widely used by operators and clients. These applications are under continuous development and integration, have a stable user interface and initially were manually tested before propagation to production environment. The manual testing process was extremely time consuming for developers and operators, specifically before releases, there-

DOI: 10.12948/issn14531305/22.3.2018.03

for we started investigating planning and developing an automated testing solution.
The servers and the machines are monitored with dedicated solutions, and the middleware includes unit testing, thus it was as important to design automated functional tests on the client side to check that there are no errors in the code, all elements are visible and operating correctly, as this application is highly used, with thousands of operations performed each day.
According to OASIS Test Assertions Guidelines Version 1.0 [13], a document containing assertion tests must be developed before implementing actual tests. The document should be updated whenever a change in the web-side platform is required. Therefore, the starting point consisted in the elaboration of the test assertions document containing all the operations that the user can perform on the app. This was a time-consuming process and includes all the inputs and outcomes of the user-side operations. To decrease pressure on developers, the operators participated in the description of the tests.

4.2 Technologies for developing testing automation

One of most widely used tools for automated code testing and bug prevention is **Selenium**, a framework supporting also headless testing, which can be integrated with a Continuous Integration (CI) server such as Jenkins or Travis. Selenium consists of a suite of tools for automating web browsers and provides a complex set of testing functions for web all types of web applications across multiple platforms, as it runs in most browsers and operating systems. It is highly flexible because it allows multiple options for locating and testing UI elements with the goal of validating expected test results against real-time application behavior.
Selenium provides interoperability with most programming languages such as Python, C#, Java, Ruby, thus it can easily be integrated in testing frameworks. Selenium basically consists of two main components the Selenium Webdriver and Selenium IDE. Selenium Webdriver is the core engine driving the browser natively as a user either locally or on a remote machine using the Selenium Server. Selenium WebDriver accepts commands and sends them to a browser through a browser-specific browser driver, which sends commands to a browser and retrieves results. Selenium WebDriver does not need a special server to execute tests. Instead, WebDriver directly starts a browser instance and controls it [9].
Selenium IDE is a complete integrated development environment (IDE) for Selenium browser-based regression automation suites and tests that enables fast development of bug reproduction scrips. It facilitates recording, playing, editing, and debugging tests. Selenium IDE was initially implemented as a Firefox Add-On and it is recently available on Chrome also.
Phantom JS is a headless WebKit scriptable with a JavaScript API for web page interaction automation that enables navigation, taking screenshots and test assertions. All these key features make it a common tool used to run browser-based unit tests in a headless environment.
Being driven by the need for testing web applications headless on a CentOS distribution we started analyzing the various options for designing the architecture of a testing system that could ensure flexible and accurate application testing. As a first step there were analyzed several configurations but only two were chosen for actual implementation and capabilities testing: Selenium Webdriver with Firefox browser used with Xvfb display server and Selenium Webdriver with Phantom JS. There were generated twenty test cases using Selenium IDE, then were exported and run. One of the tests performed to a form, consisting in asserting true the presence of a text field after clicking a “Submit” button failed on PhantomJS although using Firefox the test returned “ok". The functionality was then manually tested and was working.
The conclusion was that even though PhantomJS is a functional headless browser, it is not a real browser that users actually use while Firefox run with Xvfb provides much more accurate tests, within current environment.

DOI: 10.12948/issn14531305/22.3.2018.03

Selenium is a powerful automation testing tool as it is extremely flexible as it allows adding new functionalities to both Selenium test scripts and Selenium's framework to customize test automation.

Jenkins, a Java-based open source solution is a server used to deliver continuous build, is the tool to complete this task. It has the capability to monitor any job defined as a cron, SVN or GIT. A continuous integration server is designed to automatically or manually trigger complex workflows to build, test, and deploy software components [10].

Although it is a platform focused on building software systems, Jenkins-CI can easily be expanded with over 800 extensions for complex computational tasks. We can use Jenkins's powerful distributed model for CI to run our Selenium tests in parallel on a Jenkins cluster. For an Agile team, Jenkins provides everything needed for a robust continuous build system. Jenkins' extensibility allows the system to adapt to many different pre-existing environments. To ensure code stability, good collaboration between developers and fast release cycles, Jenkins is set up to build selenium tests automatically on every pull requests made on the Stash Server.

