Test automation on BPMS applications: An Experience with two Open Source BPM Systems

Jéssica Lasch de Moura and Andrea Schwertner Charão

Computer Systems Laboratory - LSC
Federal University of 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 different versions of two open-source BPMS: Bonita and Activiti. We submit the resulting Web applications to two types of tests (load tests and functional tests), using four different test tools: Selenium, Cucumber, Apache JMeter and TestNG. The results show the feasibility and limitations of test automation for this type of application.

1 Introduction

Business Process Management (BPM) has aroused the interest of companies and the scientific community, both for their benefits and for their challenges. The term BPM comprises a set of concepts, methods and techniques to support the modelling, administration, configuration and analysis of business processes [2]. Associated with this definition, BPM systems (*Business Process Management Systems* – BPMS) emerged as 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 modelling, 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 verification and tests are still considered a challenge in this area [1]. In particular, the automated test of BPMS applications is little addressed by the BPM community [2] or the software testing community [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 life cycle of BPM applications, involving the analysis of activities, modelling, execution, monitoring and optimization [5]. There is also convergence on the pattern BPMN (*Business Process Model and Notation*) to express the process modelling.

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 modelling 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 performance of applications created is not evident in the BPMS tools. Indeed, examining the promotional material and documentation available on the main BPMS, there is an emphasis on modelling and execution stages.

There are some studies that assess/test compliance of models of processes [8] or evaluate tools based on their features [9] latter being directed to support organizations choosing the ideal BPMS.

A recent work by Ferme et al. [10] analyzes the performance, through a framework, of two Workflow Management Systems tools or *WfMSs*, and Workflow Management or *WfM* tools relate BPM tools so that BPM can be considered an extension of the classical WfM systems and approaches [11], is also mentioned the importance of evaluate systems, but in this case focusing on help in choosing the best tool.

Moreover, the importance of the tests is widely recognized in software engineering [12]. 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 [13], 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 in Brazil: the appreciation of complementary activities, ie, activities that form the flexible part of the undergraduate curriculum (participation in lectures, events, projects). Their representation in BPMN, in Figure 1, reveals a total of 11 tasks divided between 5 actors. The process is initiated

by a student wishing to have their activity analyzed in the task "Student request use of activity", from that, there are several possible flows depending on the type of activity selected by the student and at the end of the process the student's request can be approved or rejected. In a previous work [4], presented the modelling, implementation and deployment of this process.

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 where all the process begins. 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.

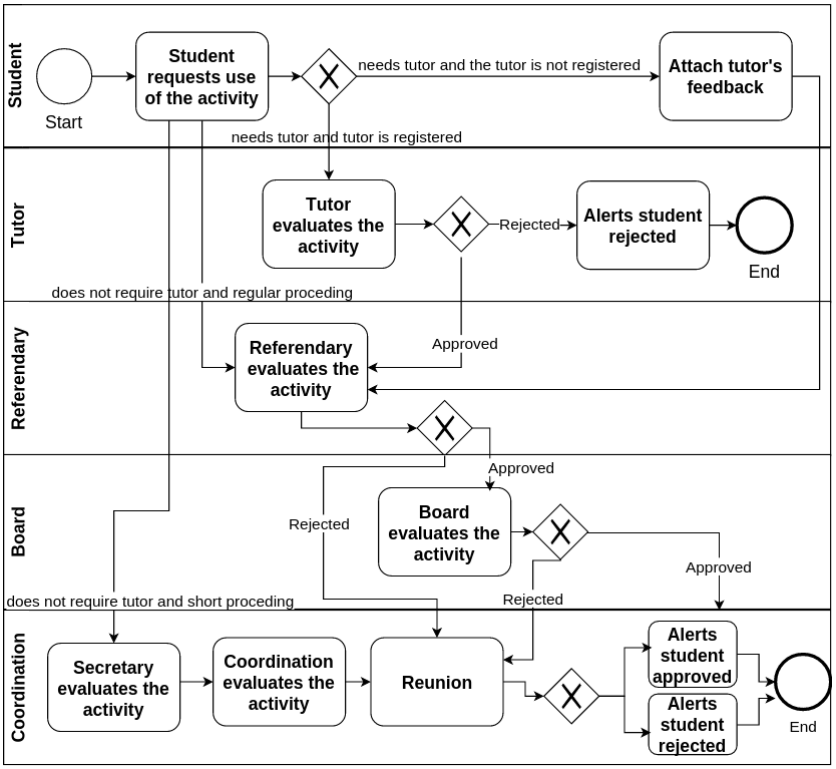

Fig. 1. BPMN process diagram

The application was subjected to functional testing performed manually, in addition to acceptance testing with a group of real users. However, a few weeks

