From Functional Test Scripts to Performance Test Scripts for Web Systems

Federico Toledo Rodríguez[1], Matías Reina[1], Fabián Baptista[1],
Macario Polo Usaola[2] and Beatriz Pérez Lamancha[2]

[1] Abstracta, Montevideo, Uruguay

[2] Universidad de Castilla-La Mancha, Ciudad Real, Spain

Abstract. When modernizing systems the software is migrated from one platform to another. There are big risks concerning the performance the system should have in the new platform. A new system cannot take more time to perform the same operations than the previous one as the users will refuse it. Therefore, the preventive performance test is crucial to guarantee the success of the modernization project. However, the automation tasks for performance testing are too demanding, in terms of time and effort, as the tools work at a communication protocol level. Though not free, the functional testing automation is easier to accomplish than the performance testing automation as the tools work at a graphic user interface level; the tools are therefore more intuitive and they have to handle less variables and technical issues. In this article we present a tool that we developed for industrial usage to automatically generate performance tests scripts from automated functional tests. The tool has been used in several projects in the industry, achieving important effort savings and improving flexibility.

1 Introduction

Two very important quality aspects to reduce risk at the moment of *going live* with a system (the day that the system is released to the users) are the correctness and the performance of the functionalities. Typically, we can perform tests at different levels to verify and improve system functionalities: unit, component, integration or system testing. Development projects are generally iterative, planning many product releases during software project lifecycle, because they were planned as such or because different bug fixes and maintenance have to be implemented after release to production environment. Regression testing is needed (the test useful to verify in every release that the software does not have quality regressions), and because of this, different tools are available to automate the execution of these tests [1], generating tests scripts as a sequence of commands to simulate user's actions. In performance testing, load simulation tools are used to concurrently generate hundreds of users connected to the system under test (SUT) [2]. When the load is simulated the infrastructure experts analyze the health status of the system, looking for bottlenecks and improvement opportunities.

Toledo Rodríguez F., Reina M., Baptista F., Polo Usaola M. and Pérez Lamancha B..
From Functional Test Scripts to Performance Test Scripts for Web Systems.
DOI: 10.5220/0004586500120020
In *Proceedings of the 1st International Workshop in Software Evolution and Modernization* (SEM-2013), pages 12-20
ISBN: 978-989-8565-66-2

Taking into account that the automation for functional testing is easier than the automation for performance testing, and also easier to maintain and to understand, our proposal is to take advantage of the functional test scripts to automatically generate performance test scripts. Basically, for web systems, the functional test scripts are automatically executed while the HTTP traffic is captured. Later, the HTTP trace is analyzed to generate a performance test script model which is finally used to generate the script code to be executed by a load generator.

The rest of the article is organized as follows: section 2 goes deeper on automation of functional tests (regression tests) and performance tests, especially for web systems; section 3 presents the proposal that is then validated in section 4, showing the first results for the usage of the tool in the industry; and after mentioning the related work in section 5, the conclusions and future work is presented in section 6.

2 Background

An extended and current practice in the development of web systems is the **automation of functional tests**, using tools to simulate the user's actions, like Selenium (seleniumhq.org) and WatiN (watin.org), just to mention some of the most popular open source projects. This kind of tools offers the possibility to follow a *record-and-playback* approach. Basically, it is necessary to manually execute the test case while the tool captures every action performed by the user on the browser. Then, the tool stores the actions in a file with a specific format or language (the *test script*) that the same tool can reproduce later. The same test case can be executed as many times as needed, performing the defined validations. Every command of the test script simulates a user action. These commands have as parameters the HTML element on which the captured action has been executed (for example, the input entered in a form), and the values entered. Fig. 1 shows an excerpt of a Selenium test script that accesses to an application (1st line), clicks the "Search" link (2nd line), enters the "computer" value in the HTML field "vSEARCHQUERY" (3rd line), and finishes by clicking the button with name "BUTTON1" (4th line).

Command	Target	Value
open	/sampleApplication/home.aspx	
clickAndWait	link=Search	
type	id=vSEARCHQUERY	computer
click	name=BUTTON1	

Fig. 1. Selenium script for functional testing.