The initial plan was to have a high degree of granularity and to create tests for each element. During the development phase it was noted that this is time-costly as each test implied authentication, form-completions procedures, run middleware commands, test itself and logout. As a result, the team changed the approach to create larger tests, for example a single test for an entire form instead creating test for each field. This decreased the granularity level but the time savings were a considerable advantage.

4.3 Automated testing system

The environment presented above in Chapter 4.1 runs on Linux based servers. In this context the main concern that arises when designing the architecture of a system for functional testing of web applications is that there is no display output for the browser to launch in. To overcome this issue the team configured the tests to launch the browser virtually using Xvfb virtual frame buffer server and Firefox.

Detailed test cases were specified in the test assertion documents and 119 tests were created for covering them. The work procedure was to develop each test in Selenium IDE, installed in Firefox installed on a machine with display output. The tests included assertions for checking the presence of elements on the web page and continued with checking the messages that were returned if one or more fields were not filled in or filled in incorrectly. After all these checks, the fields are filled with valid data (e.g., valid email), the data is sent and the confirmation / success message is recorded. Programmers decided depending on the case which is the best option to check the presence of the elements - wait for, assert presence, verify. Each of these procedures have several options. From the Selenium IDE short menu, one can manually select the required assertion command from a list of commands provided in the Recording Addition.

Each Selenium test was recorded and exported as Python2 unit test and included in a single Test Suite.

The middleware API was often used to perform certain tasks in the background and decrease the time required by test run – instead of using web forms to create data (registrant details, domain information etc), API commands were used for this tasks.

The figure bellow shows the workflow of the testing system using Jenkins Continuous Integration Server:

