Test automation on BPMS applications: an Experience Report

Jssica Lasch de Moura and Andrea Schwertner Charo

Ncleo de Cincia da Computao
Universidade Federal de Santa Maria – UFSM

Abstract. This article describes an experience of test automation of an application developed with the support of Business Process Management Systems - BPMS. For this purpose, we have implemented the same process using two different BPMS: Bonita and Activiti. We submit the resulting Web applications to two types of tests (load tests and functional tests), using test tools Apache JMeter, Selenium and Cucumber. The results show the feasibility and limitations of test automation for this type of application.

1 Introduction

Business Process Management has aroused the interest of companies and the scientific community, both for their benefits as per their challenges. Is called of BPM a set of concepts, methods and techniques to support the modeling, administration, configuration and analysis of business processes [1]. Associated with this, emerged the BPM systems (*Business Process Management Systems* – BPMS), which are software tools to support the life cycle of business process management.

Among the many BPMS currently available, it is common to find tools that support modeling, configuration and execution of business processes. In many cases, BPMS shorten the software development, delivering complete Web applications for process execution, using current technologies and requiring little code writing. On the other hand, some tasks such as checking and tests are still considered a challenge in this area [2]. In particular, the automated test of BPMS applications is rarely addressed by both the BPM community [1] as software testing area [3].

However, the lack of automation in testing can lead to problems during implementation and execution of business processes, especially when treating processes with many tasks and workflows, which easily lead to combinatorial explosion.

The purpose of this study was to explore solutions for automated testing of a process implemented in BPMS. For this, it started with a real application where manual testing proved inadequate [4]. In this article, we report the experience with automated testing this application, using open source tools. The application is presented in section 3, after a discussion of BPM and tests in section 2. Further , the section 4 presents the methods, tools and results of each type of test. Finally, the section 5 summarizes the lessons learned.

2 BPM and Testing

The BPM term can be used with different emphases, sometimes focusing on technology (software) and other times management. Still, the area has converged in understanding the lifecycle of BPM applications, involving the analysis of activities, modeling, execution, monitoring and optimization [5]. There is also convergence on the pattern BPMN (*Business Process Model and Notation*) to express the process modeling.

BPM systems (BPMS) has been claimed as essential tools to support the activities of this life cycle. Currently, it can be said that a typical BPMS provides resources for definition and process modeling in BPMN, execution control and activity monitoring of processes [6]. There is a tendency of BPMS in shortening software development, for example via web forms generators associated with process tasks [7]. Note, however, that the concern with testing is not evident in the BPMS tools. Indeed, examining the promotional material and documentation available on the main BPMS, there is an emphasis on modeling and execution stages.

Moreover, the importance of the tests is widely recognized in software engineering [8]. Whereas BPMS applications are usually Web-based systems, it can be assumed that it can be successfully tested using dedicated approaches, such as load tests or functional testing of the black-box type. There are also those who argue that BPM applications test differs from traditional Web applications testing [9], but there were no more references to deepen this view. This finding reinforced the motivation for this work.

3 Test Target Application

The target application of this work refers to a common process in higher education institutions: the appreciation of complementary activities, ie, activities that form the flexible part of the undergraduate curriculum (participation in lectures, events, projects , etc.). In a previous work [4], presented the modeling, implementation and deployment of this process. Their representation in BPMN, in Figure 1, reveals a total of 11 tasks span responsibility 5 divisions.

In that previous work, we implemented the process with the Bonita BPM tool [1] in version 5.7.2, an open source BPMS recognized in the corporate world [6]. The resulting application has several Web forms for each division of responsibility, being the first one for the data population by the student. According to the type of complementary activities, the flow is directed to those responsible for the assessment and validation of activity. Note, in Figure 1, the process has several gateways, which leads to over 15 possible different paths in the process.