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

in operation, problems emerge: Process instances failed due to unexpected inputs, services were not restored properly after being interrupted and there was overloaded because of the number of cases opened on a deadline. This experience led to the search for solutions for test automation.

We find out, however, that the BPMS used does not support any type of automated testing. We search 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" of this project: a public educational institution.

Among the analysed tools (TIBCO[2], Activiti[3], Process Maker[4], Intalio[5]), none offered clear support for automated testing. Just Activiti tool cited the possibility of allies unit tests to JUnit, but are not cited much information about this alternative. So we went to another option: use test tools external to BPMS.

In addition, we 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. But we do not want to elect the best BPMS, but to evaluate the support for both tests.

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). This BPMS also works with the same version of the BPMN notation used in Bonita, allowing import the full process originally created. Web forms created with Bonita's help, however, could not be imported and had to be recreated with Activiti. The tests were performed at two different times using the latest versions of each tool. Thus, were tested applications created with the 5.7.2 and 7.1.2 version of Bonita and version 5.14 and 5.19 of the 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 behavior of the system opposite to a large number of requests/access. 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 this option will not be exploited for to be understood that would be less priority compared to the observed problems.

Thus, we did a survey of tools intended for Web application testing and we selected the judged most promising before departing to the detailing and execution of 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

4.1 Functional Tests with Selenium and Cucumber

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 that is an integrated development environment for automated test scripts, it is implemented as a Firefox extension, and allows to record, edit, and debug tests. Also, combined with Selenium, we picked up the Cucumber-JVM[9] tool for a description of the tests, Cucumber is a tool that provides a high-level interface that allows to write tests in natural language. The choice was motivated by the large number of references to the Selenium [14, 15], confirmed by works which showed satisfactory results with Selenium and Cucumber [16, 17].

With these tools, the process for running a functional test consists of: capture the user interaction with the browser using Selenium, export the generated code created with Selenium, creation of test scenarios using Cucumber notation, creation of definitions of test steps (Cucumber/Java) in a file called *Step Definition*, creation of code for each step (Java) and test execution (Selenium WebDriver). A test scenario will be created in a "feature" file and is a flow of events through which the application that will be tested will pass, for each step of the scenario should also be created a step definition to be run on *StepDefinition* file. For example, in the case of the test combined with Selenium, if the step of the scenario is "Fill Form" the *StepDefinition* should will call the Selenium code responsible for performing the filling of the form.

In a previous work [18], versions 5.7.2 Bonita and 5.14 Activiti were tested. In these first tests was analysed only one task of the process, and because of this, were used a different scenario. As a way to broaden the experience also tests were then carried out with the latest versions of BPMS: 7.1.2 of Bonita and 5.19 of Activiti. Also the test scenario has been updated for the four different versions in order to be tested one more task.

In the Figure 2, can be viewed the scenarios used in the tests with the four versions. The test consists of two scenarios: in the first scenario the student logs in the application and fills the forms of the first task of the process (Students requests use of the activity) and in the second scenario another user logs in and runs the second task of the process available to him (Secretary evaluates the activity). For these scenarios we also used "Examples", which allows you to test the same scenario with a varied number of pre-defined entries increasing the test coverage, each list item "Examples" is a line in a XLSX file containing the entries that are tested in the execution. As the tested process is the same in all BPMS, the test scenario is also the same for all applications.

