Towards Automated Web Functional Testing Using Predefined Templates

Iulia Ștefan, Ovidiu Stan, and Liviu Miclea

***Abstract*—The paper describes the characteristics of a new experimental automated web testing approach and component that allows the users to test functionalities of a web-based software under test (SUT) using a succession of parametrizable predefined test templates. The parametrizable templates usage supports a minimal validation of a finalized work without automated testing skills development because these are provided by default. The independent web developers without the required resources to automatically test the functionality of their applications could successfully integrate solutions based on the presented approach into their practice. The tactic allows the configuration of the test cases with the specific values required by the desired SUT characteristics through a novel graphic-based policy for the definition of test cases.**

***Index Terms*—Test automation, functional testing, graphic-based testing approach, test templates, knowledge database.**

I. Introduction

Software testing represents a process to verify and validate whether the product completely satisfies its predefined requirements, usually provided through specific documents such as “Software Requirements Specification”, “User Interface Requirements”, “Functional Requirements Document” and a few more. These documents usually deliver the input and the best references for specific test development. Data based technique implies the usage of test data vectors such as (input data, expected results) in order to obtain a certain outcome/output based on the software execution.

Generally, static and dynamic approaches are used in order to test the product. The implementation complexity and desired requirements fulfilment transform the testing process into a complex activity, the possibility of an exhaustive test case generation and execution being automatically excluded based on its evident lack of efficiency.

The current work was developed in order to address a certain degree of normalization to the manual definition and automated execution of the test cases for web developments for minimal criteria such as: validation of forms, content (mainly text), user case scenarios, and functionalities by using predefined formats for test implementation.

Also, for smog or acceptance testing, usually based on the client’s usage experience and requirements, the availability of an approach that does not need anything else but web navigations skills and business know-how, could be a strong plus. The web based experimental component developed implements the graphic-based new approach’s functionalities. The article includes 7 chapters: I. Introduction - briefly presenting the concepts and the main objective, II. Web testing and related work – introducing several published results regarding solutions for requirements-based testing and the associated challenges, III. Web functionalities analysis – the chapter underlines the common functionalities to multiple web applications that need testing, a common ground for parametrizable templates IV. The proposed approach - the chapter evidencing the solution for implementation, V Comparative results - underlines reports of runs for similar tests in Selenium IDE, a web-based approach of test generation using record and playback features, and VI. Conclusion – pointing out the finalized and future work.

Manuscript received March 15, 2022; revised May 12, 2022. This work was supported in part by the Romanian Government under Grant PN-III-P2-2.1-SOL-2020-0157.

Iulia Stefan, Ovidiu Stan, and Liviu Miclea are with the Department of Automation, Technical University of Cluj Napoca, Cluj, Romania (e-mails: {iulia.stefan, ovidiu.stan, liviu.miclea}@aut.utcluj.ro).

II. Web Testing and Related Work

Automation in tests’ generation of the test cases, execution or/and results evaluation is a representative concern in a competitive web development market. There are several approaches regarding the testing of web pages.

Yu *et al.* present a new method for automatically generating test cases using page objects [1]. The application exploits the new states of the application and automatically generated tests, accordingly. The method is useful for exploratory testing but does not offer the possibility to explicitly specify the expected output regarding the evaluation of a test case obtained results, specifically designed for the familiarized testers with java and Selenium SDK, as our work does.

Madhan *et al.* formalize the software requirements’ process for critical systems through a natural language processor in order to obtain completeness and correctness of the requirements’ specification [2]. Our work tends to obtain the same completes and correctness by providing templates for the test case definition through a friendly user interface that reduces the effort of the developer for the software functionalities description.

Other authors try to resolve the issue of unidentified pages in test case generation, by expanding the Selenium IDE capabilities [3]. Here, the automation is enhanced without explicitly specifying the initial requirements regarding the web page functionalities: if the identified pages should exist or not, or if they should have a specific content or not. Similar to our experimental tool, a minimum knowledge about automated tool in testing is required in order to finalize a testing activity.