This schema, in most of the tools, can be complemented with a *data-driven testing* approach, by which the test case takes test data from a separated source as a text file or a database (called data pool). Therefore, the same script can be reproduced with different data sets, testing in that way different cases with just little extra effort: adding lines to the test data source.

A **performance test** is defined as a technical research to determine or validate the velocity, scalability and stability characteristics of a SUT, in order to analyze its performance under load conditions [2]. Performance tests are useful to reduce risks

towards the *going live* day, analyzing and improving the performance of the application and the different servers when they are exposed to the concurrent users [3, 4]. There are specific tools to do that, called *load generators* or *load testing tools*, simulating concurrent users accessing to the system. Two of the most popular *open source* load generators are OpenSTA (opensta.org) and JMeter (jmeter.apache.org).

Unlike the functional test scripts, in the performance test scripts, even though the *record and playback* approach is used, the tools do not record at a graphic user interface level, instead, they do it at a communication protocol level. This happens because a functional test script reproduces the user actions on a real browser, whilst load generators try to "save" resources doing the simulation at a protocol level, as, for the HTTP protocol, the tool will execute hundreds of processes that just send and receive text through a network connection, with no necessity of showing graphic elements or any other task that requires major resource consumption.

```
POST URI "http://localhost/sampleapplication/search.aspx HTTP/1.1" ON 1 &
    HEADER DEFAULT_HEADERS   &
    ,WITH {"Accept: */*", &
        "Accept-Language: en-US", &
        "Referer: http://localhost/sampleapplication/search.aspx", &
        "Cookie: "+S cookie_1_0+"; "+S cookie_1_1}       &
,BODY "vSEARCHQUERY=computer&BUTTON1=Search&GXState=%7B%22_EventName%" &
    "22%3A%22E~<27>SEARCH~<27>.%22%2C%22_EventGridId%22%3A45%2C%22_Ev" &
```

Fig. 2. OpenSTA Script for performance test.

The performance test script contains a sequence of commands that manage HTTP requests and responses according to the protocol. This script is much more complex than the equivalent functional test script. For example, for the same actions presented in Fig. 1, where the script has only four lines of Selenium code, the equivalent performance test script has 848 lines using OpenSTA. That corresponds to each HTTP requests sent to the server, considering that each request triggers a sequence of secondary requests, which correspond to images included in the webpage, CSS files, Javascripts, etc. Each request (primary or secondary) is composed by a header and a body, as shown in the example in Fig. 2. Embedded, there are parameters, cookies, session variables, and any kind of elements used in the communication with the server. The example in this figure corresponds to the primary HTTP request of the last step of the test case; so, it includes the value "computer" in the parameter "vSEARCHQUERY" (the box).

Once the script is recorded, a number of adjustments must be performed in order to make it completely reproducible and representative of a real user. These scripts will be executed by concurrent processes (known as *virtual users*), so that it does not make any sense to execute them with the same user name and password to get connected to the system, or that all of them to use the same search key (because in these kind of cases the system will work faster thanks to the caches, at a database level and a web server level, obtaining misleading results). The cost of this kind of adjustments depends on the automation tool and on the SUT. In most of the cases, it is necessary to adjust the management of cookies and session variables (because in dynamic systems, the values cannot be simply used as they were obtained during the recording, because they should be unique or according to other restrictions). Adjustment of the parameters in the header and in the body will also be required.

From our experience in performance testing for more than 8 years, the scripting phase takes between 30% and 50% of the total effort of a performance testing project. On the other hand, the maintenance of these scripts (when the SUT changes) tends to be so complex that it is better to rebuild a script from scratch instead of trying to adjust it. Because of that, the process becomes pragmatically inflexible. The test generally will identify improvement opportunities, which imply modification on the system; however, our scripts will stop working if we change the system. How can we verify that the changes take a positive effect?

3 Automatic Generation of Performance Tests Scripts

The methodology proposed extends only the automation phase of the process presented in Vázquez et al. [3]. Instead of building the performance test scripts from scratch, the user has to provide a set of functional test scripts. As shown in **Error! Reference source not found.**, the tool will build a model of the HTTP traffic captured from the execution of each of these scripts. The model is the entry of the tool that generates the script code for the preferred load testing tool.