The application was subjected to functional testing performed manually, in addition to acceptance testing with a group of real users. However, a few weeks in operation, problems emerge: Process instances failed due to unexpected inputs, services were not restored properly after being interrupted and there was

[1] Bonita BPM (formerly Bonita Open Solution). Available at: www.bonitasoft.com

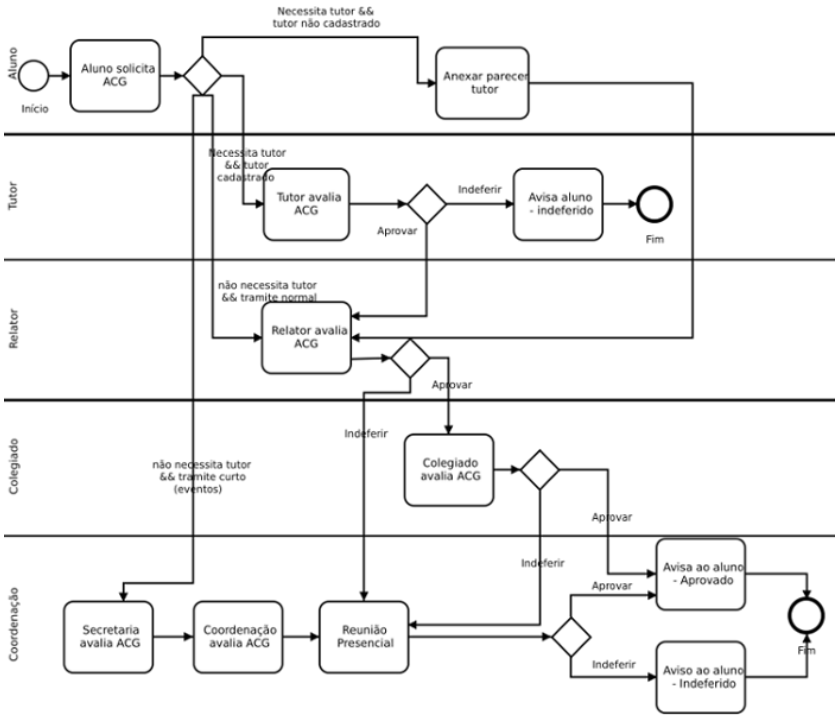

Fig. 1. Diagrama do processo em BPMN

overloaded because of the number of cases opened on a deadline. This experience led to the search for solutions for test automation.

It was found, however, that the BPMS used does not support any type of automated testing. We attempted to other BPMS with open source or freeware licenses, which could implement the process in question and offered testing support. The preference for this type of license was due to its flexibility and affordability to the "client" this project: a public educational institution. Among the analyzed tools (TIBCO[2], Activiti[3], Process Maker[4], Intalio[5]), offered no clear support for automated testing. Just Activiti tool cited the possibility of allies unit tests to JUnit, but not much information about this alternative. So we went to another option: use test tools external to BPMS.

In addition, it was decided to deploy the same application using other BPMS in order to broaden the experience and possibly identify similarities and differences in automated test implementations with different BPMS. Yet it had intended to elect the best BPMS, but to assess support for both tests.

[2] TIBCO. Available at: www.tibco.com
[3] Activiti. Available at: www.activiti.org
[4] Process Maker. Available at: www.processmaker.com
[5] Intalio. Available at: www.intalio.com

The tool chosen was Activiti, a BPMS based on Java technologies, like Bonita, allowing you to work with the same server-side technologies (Tomcat and MySQL, in this case). At the time, the latest version of the application was 5.14. This BPMS also works with the same version of the BPMN notation used in Bonita, allowing import in full the process originally created. Web forms created with Bonita's help, however, could not be imported and had to be recreated with Activiti.

4 Description and Execution of tests

In planning automated tests, priority was given to the test of aspects that in fact revealed problems during deployment, in previous work. The selected tests were: (a) functional tests to verify the produced system output from pre-defined inputs and, (b) load tests, which are one type of performance testing, to evaluate the opposite behavior of the system to a large number of requests. None of these types of tests is supported in BPMS Bonita and Activiti, which include only limited functionality simulation and debugging process execution. Although Activiti quoting support unit tests not be exploited for this option to be understood that priority would be less compared to the observed problems.

Thus, there was a survey of tools intended for Web application testing and was selected the most promising judged before departing to the detailing and execution of tests. The scripts that configure the tests are available for consultation in http://www.inf.ufsm.br/ andrea/bpmtest-scripts.zip.

4.1 Functional Tests

To perform functional tests on Web applications, you can use free tools like Selenium[6], Watir[7] or Geb[8].For this work, we picked up the Selenium tool, combined with Cucumber-JVM [9] for a description of the tests.