In [4], the UML specified requirements are the starting point in the automated test case generation proposed technique, in order to automatically define test cases. Our solution replaces the UML specification of the requirements by providing explicit options inside de graphical interface, regarding input types, input characterization properties, expected output identification data, URL's description and details and maintains the automation level at test run and at test result analysis.

Related work regarding web testing involves the Finite State Machine (FSM) [5], [11], [12] modelling approach for automatically generating the tests. These instruments allow the model creation directly with the testing tool. Our solution reduces the model design to a series of specific features in association to web based graphical user implementation, in order to simplify the formation of tests and the usability of the software component.

The proposed experimental tool provides a graphical user interface to define, in fact, different states of an SUT regarding a user interaction with the SUT, through the graphical interface of the component, extracting the specific configuration pairs of parameters in a JSON files, and populating the tests suites with desired input values/expected results. The proposed experimental tool runs the test suites, with specific tests and test cases, automatically assesses the results and exposes them to the user.

Other new approaches include, in a certain degree, the usage of digital image processing and visual aids for UI testing. In [6], for example, similar to code smells in source code, the developed tool has visual design triggers called "smells" to point out the need to address inconsistent aspects of the UI design. In our tactic, the visual aids are offering a graphic based interaction with the component in order to implement, in fact, user stories.

Approaches based on graphic interaction with the tester are reported mainly on image automated evaluation between expected/actual snapshots comparisons including evaluating the instruction set architecture's activity [7].

The current work is not focusing on the automated evaluation of results, instead is focusing on graphic aids for user story definition for tests based on predefined templates for tests. These templates represent, in fact, a test knowledge database. The current work represents a similar approach to behaviour driven testing in tools such as Cucumber, but with exploitation of graphic aids, not natural language programming.

The work could be, also placed, in the context of Concept Maps presented by Elsayed *et al.* [9], as a resource for content organization, in that case, in education.

III. Web Functionalities Analysis

There are several types of web content, dynamic or static. Nowadays, the main majority is represented by prolific and experienced web users and based on this experience, one can confirm that all needed functionalities exposed by the GUI of a web application is mainly based on links, events (movement, clicks) and input elements. The user's actions activate the dynamic exposure of the content.

The analysis of several web sites, from e-commerce and blogs, to business sites, media content, social media indicates that the specificity of web exposure and implementation supports the idea of predefined templates for a functional testing approach, based on the user's actions through the GUI interface mainly, opened in browsers.

These templates represent a knowledge-database of test structures. As also presented in a previous work by Stefan *et al.* [10], these functional properties are mainly the ones exposed in the following table.

TABLE I: Web Sites Functionalities

User actions	Site's specific functionalities exposed by GUI	Test Templates developed
Form's completion	User registration, comments, email toward a specific address, filters (e-commerce, etc)	Predefined structure of tests for form evaluation
Specific succession of pages	Web site navigation	Predefined structures for page succession based on content recognition

A web site navigation means, from the user's point of view, events, mainly mouse/hand events, and data completion of forms. For test case generation based on user stories, the graphic based approach is an improvement in relation to a beginner level of automated testing.

IV. The Proposed Approach

The proposed approach makes available, for a developer not accustomed with the automated testing tools and testing methods, a minimal set of configurable test case templates.

The use case diagram presented in Fig. 1 is offering a clarification regarding the available actions for a specific user when accessing the software component:

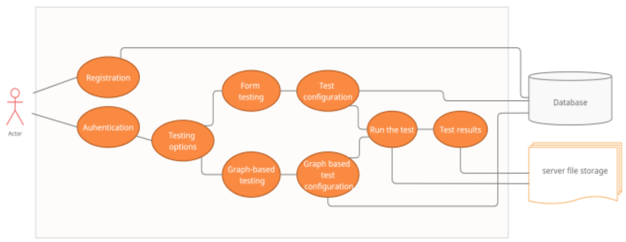


Fig. 1. UML use case diagram.

In Fig. 2, the overall architecture of the web component exposing the structure and associated servers (web server, database management server, Selenium standalone server) with the testing component [15] is available.