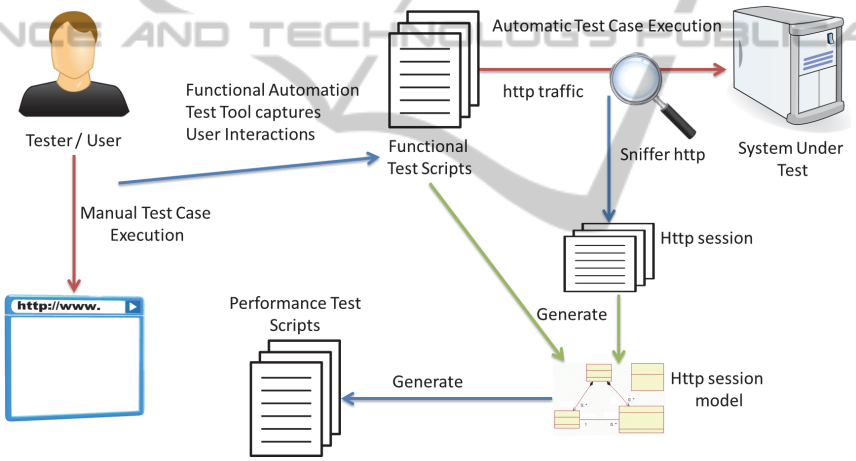

Fig. 3. Scripts generation proposal for performance tests.

The tool executes Selenium and WatiN scripts, but it can be easily extended for more automated testing tools. During the execution of the functional test scripts, it captures the HTTP traffic between the browser and the SUT with an HTTP sniffer (a tool capable to capture network traffic) called Fiddler (fiddler2.com). With this information it builds a model that is used to generate the scripts for OpenSTA. Also, it is easily extensible to generate scripts for different performance testing tools.

Fig. shows the main elements of the HTTP traffic model, which is useful to generate the performance test scripts. This model is built using the information obtained by the sniffer (all the HTTP requests and responses) and by the functional test script, correlating the user actions with the corresponding HTTP traffic. It is therefore composed of an ordered sequence of actions, including invocations to the application through HTTP (*requests*), or *validations* on the response to verify that it is

as expected. Each HTTP request is composed of a *header* and a message *body*. Both parts of the message are composed of parameters with their corresponding values. The header also has a set of fields that include, among others, cookies and session data. Each value can be hardcoded or can be taken from a data pool. It is important to keep the references between each HTTP request and its response, and with the corresponding functional test script command that generated it.

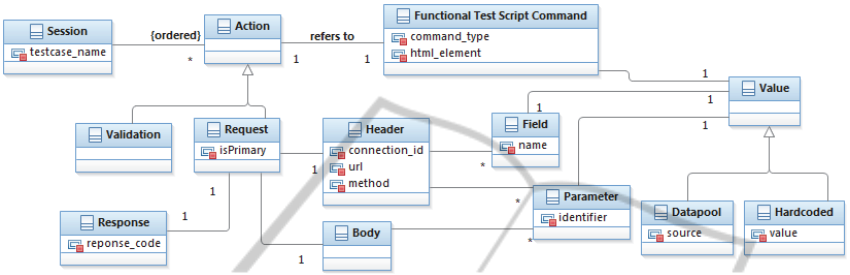

Fig. 4. HTTP session metamodel.

This model is used to generate code according to the language provided by the load generation tool. The generated code is specifically for OpenSTA. To perform this code generation the tool has an approach similar to the one proposed in model-driven environments for the model-to-text transformations [5], where the code generation is defined with code templates for each element of the model. Table 1 includes some examples of those templates; the first one is for the general structure of the script, used for each test case of the model, and the second one corresponds to an HTTP request, according to the specification of the HTTP protocol.