For applications created with both versions of each BPMS, initially errors occurred in the tests relating to the location of the fields in web forms. We found that the fields were located in a *iframe*, while the capture of the interaction to occur without problems, the generated code did not select the *iframe* and so

[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.

```
Feature: Testing BPM Process

Scenario Outline: Scenario 1
Given I start the process
When I select the Student requests use of the activity task
Then The form page title is expected
When I fill the forms of first task with excel row "<row_index>" of dataset
Then The Secretary evaluates the activity task is ready

Examples:
  |row_index|
  |1|
  |2|...

Scenario Outline: Scenario 2
Given I select the task The Secretary evaluates the activity available
When I fill the forms of The Secretary evaluates the activity task with excel row "<row_index>"
Then The next task is ready

Examples:
  |row_index|
  |1|
  |2|...
```

Fig. 2. Test Scenarios

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 version 5.14 of 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, i.e., not captured filling the fields " User " and " password ". This problem was repeated with some other elements of the Web form when recording the interaction. We believe that this 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.

This problem did not occur with the version 5.19 of Activiti as well as in the two versions of Bonita, the capture and execution of the elements was successful, it was only needed some minor changes in the generated code.

Functional Tests Results The functional tests proved to be viable, mainly because a good portion 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 two initial tasks 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 correctly expected inputs in forms), these errors occurred

in the two versions of both BPMS tools because forms and tested entries were configured equivalently.

As it was possible to realize the tests successfully on all the tools and code testing is similar (only required a few modifications), it is possible to say that the functional test is little dependent to the chosen BPMS. Table 1 presents a summary of the main similarities and differences found. In all cases, it took a few modifications to the code generated by the Selenium and Cucumber by simply inspecting the structure of web pages. Cucumber makes the implementation 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.

	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

4.2 Load Tests with Apache JMeter

Load tests on web applications are 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 reproduced. There are several tools that purport to facilitate this type of testing, among which we can mention: JMeter, The Grinder[10] and WebLOAD[11]. All tools are open source and have several options, but chose to JMeter tool for its functionality "proxy server" which is a great help to the capture of requests.

[10] The Grinder. Available at: www.grinder.sourceforge.net/
[11] WebLOAD. Available at: www.radview.com/webload-download/

In the load testing with JMeter it was evaluated the performance of the execution of the first task of the process as it can be considered a "bottleneck" because of the number of users who can start a process simultaneously.

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 BPMS applications, many users act on different tasks in a same process, so HTTP requests carry user identifying keys and processes.

In the case of application created with Bonita in the version 5.7.2, we found that there is a session identifier key that is generated at the time user accesses the system and a instance identifying key which identifies each execution of the process as only. Both keys are created by the application when the user starts a process. When running the test directly with the captured requests errors are returned due to non-existence of these keys, i.e., the BPMS not find in which instance/task requests are treated. This happens because the keys that were captured with the requests are no longer available in time of execution of the test. So, to run the tests, 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 keys is made internally by the BPMS, i.e., not in an HTTP request and therefore this can not be captured and reproduced in JMeter.

In the tests in version 7.1.2 of Bonita, capture and analysis of requests was executed in the same manner described above. 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, it was impossible to find where the identifying keys are generated.

Test results with Apache JMeter Due to problems reported in the previous section, load tests with JMeter could only be executed in the application created with the version 5.7.2 of Bonita. In order to test the system performance with different load levels, tests were performed with 1, 50, 100 and 200 virtual users and were analysed requests regarding the essential steps to begin the process: Login, view the Bonita's homepage, select the process, show the first form and send the completed form. The application is hosted on a Intel R2312 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 2. The most alarming results are for 200 virtual users, where the average response time on the login request was for 6.930 ms, or approximately 6 seconds, which is a high response time. The average response time for all requests in the test with 200 virtual users was 2.903 ms (i.e. 3 s). Besides high response times, the test with 200 virtual users showed error rates in some requests which were not found with a smaller number of users. For example, the request that performs login showed a error rate of 1.99% and, in total, the requests have obtained an error rate of 6.98%. The errors occurred because of unanswered requests.

Users	Login	Home Page	Process selection	First form	Send form
1	65	20	19	52	42
50	387	101	92	260	90
100	1072	490	473	601	564
200	6.930	2.345	876	1.611	1.087

Table 2. Average response times, in milliseconds

Overall, the load testing with JMeter helps explain the overload that occurred with the application in production, when many users tried to access the initial form in a deadline, mainly because the test with 200 users has created a charge that led some pages to stop responding. 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 (*iframes*, Ajax) used by BPMS, therefore, the four versions tested in only one of them it was possible to conduct the test successfully and this was mainly due to how the BPMS are implemented.