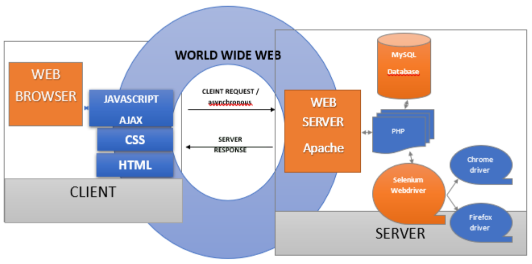


Fig. 2. Client- server component's architecture.

The Selenium server is running all tests based on the configuration of the servers' side predefined templates. These templates are generating different test results based on the parametrization made from the web interface of the testing component.

The client-side functionalities run through the web browser, and the AJAX calls update the database in relation to a specific feature of a test.

Several features have been implemented such as: authentication/registration for user access, new test cases definition based on predefined templates for testing forms and page navigation using the presented new graphic based design approach, test objects instantiation and parametrization, *data_provider* function implementation for parameterizing test methods, asynchronous data update and storage in the database/file storage, the possibility of individual parametrization of tests according to the expected behavior, automatic execution of tests, automatic user graphic interface integration/presentation of results obtained by automatic execution, directly in the component's web page.

A. The Chosen Frameworks and Software Packages

The tool was developed using PHP 7, PHPUnit, PHPUnit_Selenium frameworks, Selenium 2.x server, Selenium Webdriver, at the server-side component development.

PHP was chosen due to its HTML, JavaScript, CSS, intercalation capabilities in web documents, with serious improvements since PHP5, regarding OOP capabilities, and last, but not least, PHP is one of the widest used languages in server-side programming languages, in 2020 included, with a market share equal to 79.2%, having important competitors as: ASP.NET, Java, RUBY, JavaScript, as reported in [8] .

For current version of the component, the tests are run automatically against Chrome and Firefox platforms and Selenium webdriver, which was integrated based on its capabilities to automatically execute a web browser using its innate support for automation.

PHPUnit is integrated in order to automatically assess the test results by calling assert type functions, to efficiently provide the test data by data providers, to choose and define testing dependencies, to save test result after test run in XML files.

PHPUnit and Selenium integration [13], [14] is possible by using PHPUnit_Selenium and PHPUnit_Extensions_Selenium2TestCase packages. In order to access specific elements from the web documents via the Document Object Model (DOM), the PHP class DOMElement was used.

There are a minimum of 3 operations available for running our predefined test cases based on the server side of the proposed tool, functionalities are available through the web GUI of the tool:

- *Create account,*
- *Choose a test type* (pages succession/navigation, input validation, and combinations of first two, etc),
- *Configure test cases* with specific details (e.g. site under test web address, forms names, ids, resources used, input types for the forms to be tested, names of the pages, page's specific text for test evaluation),
- *Request* server to *run tests,*
- *Access test results generated* and sent by the server.

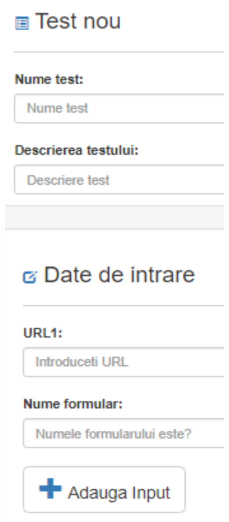
The solution was implemented using MVC architectural pattern. Data provided to the tests can be generated by genetic algorithm in order to obtain the site's specific content as presented in a previous work [15].

B. GUI Functionalities for the Client-side Component

The graphical interface (GUI) of the web client exposes functionalities for login and register, choosing a testing option, parametrization of the tests, results viewing.

As a novelty, the GUI exposes the graphic based definition of test cases solution. The two main possibilities are: to test forms and/or to test site navigation paths using specific assessment in relation to site content and event validation.

The form testing choice allows new test generation configuration or running/re-configuring an old test. Test configuration options include: test name (Romanian language "nume test"), test description (Romanian language "descrierea testului"), site under test (SUT) URLs, add input characteristics (type, id, name and/or size, value, if available) and the address of the next page when the forms' submit button is clicked (0.) When the configuration is completely specified, the "Send and test" button requests specific operation on the server's side.