Table 1. Templates for scripts generation.

```
[template public generateScript(s: Session)]
[file (s.testcase_name().concat('.scl'), false, 'UTF-8')]
Definitions
        Timer T_TestCase_[s.testcase_name/]
        [s.variableDeclarations()/]
        CONSTANT  DEFAULT_HEADERS = "Host:  [s.getBaseURL()/]
        User-Agent: Mozilla/4.0"
Code
        Entry USER_AGENT,USE_PAGE_TIMERS
        Start Timer T_TestCase_[s.testcase_name/]
        [s.processActions()/]
        End Timer T_TestCase_[s.testcase_name/]
Exit
[/file]
[/template]
```

```
[template public processRequest(r: Request)]
Start Timer [r.name/]
[if ([r.isPrimary/])]PRIMARY [/if] [r.header.method/] URI [r.header.url/]
HTTP/1.1" ON [r.header.connection_id/] &
        HEADER DEFAULT_HEADERS, WITH [r.header.processFields()/]}
        [r.processBody()/]
[r.response.processLoadCookies()/]
End Timer [r.name/]
[/template]
```

As mentioned, after the recording the resulting script must be adjusted. Many of them are very repetitive tasks. Our tool makes this kind of things automatically, using the templates mechanism. Some of them are:

- Adding *timers* to each user action in order to measure the response time when executing the test scenarios, considering the kind of actions performed in the functional test script and the corresponding HTTP requests for each one.
- Taking advantage of different design aspects of the functional test script, in the performance test scripts: (1) the data are taken from the same data pools; (2) the same validations are performed; (3) same structure and modularization in different files promoting the readability of the test script.

By this way we get scripts even better than when recording them with the OpenSTA recorder. The more we use the tool, the more improvements and automatic adjustments we add to the scripts, avoiding that the tester commits mistakes during the preparation of the performance test.

Once the scripts are finished, the effort can be invested in the most important part (and the most interesting and beneficial) of a performance testing project which is the execution of the load scenario and the system's behavior analysis.

4 First Experiences in the Industrial Usage of the Tool

The tool has been used in five different projects within the services offered by the Uruguayan company Abstracta, where the tool has been developed. There were two testers working in all the projects, both with high knowledge about Selenium and OpenSTA. The SUTs were web systems from different domains and on different platforms, and very good results were obtained in all of them. Table 2 shows the number of generated scripts for each project, and the amount of simulated virtual users concurrently accessing to the SUT.

Table 2. Use of the tool in performance testing projects.

Project	SUT	# Scripts	# VU
Human Resources System	AS400 database, Java Web system on Websphere	14	317
Production Management System	AS400 database, C# Web system on Internet Information Services	5	55
Courts Management System	Java Web system on Tomcat with Oracle database	5	144
Auction System	Java Web system on Tomcat with MySQL database	1	2000
Logisitcs System	Java Web system on Weblogic with Oracle database	9	117

It is important to highlight that there are cases with few scripts, like in the 4th row, where only one script was required. That was defined based on the statistical analysis about the normal use of the system, which revealed that the 80% of the load is generated only with few use cases, perhaps with different flows internally represented in each script. Also, each script was executed with big and varied data sets.

In some projects the SUT were developed with Model-driven Development tools (particularly with GeneXus: genexus.com) which generates code from models with high level of abstraction. This raises a special complication, because just small

modifications to the models could mean many modifications on the generated code and therefore on the HTTP traffic. The process was the same as in the rest of the systems that were tested: first it is necessary to adjust the functional test scripts to regenerate the performance test scripts with our tool. It is in this kind of systems, where the SUT suffers many modifications during the testing project, where our approach reports the best benefits, because it was necessary to regenerate the scripts several times, and this would have required a major effort if manually executed.

In one of the projects there were no previous functional test scripts, so it was necessary to automate functional test scripts to use the tool. These functional test scripts were developed by a user (without knowledge about regression testing) which is almost impossible with any load generator. Once the project ended, the testing team started to manage a regression testing environment, using the scripts that were developed in the performance test project. In a certain way, the performance quality control favored the functionality quality control.

To summarize, the case studies have shown promising results in the performance testing, demonstrating that it can be made in a more flexible way and with less effort, according with what the testers involved in the projects reported. These results are also in line with the ones reported in the case study of [6].

