A user-centric approach to test automation of web-based applications

Ionuț PETRE
ICI Bucharest
8-10 Averescu Avenue
ionut.petre@ici.ro

Dragoș-Marian SMADA
ICI Bucharest
8-10 Averescu Avenue
dragos.smada@ici.ro

Radu-Marius BONCEA
ICI Bucharest
8-10 Averescu Avenue
radu@rotld.ro

ABSTRACT
Modern society is imposing a faster rhythm in most of the life aspects, with software development being in the front line. Business requirements are stressing development teams for faster releases of new features and functionalities of the applications. The software developers adapted to the market demands by migrating from traditional development models to what is known as "continuous delivery". This implies short cycles of software development to ensure a higher degree of reliability for application releases. The use of this approach leads to a frequent building, testing and releasing of the software. Modern web applications offer a wide palette of user interactions and reached the level where their behavior is very similar to native applications. As a result, automation of the tests on the app behavior on the client side should be a priority for software development teams. For web-based apps, the client-side experience is the most important factor, as it is the most important factor deciding the application success. Frequent releases can induce many errors in the app's source code and can affect the functioning on the user side; tests automation is the solution that can prevent flaws in the app behavior at the user.

Author Keywords
continuous delivery; user-centric test; automatic web tests; app behavior tests; continuous deployment

ACM Classification Keywords
Software and its engineering

INTRODUCTION

In a fast-pacing society in which technologies are emerging continuously, software development is a continuous task for every business or institution that wants to maintain a high level of quality. Organizations need unprecedented velocity and agility to seize new opportunities and software development teams are under pressure to deliver high-quality software at a faster rate than ever before. Drawn-out development projects are no longer acceptable in the conditions of the modern business demands. The development team must stand up to this challenge while facing the most diverse technology ecosystem in the history of computing [5]. The creation of software is a team effort, with a wide range of aspects that require integration and conflict solving. The number of web-based applications is increasing, and so are the levels of reliability and complexity required to develop the apps. Scalable, instantly-accessible multi-user, multi-component, multi-tier systems are being required to perform mission-critical tasks in an environment of continuous delivery and operation, with an expectation of zero downtime and easy maintenance and administration [3]. A poor communication between team members or a poor planning of tasks can lead to major failures. In global software engineering, communication and coordination become more challenging, and that fact affects the quality of the product [1].

A proper test automation solution reduces the time and resources required to adequately test software, while also increasing the level of quality of the application under test [2].

CONTINUOUS SOFTWARE DELIVERY

Continuous delivery represents a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time [4]. The goal is to build, test, and release software application versions at a frequent rate. Continuous delivery is different than the concept of continuous deployment, where any change is automatically deployed to the production environment. Continuous delivery means that the team ensures every change can be deployed to production but may choose not to do it, usually due to business reasons. In order to do continuous deployment, one must be doing continuous delivery [10]. Continuous delivery and tests automation are two symbiotic concepts with the goal of providing a faster release of new versions of the application.

Agile development

Facing business demands to accelerate time for software release, developers are switching from traditional development models to continuous delivery based on Agile development methods. **Agile development** is appealed to be an innovative and receptive effort to address users' needs concentrated on the prerequisite to distribute applicable working business applications faster and inexpensive [6]. 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 [7]. For development teams to take application agility and efficiency to the level required by the modern society, three major concepts must be followed [5]:

- Focus on modern software architecture paradigms and frameworks. These are the new trend in software development and can reduce considerably the time to launch an application.
- If possible, replace heavyweight platforms with lightweight application infrastructure. This can bring scalability to the system while maintaining fast development.
- Invest in automation and strive for increased agility across the life cycle. Automation is a necessity due to the fact that modern architecture and lightweight infrastructure are characterized by changes which cannot be manually managed.

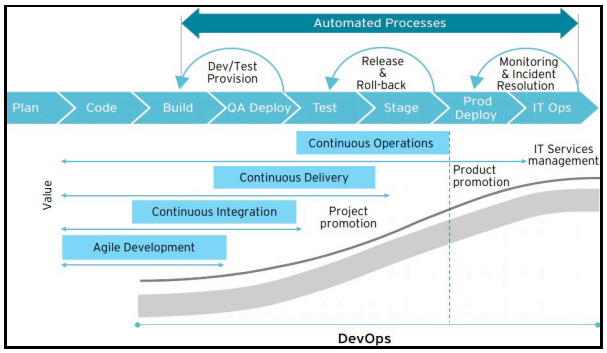

Figure 1 - Continuous Delivery[7]

Most organizations follow a software development cycle plan to develop, maintain, modify or enhance their software. The development cycle is a complex job that allows team managers to estimate development costs, to set timeframes and plan tasks. Developing quality applications require teamwork and complex planning of development cycle. Given the volume of work and tasks that must be performed by the development team, the appearance of flaws can hardly be avoided.