With these tools, the process for running a functional test consists of: capture the user interaction with the browser (Selenium IDE), export the generated code (Selenium IDE), creation of test scenario (Cucumber), creation of definitions of test steps (Cucumber), creation of methods for each step (Java) and test execution (Selenium WebDriver). The test scenario is a definition, in order of execution of steps that run the application, and the expected results. For this work, was set up a scenario in which the student logs in and fills 2 forms as a first task of the process. For this scenario, methods were created by varying the entries on the forms. As the tested process is the same in both BPMS, the test scenario is also the same.

For applications of both BPMS, initially errors occurred in the tests relating to the location fields in web forms. It was found that the fields were located in

[6] Selenium. Available at: www.seleniumhq.org.
[7] Watir. Available at: www.watir.com.
[8] Geb. Available at: www.gebish.org.
[9] Cucumber-JVM. Available at: www.github.com/cucumber/cucumber-jvm.

a *iframe*, while the capture of the interaction to occur without problems, the generated code did not select the *iframe* and so could not find the fields. Thus, it was necessary to use a Selenium method to access the *iframe* before selecting the desired element.

In application with Activiti also ran into other trouble. Unlike what happened with the Web forms generated by Bonita, the Selenium IDE did not capture all user interaction with the application. Indeed, in the login stage, Selenium just captured the access page and the " click " to the login button, ie not captured filling the fields " User " and " password ". This problem was repeated with some other elements of the Web form when recording the interaction. It is believed that the problem occurs due to the structure of the Web page, which can contain elements that Selenium does not automatically identify, such as *divs*, *frame* and *scripts*, for example. However, this does not preclude the creation and execution of tests. To work around the problem, it was necessary to study the structure of the Web pages in question, locate the missing elements and then add the code to access them in their methods.

Resultados dos Testes Funcionais The functional tests were more viable than load tests, because a good part of the interaction is performed on the client side without the need to explicitly handle the interactions with the server. It can be said that this test reached all its goals because it allowed reproduce user interaction and create the code to test applications with different inputs in a scenario involving the initial task of the process, expanding the coverage of the tests. Functional testing also reproduced errors found in production and that had gone unnoticed in manual tests (not properly provided for in the forms entries), these errors were the same in both BPMS tools because the forms were configured equivalently.

Functional testing was also less dependent on BPMN. The Table 1 presents a summary of the main similarities and differences found. In both cases, it took a few modifications to the code generated by the Selenium and Cucumber simply by inspecting the structure of Web pages. Cucumber makes the implementation of testing faster and less labor than if it were used only Selenium, shortening the code generation aligned with the test scenarios. Still, if necessary to extend the test to many tasks of a process, changes in the test code may become laborious.

4.2 Load Tests

Load tests on web applications is typically performed by generating multiple HTTP requests to the server in a controlled manner. For this, a critical step is to identify the requests that should be played. There are several tools that purport to facilitate this type of testing, among which we can mention: JMeter[10], The Grinder[11] and WebLOAD[12]. All tools are open source and have several options,

[10] Apache JMeter. . Available at: www.jmeter.apache.org
[11] The Grinder. Available at: www.grinder.sourceforge.net/
[12] WebLOAD. Available at: www.radview.com/webload-download/

	Bonita 5.7.2	Bonita 7.1.2	Activiti 5.14	Activiti 5.19
Web Technologies	HTML, CSS, Ajax	HTML, CSS, Ajax	HTML, CSS, Ajax	HTML, CSS, Ajax
Capture of user interaction using Selenium	Total	Partial (required manual insertion of some fields)	Partial (required manual insertion of some fields)	Partial (required manual insertion of some fields)
It was possible to export the code generated by Selenium?	Yes	Yes	Yes	Yes
Recognition of all fields captured WITHOUT code change	Partial	Partial	Partial	Partial
Recognition of all fields captured WITH code change	Total	Total	Total	Total

Table 1. Comparative overview of the functional test

but chose to JMeter tool for its functionality "proxy server" which is a great help to the capture of requests.