Test nou

Nume test: Nume test

Descrierea testului: Descriere test

Date de intrare

URL1: Introduceti URL

Nume formular: Numele formularului este?

Fig. 3. Set-up of the form testing.

The site navigation test choice allows the user to specify the start page and the end page, with expected output.

The expected output is mainly a defining text inside the page in order to uniquely validate the page, besides the URL.

To define the test case, one needs to define nodes and relations. Each node has associated several methods or actions to it, as presented in Fig. 4: Are you a parent; Are you a child; Add specific URL (Romanian language "Introduc URL"); Add title (Romanian language "Introduc Titlu"); Add words on page (Romanian language "Introduc cuvinte pagina"); Add form characteristics; Add focus event; Add hover event.

The relationship parent – child defines the desired succession of the pages or content change based on user actions using, for example, a hyperlink clicks or hover events, or a specific form submission, presented in Fig. 5 [15].

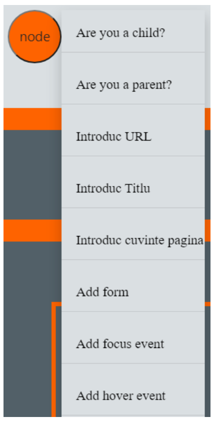


Fig. 4. The GUI based configuration of the site navigation test.

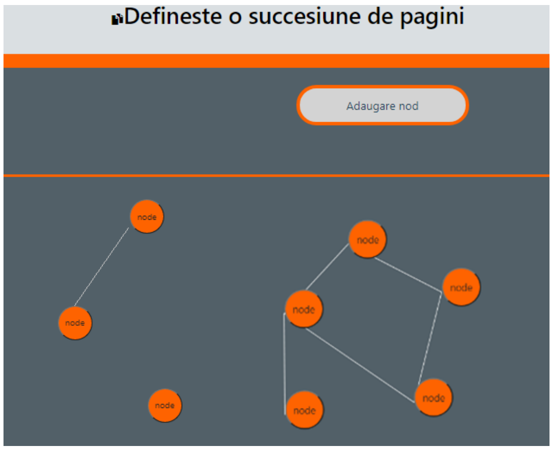
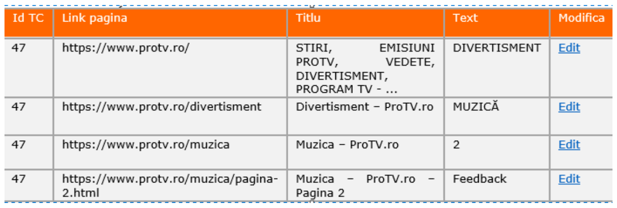


Fig. 5. The graphic based configuration of the site navigation test.

After the completion of the test's input data and click on the "Salveaza test" (translated as Save test), a button available under the graph, the succession of pages and events is available through a table format before running the test cases and the result display. The table format is an operation associated to the parametrization of the test, as presented in Fig. 6 [15].



Id TC	Link pagina	Titlu	Text	Modifica
47	https://www.protv.ro/	STIRI, EMISIUNI PROTV, VEDETE, DIVERTISMENT, PROGRAM TV - ...	DIVERTISMENT	Edit
47	https://www.protv.ro/divertisment	Divertisment – ProTV.ro	MUZICĂ	Edit
47	https://www.protv.ro/muzica	Muzica – ProTV.ro	2	Edit
47	https://www.protv.ro/muzica/pagina-2.html	Muzica – ProTV.ro – Pagina 2	Feedback	Edit

Fig. 6. Parametrization of the test – An extra step for correcting the graphic based test template parametrization.

Based on the request generated by the client side of the component, the server will access and run the required scripts.

Mainly, the server starts a tread for each test call in order to generate a test result file, by verifying the operating system and executing the test runner application (PHPUnit).