Continuous delivery is driven by continuous integration, where the code is logged-in several times a day and then re-compiled, leading to frequent deployments, which mandates greater levels of process automation. For a successful project, automation of the build and deployment process is critical; this ensures a self-testing build and makes the entire process transparent and agile system [7].

Basic considerations on user-centric tests

The web-based applications are written to run in a browser, therefore testing these applications in an automated manner must involve the simulation of browser behavior. The testing automation from a user-centric perspective involves the usage of software tools that perform tests repeatedly on the application by simulating the actions of an operator or user in a browser. The automation tests for a web-based application must be focused on the user experience, having the goal to raise the users' confidence in the quality of the application. The user-centric approach also provides advantages for the development team and for the clients. The developers can identify and solve issues, and the clients can verify that the application fulfills specifications and requirements.

Test automation is not appropriate in all cases of software development. There are times when manual testing is more suitable to test an application – such is the case when the application interface is changing frequently. Therefore the user-centric automation 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 organization prior to the decision of automation.

USE CASE OF USER-CENTRIC TEST AUTOMATION

Solutions and environments

The use case presented in the article represents the test automation of a complex web application used by the operators at the Romanian Top Level Domain registry. The authentication in the application is done by username and password, with users being already set.

The web application functions over a middleware architecture, meaning that this provides ways to connect the various software components into an application so that they 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 application is written in *PHP* while the middleware is written in *Python*. The middleware provides a set of commands that can be integrated into the test scripts. The application is under continuous development and integration with a stable interface, therefore, tests automation was considered a proper solution for the development team.

Selenium is an open-source solution consisting of a suite of tools for automating the behavior of web browsers across various platforms. It provides support for several programming languages, such as Java, Python, C#, Ruby. It has two options for usage [8]:

- Selenium IDE – an add-on for Firefox browser to record-and-playback interactions with the browser;
- Selenium WebDriver – a collection of language-specific bindings, to create browser-based regression automation suites and tests. The scripts can be used in different environments.

Selenium Python bindings provide a simple API to write functional/acceptance tests using Selenium WebDriver.

Through Selenium Python API the functionalities of Selenium WebDriver can be accessed for widely-used browsers such as Firefox, Internet Explorer, Chrome, Safari, phantomjs, etc [8].

Tests automation implementation

The starting point is represented by the elaboration of a document containing all the operations that the user can perform on the app. The test assertions in the document were derived from the OASIS Test Assertions Guidelines Version 1.0 [9] that describes best practices in applying a general test assertion model, what test assertions are, their benefits, and how they are created. The main purpose of the test assertions is to remove the ambiguities or statements that can lead to impractical or unnecessary development efforts. The document can be used whenever there is the need for a test change or for creating new tests. If not developed by the specification authors, test assertions should be reviewed and approved by them which will improve both the quality and time-to-deployment of the specification. Therefore, best results are achieved when assertions are developed in parallel with the specification. An alternative is to have the leader of the team that is creating test suites to write the test assertions as well and to provide feedback to the specification authors [9]. In this case, the document was elaborated by the authors of the application specifications in collaboration with the team of operators. The document is updated through the test development process with situations and test cases that the operators missed

Selenium IDE comes with an interface for developing automated tests, with a recording feature which records user actions as they are performed. Each user action is translated into a Selenium command that can be personalized. The recording can be exported as a reusable script written in a programming language that can be later executed on a server. Selenium IDE was the tool used for translating into user behavior on the browser for the assertion tests specified in the document. Each test recorded user actions in various key elements – authentication, application menu, links, buttons, or confirmation/warning messages. The records were exported as Python2 WebDriver script files. The recording was done in Firefox browser, on a Windows platform.

The script files of tests must run headlessly on a continuous integration server, running with Linux Centos 6, which does not provide a display output to launch a browser. The issue is that Selenium requires simulating a web browser in order to run the python files exported from IDE. The solution to this problem was **Xvfb**, an X server for machines without hardware display which emulates a virtual framebuffer using the virtual memory of the machine and in our case is used to simulate the display needed for the browser. To link the tests from scripts to Firefox, another tool is required: **Geckodriver**. This, in fact, is a proxy for the W3C WebDriver-compatible clients to interact with Firefox.