An application test with JMeter consists of four steps: capture HTTP requests, export requests (.jrxml format for JMeter), set up the test plan and, finally, run the test in JMeter. The application generates different HTTP requests that need to be identified and interpreted to be played automatically. The use of Web technologies with asynchronous processing, client-side, can hinder this step because a user action can not immediately generate a request to the server. Moreover, in applications BPMS, many users act on different tasks in a same process, so that the HTTP requests that carry user identifying keys and processes.

In the case of application created with Bonita in the version 5.7.2, it was found that there is a session identifier key that is generated at the time user accesses the system and other instance identifying key, that is, identifies each execution of the process as only being created by the server when the user initiates the process. To run the tests, so it was necessary to locate the request in which these keys are generated and use the "Regular Expression Extractor" tool of JMeter to get their values. In the application created with Activiti in the version 5.14, it was impossible to identify the request in which the keys are generated, as there is not a request whose return (server response) contains key values. This suggests that the generation of the identifying key is made internally by the BPMS, ie not in an HTTP request and therefore this can not be captured and reproduced in JMeter.

To increase the experience and see if some problems are repeated in other versions, it was decided to perform the load test with the currently most recent issue of BPMS tools. It began with the tests in version 7.1.2 of Bonita, cap-

ture and analysis of requests was executed in the same way the previous tests. However, even replacing the identifier keys tests were not successfully executed. Some steps that should be reproduced were not performed, for example, start a process or start a task. This leads to believe that some steps are not performed in a way that can be captured via an HTTP request, problem similar to what happened in test with the version 5.14 of BPMS Activiti. When performing the load test with the 5.19 version of Activiti occurred the same problems seen in version 5.14.

As the functional tests with Selenium were successful executed in all tested versions of both tools, unlike load test with JMeter, it was decided to figure out a way to perform load testing through the information obtained from the Selenium. Was chose to use the TestNG [13] tool, which is a framework commonly used to automate testing with Selenium [10]. The TestNG has a notation that allow you to set the number of times the test must be performed and the number of threads, in that way is possible to simulate the execution of several threads simultaneously. Load testing using Selenium and TestNG consist primarily of the code that simulates the user running with the application, similar to that used in functional testing, and annotation in TestNG format.

By default, each test execution with Selenium opens a new window in the selected browser (in this case, Mozilla Firefox [14]), the which can hinder test execution with a large number of virtual users. So in addition to using codes generated with Selenium and TestNG tool to perform the tests, was used a Virtual Frame Buffer [15] to allow the browser to perform the test without opening a new window.

It was possible to run the load test using the code obtained in Selenium. With this test, we were able to analyze the performance of all versions of the analyzed BPMS tools. In Table 2 may have seen an overview about which tests could be performed.

BPMS and version	Load Test with JMeter	Load Teste with Selenium and TestNG
Bonita 5.7.2	Able to run the test	Able to run the test
Bonita 7.1.2	Unable to run the test	Able to run the test
Activiti 5.14	Unable to run the test	Able to run the test
Activiti 5.19	Unable to run the test	Able to run the test

Table 2. Overview of the load tests

[13] TestNG. Available at: www.testng.org

[14] Mozilla Firefox Available at: www.mozilla.org/pt-BR/firefox/

[15] XVFB - X Virtual Frame Buffer. Available at: http://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml.

Load Test Results Due to problems reported in the previous section, load tests with JMeter could only be achieved in the implementation created with Bonita. In order to test the system behavior with different load levels, tests were performed with 1, 50, 100 and 200 virtual users and requests were analyzed regarding the essential steps to begin the process: log in, view the Bonita's homepage, select the process, show the initial form and send the completed form. The application is hosted on a SGI Altix XE 210 server with 24GB of RAM, accessible by a Fast Ethernet network.

The response times of each stage, depending on the number of users, can be seen in Table 3. The most alarming results are for 200 virtual users, where the average response time on the login request was for 10,149 ms, or approximately 10 seconds, which is a high response time. The average response time for all requests was 3111 ms (i.e., 3 s), and the standard deviation was 13,088. Besides high response times, the test showed virtual users 200 with error rates in some requests which were not found with a smaller number of users. For example, a request that has performed login showed a rate of 2% error and, in total, the requests have obtained an error rate of 7.82%. The error occurred because of unanswered requests.