5 Related Work

There are tools that, in order to ease the construction and maintenance of the performance test scripts, work at a graphic user interface, using Selenium scripts to execute load tests. The limitation of this approach comes from the fact that using PCs is probably not enough to simulate the typical number of users of a load test. These tools typically execute the tests scenarios from the Cloud, or with huge infrastructures. Some examples are Scaleborn (www.scaleborn.com) and Test Maker (www.pushtotest.com). With our approach instead, the number of required machines is kept low, being in that way a cheaper alternative, and obtaining the same results.

As far as we know, there are few proposals to generate performance tests. Some propose to design models as the basis of the performance test scripts generation, as in the one published by García-Domínguez et al. [7], which points to performance testing of workflows systems invoking Web Services. There are also some proposals to use stereotyped UML models, such as [8–10], or even others that extend a UML design tool to generate a complete set of performance test artifacts from the modeled diagrams [11]. The main disadvantage of these proposals is that a big effort is required to design the input artifacts for the generator. Last but not least, we would like to highlight the article of de Sousa et al. [6] where an approach similar to ours is proposed, taking advantage of the functional test scripts to generate performance test scripts. We observe two important limitations with this approach: on the one hand, as they do not consider the HTTP traffic (they only use the functional test script as input), it is impossible to generate the secondary requests and the primary requests coming from redirects that the SUT is doing, and on the other hand, it is not considering any javascript modification on the requests; therefore, the resulting simulation is not faithful to the real users load.

6 Conclusions and Future Work

Performance testing is needed to reduce risk in the *going live* process of any system, but, as it is expensive and resource demanding, it is typically made in a poor or incomplete way, or the results come too late. The most demanding task is the automation of the functionalities to be tested, taking part of the time that could be used to execute tests and analyze how to improve the system.

Taking this into account, this article presented a tool to generate performance test scripts in a cheaper way, taking advantage of the functional test scripts. This not only gives major flexibility when adjusting test scripts according to the changes and improvements performed on the application (that are always performed during any performance testing project), but it also helps generating better scripts, with better quality, in less time and with less effort.

The tool from this approach has being used in different projects to test the performance of a variety of systems, demonstrating the benefits of the proposal.

We plan to extend the performance test script generation to different load generators, like JMeter, which supports different communication protocols, allowing the execution of tests against systems that are accessed by different interfaces (HTTP, SOAP, FTP), and managing the test centralized in one single tool.

Acknowledgements

This work has been partially funded by Agencia Nacional de Investigación e Innovación (ANII, Uruguay) and by the GEODAS-BC project (TIN2012-37493-C03-01). We would also like to express our special acknowledgement to Abstracta team.

References

1. Graham, D., Fewster, M.: Experiences of Test Automation: Case Studies of Software Test Automation. Addison-Wesley Professional (2012).
2. Meier, J., Farre, C., Bansode, P., Barber, S., Rea, D.: Performance testing guidance for web applications: patterns & practices. Microsoft Press (2007).
3. Vázquez, G., Reina, M., Toledo, F., de Uvarow, S., Greisin, E., López, H.: Metodología de Pruebas de Performance. Presented at the JCC (2008).
4. Barber, S.: User Experience, not Metrics. (2001).
5. OMG: MOF Model to Text Transformation Language (MOFM2T), 1.0. (2008).
6. Santos, I. de S., Santos, A.R., Neto, P. de A. dos S.: Reusing Functional Testing in order to Decrease Performance and Stress Testing Costs. SEKE. pp. 470–474 (2011).
7. García-Domínguez, A., Medina-Bulo, I., Marcos-Bárcena, M.: Performance Test Case Generation for Java and WSDL-based Web Services from MARTE. Advances in Internet Technology. 5, 173–185 (2012).
8. Garousi, V., Briand, L.C., Labiche, Y.: Traffic-aware stress testing of distributed systems based on UML models. ICSE. pp. 391–400. ACM, New York, NY, USA (2006).
9. Shams, M., Krishnamurthy, D., Far, B.: A model-based approach for testing the performance of web applications. Presented at the SOQUA (2006).

10. Da Silveira, M., Rodrigues, E., Zorzo, A., Costa, L., Vieira, H., Oliveira, F.: Generation of Scripts for Performance Testing Based on UML Models. SEKE. pp. 258–263 (2011).
11. Cai, Y., Grundy, J., Hosking, J.: Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool. Presented at the ASE (2004).