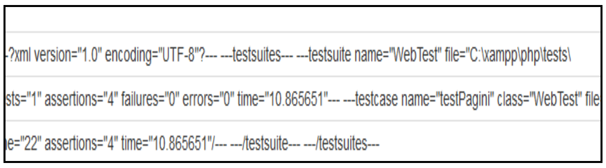
The site navigation test choice from the GUI of the client side represents a request for the server to execute a specific procedure. The script involves: extracting the test data recorded into the database, choosing the test template to run, automatically running the tests (access the web driver, open the chosen browser, accessing the URL of the SUT, loading the DOM document of the page under test, validating the DOM, extracting the URL and validating the page through the specified content, identifying the element chosen to navigate to, executing the event to trigger the navigation, identifying the new page's URL, automatically navigate to the new page/update the content, automatically asserting the content or other identification elements of the new page), publishing the result in the result file, updating the component result page with specific content.

The form test choice request, for the server involved in the execution, is similar and contains the following steps: extracting the test data, specifying the browser to run the tests, accessing each element specified by the form test in a json file, evaluating each input and its properties: expected (specified by the user) versus existing values (extracted by the test), saving the test result, sending the test result to the client (the page is updated).

Regarding the sequence of pages, for the implementation of the graphic definition characteristics of the tests in the browser, the HTML tag "canvas" was used in particular; inside, specific graphic elements (lines) were created in order to visually define the structure of the sequence of pages. Through the specific functionalities, buttons were created and added to visually abstract the concept of "page" as a node in the sequence of pages represented as a network of nodes and edges. In order to be able to associate a series of events to these buttons, to support the definition of the page sequence, JQuery methods were used. These events are: add parent node, add child node, add page title, add URL, add specific text, and more. The JQuery ajax method allowed the triggering of asynchronous requests to the server, extracting and saving relevant information from the database, such as unique ids and information associated with a sequence between the parent page - the child page. A conceptual representation of the sequence graph, in which, in fact, each node has a unique id generated and saved in the database, can be observed in previous work [15].

C. *Test Results*

The results are available from the user account and involve information related to number of assertions, tests, failures, test case name as available in Fig. 8.



```
-?xml version="1.0" encoding="UTF-8"?--- ---testsuites--- ---testsuite name="WebTest" file="C:\xampp\php\tests\
sts="1" assertions="4" failures="0" errors="0" time="10.865651"--- ---testcase name="testPagini" class="WebTest" file
e="22" assertions="4" time="10.865651"/--- ---/testsuite--- ---/testsuites---
```

Fig. 8. Test results.

Each test result is saved in a different xml file with a specific timestamp in order to obtain a unique identifier for each test run.

V. Comparative Results

Several tools were taken into consideration for a minimal evaluation of the current experimental tool, in terms of execution time: UFT One and Selenium IDE. Next, a short feedback related to one of those is presented.

Several test cases (TCs) regarding a predefined web site navigation were implemented in both, Selenium IDE and graphic based component. The next results were obtained, as presented in Table I.

TABLE I: EXPERIMENTAL RESULTS

TC	Test description	Selenium IDE	Graphic based component
#1	A 4 pages succession, 3 tests	<18sec	<=21sec
#2	A 3 text inputs form completion, 1 test	~3sec	~5sec
#3	Combination of 1 form completion and 2 pages succession	<10sec	<12sec

In conclusion, the tool could represent an option for functional testing and acceptance testing, but more tests should be conducted for reducing the run time.

VI. CONCLUSION

The presented experimental component offers a new method to evaluate and document at a minimum the functionalities of a developed site, by accessing predefined test methods templates. The user can also access previous test runs, previous configurations in order to re-test desired functionalities. An improvement, in relation to test report generation, could be represented by adding different details on the graphic aids, in order to establish the defect significance, detected by a test run.

The test report is not considered in the article due to the fact that, in the current version, it does not represent a novelty to the presented approach.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

All authors have contributed equally. Iulia Ștefan conducted the research and wrote the article, Ovidiu Stan and Liviu Miclea analysed data, reviewed and concluded feedback for the article.

ACKNOWLEDGMENT

This research was partially funded by the Ministry of Education and Research of Romania, grant number PN-III-P2-2.1-SOL-2020-0157 (10Sol/2020) and, also, represents research conducted by the first author during her PhD thesis.