Users	Login	Home Page	Process selection	First form	Send form
1	126	32	38	80	73
50	597	191	179	368	152
100	1972	571	552	760	694
200	10.149	3.239	934	2.122	1.918

Table 3. Average response times, in milliseconds

Overall, therefore, the load testing with JMeter achieved its objectives and helps explain the overload that occurred with the application in production, when many students tried to access the initial form in a deadline. However, it is important to note that this test was performed only in the early stages of the process and, even then, was already laborious and consumed a few hours of preparation, because it requires a deep analysis of HTTP requests to run the tests successfully. In all, about 100 requests were captured only in these steps, therefore it is estimated that the testing of a task towards the end of the process can become unwieldy with JMeter, it demands the identification and interpretation of many requests. Another important observation in this experiment is that this testing approach suffers from dependence on Web technologies used by BPMS.

On the other hand, in the load test performed with Selenium and TestNG all versions of the tools could be successfully tested. Because it is a load test conducted in a different way the JMeter, without HTTP requests, could not be analyzed on isolated results of each request. The result of TestNG test shows the average test runtime in the execution of the first task of the process. In the

Table 4 can be seen a summary of the average response time for each version of the tools for each amount of virtual users/threads.

Users	Bonita 5.7.2	Bonita 7.1.2	Activiti 5.14	Activiti 5.19
1	xx	xx	xx	xx
50	xx	xx	xx	xx
100	xx	xx	xx	xx
200	xx	xx	xx	xx

Table 4. Average response times, in milliseconds

5 Consideraes Finais

In this paper we explored automated test solutions at an application of BPMS. In the absence of supporting the load and functional tests in Bonita BPMS and Activiti was applied test tools intended to Web applications in general.

Regarding the load test, it was useful to explain failures observed in previous work. Also proved to be a laborious test, or even impossible, depending on the BPMS used. Experience with two BPMS strengthened this conclusion because there were different situations, with Bonita the test was successful, but with Activiti the test could not be executed, since it failed to reproduce all requests.

Functional test, with the approach taken, it obtained greater success in the execution of tests and was observed less reliance on BPMS, compared to the previous test. The test task may prove to be cumbersome, especially when you want to test many tasks and flows that a business process can have.

Overall, how lessons learned have that under certain conditions it is feasible to test application of BPMS with testing tools for Web systems. The main condition in the case considered, was the targeting of testing an initial task of the process, identified as neck. As the time and effort to a single task was significant, this approach can become unwieldy if necessary extend the tests to many tasks of a process.

Another aspect to consider is that when the BPMS implements the interaction with the user and/or servers, the developer do not to choose and control all technologies used. This facility, however, may make it difficult to test automation with external tools, that benefit from knowledge about the implementation (eg, use of *iframes*, Ajax, in the experience in question).

Although the experience of this work was not exhaustive, it might be noted that support automated testing is underexplored in BPMS. It is understood that this would be a welcome feature, assuming that shorten an essential step to ensure the quality of the resulting software.

References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, 2nd edition, (2012)
2. van der Aalst, W. M. P.: Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013(507984)
3. Graham, D. and Fewster, M.: Experiences of Test Automation: Case Studies of Software Test Automation. Addison-Wesley (2012)
4. de Moura, J. L. et al.: Gesto de processos de negcio em curso de sistemas de informao: Relato de experincia utilizando software livre. In *IX Simpsio Brasileiro de Sistemas de Informao*, pp. 206–217, (2013)
5. Guide to the Business Process Management Body of Knowledge (BPM CBOK). Association of Business Process Management Professionals, 2nd edition (2012)
6. Forrester Research.: The forrester wave: BPM suites, Q1 (2013)
7. Winter Green Research.: Business process management (BPM) cloud, mobile, and patterns: Market shares, strategies, and forecasts, worldwide, 2013 to 2019.
8. Bourque, P. e Fairley, R. E.: Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 3.0. IEEE Computer Society (2014)
9. Chetty, N. K.: How to perform workflow testing for BPM applications (2014), `http://www.evoketechnologies.com`
10. Bindal, P. and Gupta, S.: Test Automation Selenium WebDriver using TestNG. In *Journal of Engineering Computers & Applied Sciences*, pp. 18–40, (2014)