DOI: 10.12948/issn14531305/22.3.2018.03

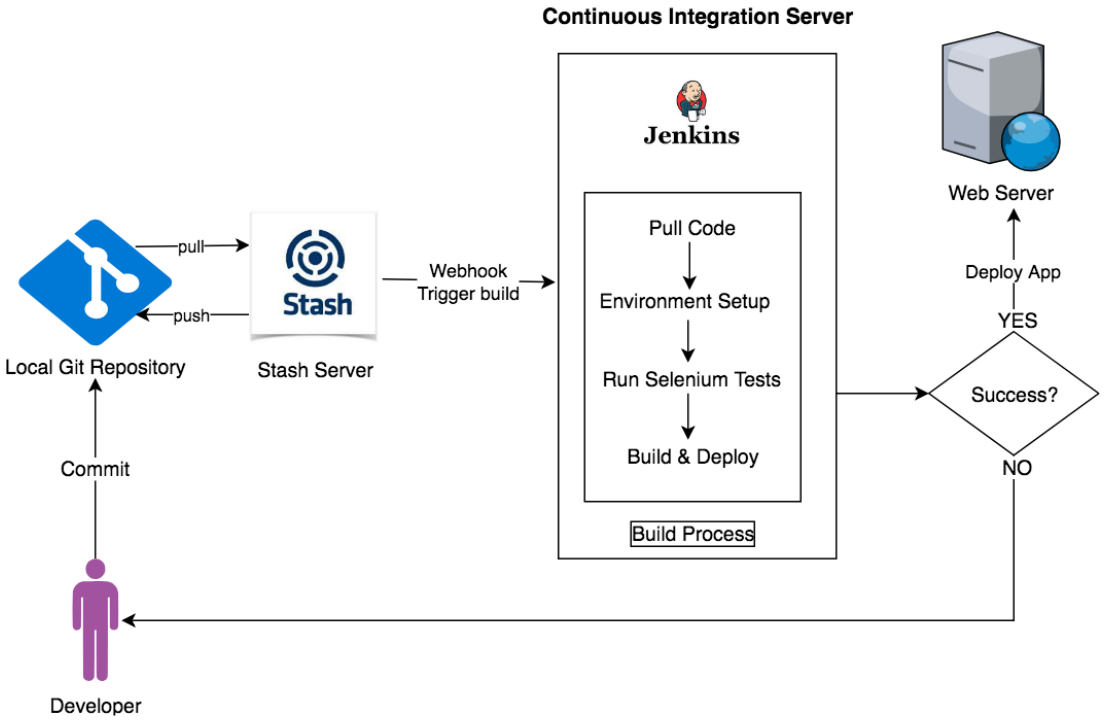

Fig 2. Workflow of the testing system

The software developer commits the code into the local repository, then pushes the code onto the stash server in its own repository. Once the changes are pushed, here are reviewed by the scrum master, then the stash server triggers a webhook which notifies the Jenkins server - the continuous integration server. Jenkins pulls the code, then sets up the test environment and runs the selenium tests.
Then creates an email report for all the tests and sends it to development team. If any of the tests fail, the code is rejected and the developer must review it and correct it. If test suite runs successfully the software is deployed into the production environment.
The following figure is an Archimate diagram showing the artefacts, functions and results(test reports) and relationships between them in a continuous integration environment using Jenkins:

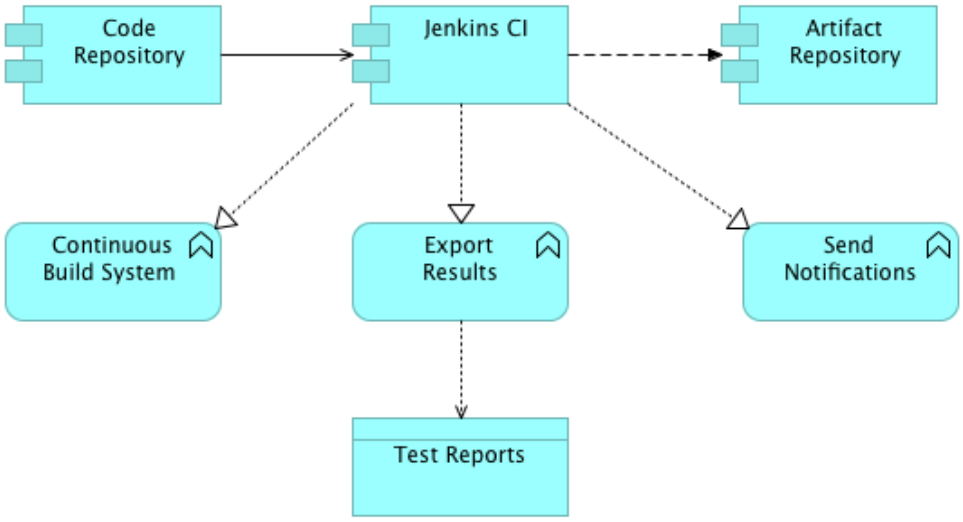

Fig. 3. Continuous integration workflow with Jenkins CI

DOI: 10.12948/issn14531305/22.3.2018.03

Impact evaluation

The period taken into consideration is six months. During these time, the system was triggered at each code submission and also run manually by the Scrum Master to ensure reliability.

Table 1 presents details of the automated testing system runs.

Table 1. System runs

Code operations	Details
Code submissions	174
Tests runs	206
Success	166
Failures	40
Average duration (minutes)	38
Updates in the testing system	22

Each failure was reported to the developer that submitted broken code and to the specified reviewers. The code submission was automatically denied from production environment.

The overall impact of the deployed system was a decrease in time spent for testing and a decrease in the number of bugs in the production environment. Initially there were 5 developers and 3 operators testing the system manually before each major release for approximately 1 week and after the automated testing system implementation the number reduced to 2 developers and 1 operator performing manual tests.

Disadvantages

The development and maintenance of an automated testing solution requires considerable effort on the development team and costs on the client when it comes to complex web applications, especially when the user interface changes frequently. In these cases, it can prove a hard task to create and maintain automated tests for dynamic contents.

The assertion document must be elaborated considering all the aspects, including different account types in case the displayed content or the client interface is different. These situations require additional development effort.

5 Conclusions

Automated software testing primarily reduces human errors, either in development or in manual testing. Test results can be stored in a database and advanced statistics can be developed. The decision on whether to perform automated tests varies from one organization to another, but in times where Agile development is spreading for faster software development, the automation of tests becomes a requirement for a successful implementation.

Testing automation on user interfaces is the solution when the interface is stable and provides key elements that are rarely or never changed. The alteration of the interface implies the reconstruction of test-cases and an analysis on costs-benefits must be done by the client prior to the decision of developing automated tests.

By implementing automated testing, the software producers gain significant cycle-time and quality improvements. The time cycles for software releases are shortened and the reliability of the UI is increased.