The commands for the installation of Selenium, Xvfb, and Geckodriver on the application' server:

```
> yum install Xvfb firefox
> pip2.7 install -U selenium==2.53.2
>wget geckodriver-v0.11.0-linux64.tar.gz
> tar -xvzf geckodriver-v0.11.0-linux64.tar.gz
```

The work procedure followed by the team was to perform the test in Selenium IDE, installed in Firefox. The assertions were made for elements considered stable on the web page, such as <h> elements, action buttons, menu links, alert messages, confirmation messages, user-interaction alert windows:

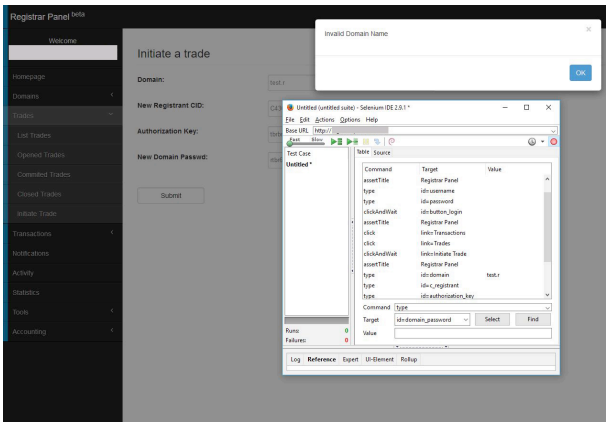

Figure 2 - Use of Selenium IDE for recording web activity

After recording the operator actions in the web application, we run the test from Selenium IDE to ensure proper assertion of the saved record.

Each test is exported as Python2 unit test WebDriver in a CentOS platform. Running the test under Python led to the appearance of errors due to the insufficient transition time between the page and any messages or alerts that appeared. This required the manual setting of additional pauses of 1 or 2 second between the launching and closing of any messages that require user interaction.

Tests automation server

For a complete automation process, the tests must run without user assistance. **Jenkins**, a Java-based open source solution 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. Although it is a platform focused on building software systems, Jenkins-CI can easily be expanded with over 800 extensions for complex computational tasks. To create, organize and discuss, the development team uses Atlassian Stash, a Git repository management solution for enterprise

teams. It allows everyone to easily collaborate on your Git repositories. To ensure code stability, good collaboration between developers and fast release cycles, the developers integrated Jenkins to build tests automatically on every pull requests made on the Stash Server.

Stash Webhook was configured on the Stash server to trigger Jenkins automatically. In the workflow, a developer creates a branch on his local repository, completes its task, then commits and pushes the changes to Stash and creates a pull request. When the pull request is merged, Jenkins is triggered automatically to make the build and run the Selenium tests.

Disadvantages

The development and maintenance of tests automation imply 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 effort.

CONCLUSION

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.

User-centric tests automation presents major advantages:

- Better experience for the users on the client side, as the automation increases the application's reliability;
- Reduces the load on the development team. Code review is performed faster as the reviewers only verify the code that passes the client-side tests;
- Faster integration of new developers in a team. The tests assertions document leads to a better understanding of the application for new-comers;
- In the event of committing bad code, the committed code fails the tests, is automatically denied so the code flaws do not reach the production environment

Besides the development of the tests, there are virtual machines that need to be provisioned and configured, and testing environments must be set and deployed. But in a long-term vision, tests automation reduces the development time and decrease the risk of bugs that might interfere with the application behavior and raises the users' confidence.

ACKNOWLEDGMENTS

This work has been supported by a Romanian grant, financed by ANCS under COGNOTIC 1609 0802 / 2016. We thank all the colleagues who participated in the tests automation development and also provided helpful comments on previous versions of this document.

REFERENCES

1. DeFranco, J.F., Laplante, P.A. An Integrative Review and Analysis of Software Development Team Communication Research. *IEEE Transactions on Professional Communication PP(99) · June 2017*
2. Dustin, E., Rashka, J., Paul, J. Automated Software Testing: Introduction, Management, and Performance, 1999
3. Hearn, J.E., The Development of an Automated Testing Framework for Data-Driven Testing Utilizing the UML Testing Profile, *2016*
4. Lianping, C. "Continuous Delivery: Overcoming Adoption Obstacles", Continuous Software Evolution and Delivery (CSED) *IEEE/ACM International Workshop on, pp. 84-84*, *2016*
5. Knoernschild, K., Knipp, E., Watson, R., Kenefick, S., Brian, D., Olliffe, G., Holz, B., Dayley, B., Cheetham, L., Modernize Application Development to Succeed as a Digital Business, *Gartner, March 30, 2016* https://www.gartner.com/doc/3270018?refval=&pcp=mpe
6. Saleh, S.M., Rahman, A.M., Asgor,K.A. Comparative Study on the Software Methodologies for Effective Software Development. *International Journal of Scientific & Engineering Research, Volume 8, Issue 4, April-2017*
7. Sitaraman, S., Bar, R., Test Automation Strategies in a Continuous Delivery Ecosystem, Cognizant, 2016
8. http://www.seleniumhq.org/, accessed in May 2017
9. Test Assertions Guidelines Version 1.0, 2013 http://docs.oasis-open.org/tag/guidelines/v1.0/cn02/guidelines-v1.0-cn02.pdf
10. https://martinfowler.com/bliki/ContinuousDelivery.html , accessed in May 2017