REFERENCES

[1] B. Yu, L. Ma, and C. Zhang, “Incremental web application testing using page object,” in *Proc. 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, Washington, DC, 2015, pp. 1-6.

[2] V. Madhan, V. K. G. Kalaiselvi, and J. P. Donald, “Tool development for formalizing the requirement for the safety critical software engineering process,” in *Proc. 2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, Chennai, 2017, pp. 161-164.

[3] S. J. Lee, J. L. You, and S. Y. Hsieh, “Automatically locating unnamed windows and inner frames for web regression testing,” in *Proc. 2017 International Conference on Applied System Innovation (ICASI)*, Sapporo, 2017, pp. 184-187.

[4] D. Clerissi, M. Leotta, G. Reggio, and F. Ricca, “Towards the generation of end-to-end web test scripts from requirements specifications,” in *Proc. 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, Lisbon, 2017, pp. 343-350.

[5] B. Song, S. Gong, and S. Chen, “Model composition and generating tests for web applications,” in *Proc. 2011 Seventh International Conference on Computational Intelligence and Security*, Hainan, 2011, pp. 568-572.

[6] S. S. Singh and S. R. Sarangi, “ISAMod: A tool for designing ASIPs by comparing different ISAs,” in *Proc. 2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*, 2021, pp. 1-6.

[7] B. Yang, Z. Xing, X. Xia, C. Chen, D. Ye, and S. Li, “UIS-Hunter: Detecting UI design smells in android apps,” in *Proc. 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 89-92.

[8] Technologies Report. [Online]. Available: https://w3techs.com/technologies/report/pl-php

[9] O. Elsayed, C. Limongelli, F. Sciarrone, M. Lombardi, A. Marani and M. Temperini, "An on-line Framework for Experimenting with Concept Maps," in *Proc. 2019 18th International Conference on Information Technology Based Higher Education and Training (ITHET)*, 2019, pp. 1-8, doi: 10.1109/ITHET46829.2019.8937376.

[10] I. Stefan and L. Miclea, "The usage of contextual information to develop data test vectors," in *Proc. 2012 IEEE International Conference on Automation, Quality and Testing, Robotics*, Cluj-Napoca, 2012, pp. 302-306.

[11] A. Arora and M. Sinha, "Applying variable chromosome length genetic algorithm for testing dynamism of web application," in *Proc. 2013 International Conference on Recent Trends in Information Technology (ICRTIT)*, Chennai, 2013, pp. 539-545.

[12] H. Shahriar and M. Zulkernine, "PhishTester: Automatic testing of phishing attacks," in *Proc. 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, Singapore, 2010, pp. 198-207.

[13] Phpunit. [Online]. Available: https://phpunit.de/

[14] Seleniumhq. [Online]. Available: http://www.seleniumhq.org/

[15] I. Ștefan, *Methods for Increasing Dependability in Decision Making Process Levels for Cyber-Physical Systems*, UT Press, 2020.

[16] I. Stefan and L. Miclea, “The usage of contextual information to develop data test vectors,” in *Proc. IEEE International Conference on Automation, Quality and Testing, Robotic*s, Cluj-Napoca, may 2012, pp. 302-306.

Ștefan I. is a member of the Dependability System Research Group at Automation Department at Technical University of Cluj-Napoca. Her work and research interests include: automation testing, system dependability, computer aided design, augmented reality industrial application, serious games development for cognition, learner centered approach and methods in education. She is an IEEE Member. In 2015, she received the ITC Gerry Gordon Student Volunteer Award.

Stan O. is an associate professor at the Automation Department at the Technical University of Cluj-Napoca. His research interests include medical informatics, semantic interoperability, information management in the age of the Internet, dependability and fault-tolerant systems. Stan received a PhD in systems engineering from the Technical University of Cluj-Napoca. He is a member of IEEE.

Miclea L. is a full professor the Automation Department at the Technical University of Cluj-Napoca. He is also the dean of the same faculty. He is the author or co-author of 17 books, 40 research works and more than 180 scientific publications. His research interests include are dependability, cyber-physical-systems, agent systems. Miclea is a Senior member of IEEE and is regular the general chairman of the IEEE-CS-TTTC-AQTR conference.