References

[1] Alexandros, N. K., Sakas, D. P., Vlachos, D., and Dimitrios, N. K. (2017). Comparing scrum and xp agile methodologies using dynamic simulation modeling. In Strategic Innovative Marketing, pages 391–397. Springer

[2] Elfriede, D., Garrett, T., Gauf, B.. Implementing automated software testing: How to save time and lower costs while raising quality. Pearson Education, 2009.

[3] Lianping, C. "Continuous Delivery: Overcoming Adoption Obstacles", Continuous Software Evolution and Delivery (CSED) IEEE/ACM International Workshop on, pp. 84-84 , 2016.

DOI: 10.12948/issn14531305/22.3.2018.03

[4] Pestak, T., Rowell W., Automated Software Testing – Practices and Pitfalls, 2017, STAT COE-Report-02-2017
[5] Sitaraman, S., Bar, R., Test Automation Strategies in a Continuous Delivery Ecosystem, Cognizant, 2016
[6] Schwaber, K. & Sutherland, J., The scrum guide. 2016.
[7] Udroiu, M & Vevera, V. (2018). LIFELONG LEARNING FOR RAISING CYBERSECURITY AWARENESS. 5381-5387. 10.21125/inted.2018.1272.
[8] Continuous Software Development. Available: https://searchsoftwarequality.techtarget.com/definition/Continuous-Software-Development.
[9] Selenium HQ browser automation - documentation. Available: https://www.seleniumhq.org
[10] Jenkins - documentation. Available: https://wiki.jenkins-ci.org
[11] Standard Glossary of Terms Used in Software Engineering (2011). Available: https://www.astqb.org/documents/Standard-glossary-of-terms-used-in-Software-Engineering-1.0-IQBBA.pdf
[12] Automated Testing Advantages, Disadvantages and Guidelines. Available: http://www.exforsys.com/tutorials/testing/automated-testing-advantages-disadvantages-and-guidelines.html
[13] OASIS Test Assertions Guidelines Version 1.0. Available: http://docs.oasis-open.org/tag/guidelines/v1.0/testassertionsguidelines.html
[14] Extreme Programming http://www.extremeprogramming.org/
[15] What's the Difference? Agile vs Scrum vs Waterfall vs Kanban. Available: https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban

Dragoș SMADA graduated the Faculty of Electronics, Telecommunications and Information Technology in 2005. In 2012 he graduated the Documents Information Management Master program organized by the University of Bucharest. Currently he works as a Senior Researcher at I.C.I Bucharest. His main areas of interest are Big Data, Internet of Things, software engineering, automated testing, information security, software architecture. He participated in both national and international research projects in the IT&C. He published as author and co-author of journal articles and scientific presentations at conferences.

Carmen ROTUNĂ has graduated the Faculty of Mathematics and Computer Science, University of Bucharest Database and web technologies Master program. Currently she works as a Scientific Researcher with expertise in Software engineering, eHealth, eServices, eGovernment, IoT, Big data at the "National Institute for Research & Development in Informatics", Bucharest. She was a team member in several international and Romanian research projects, three of them in the eServices domain: "Simple Procedures Online for Cross-border Services (SPOCS)", Electronic Simple European Networked Services(e-SENS) and The once-only principle project(TOOP), where she is currently the WP2 leading architect for the Romanian team and piloting coordinator. She published several articles, co-authored project deliverables and collaborates as a reviewer for scientific publication

Radu BONCEA is a Researcher at I.C.I. Bucharest. He has been involved in several large European projects such SPOCS, eSENS, Cloud for Europe and The Once Only Principle. As a Ph.D. student at Electronics, Telecommunications and Information Technology, he's interested in IoT and Cloud Computing related technologies.

DOI: 10.12948/issn14531305/22.3.2018.03

Ionuț PETRE graduated the Faculty of Electronics, Telecommunications and Information Technology in 2005. He is a PhD student at University Lucian Blaga from Sibiu, Faculty of Management. Currently he works as Researcher at *I.C.I Bucharest*. His main areas of interest are Internet of Things, e-Government, Big Data, software engineering, automated testing, digital libraries. He is involved in research projects specific to the Information Society. His research was published in journal articles and proceedings of conferences.

DOI: 10.12948/issn14531305/22.3.2018.03