4.3 Load Tests with Selenium and TestNG

As the functional tests with Selenium were successful executed in all tested versions of both tools, unlike load test with JMeter, we decided to verify the existence of a way to perform some kind of load testing through the information obtained from the Selenium. This alternative also could be less dependent on Web technologies used by BPMS. In this test was evaluated the behavior only of the execution of the first task of the process. The application was hosted on the same server where the load tests with JMeter were performed.

To perform the load tests with Selenium were analyzed and tested different tools such as: Browser Mob[12], JMeter WebDriver Sampler[13], Selenium Grid[14] and TestNG[15]. Lastly, the tool that allows running tests completely was the TestNG and therefore we chose it to perform load testing with Selenium. TestNG is a framework commonly used to automate testing with Selenium [19]. The TestNG has a notation that allow to set the number of times the test must be performed and the number of threads, in that way is possible to execute 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 thread in the test execution with Selenium opens a new window in the selected browser (in this case, Mozilla Firefox [16]), i.e., 100 threads are equal 100 new browser windows. Also, each window open by the browser takes a while to be loaded which can increase the total time to run the test. This also can make it difficult to run a test with a large number of users due to very long time to run the test and can cause memory problems on the machine that runs the script.

To avoid this problem, in addition to using codes generated with Selenium and TestNG tool to perform the tests, we used a Virtual Frame Buffer [17] to allow the browser to perform the test without opening a new window for each thread.

Load testing using Selenium consists of basically four stages: capture user interaction with the application through Selenium, create the test method using the code generated from the Selenium and TestNG annotation, configure Firefox and run the test. Some changes may be required in the code generated by Selenium depending on which elements are used on the page.

For the tests, the code captured by Selenium functional test was reused and it was only necessary to insert the annotation of TestNG and the code of the Virtual Frame Buffer. A fragment of the code of load test simulating 200 virtual users in the first task of the process can be seen in Figure 3. In this code, the line represented by the number 1 is possible to see the TestNG annotation to run 200 threads, the beginning of the test method is in the line 2, in the lines 3 to 6 is performed the creation the Firefox browser driver using a Virtual Frame Buffer. Starting from the line 10 in the Figure 3 is the code related to simulate the user interaction with the application (obtained through Selenium). It was possible to run the load test using the code obtained in Selenium combined with TestNG. With this test, we were able to analyse the performance of two versions

[12] Browser Mob Proxy. Available at: www.bmp.lightbody.net
[13] JMeter WebDriver Sampler Plugin. Available at: www.jmeter-plugins.org/wiki/WebDriverSampler
[14] Selenium Grid. Available at: www.seleniumhq.org/projects/grid
[15] TestNG. Available at: www.testng.org
[16] Mozilla Firefox Available at: www.mozilla.org/pt-BR/firefox/
[17] XVFB - X Virtual Frame Buffer. Available at: http://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml.

```
1 @Test(invocationCount = 200, threadPoolSize = 200)
2 public void SeleniumMasterTest() {
3     String Xport = System.getProperty("lmportal.xvfb.id", ":10");
4     FirefoxBinary firefoxBinary = new FirefoxBinary();
5     firefoxBinary.setEnvironmentProperty("DISPLAY", Xport);
6     WebDriver driver = new FirefoxDriver(firefoxBinary, null);
7
8     baseUrl = "[redacted]";
9
10    driver.get(baseUrl + "/activiti-explorer/ui/");
11    driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
12    driver.switchTo().frame(driver.findElement(By.name("PID3")));
13
14    driver.findElement(By.name("username")).clear();
15    driver.findElement(By.name("username")).sendKeys("[redacted]");
16    driver.findElement(By.name("password")).clear();
17    driver.findElement(By.name("password")).sendKeys("[redacted]");
18    driver.findElement(By.cssSelector("span")).click();
19    driver.manage().window().maximize();
20
21    driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
22    driver.switchTo().defaultContent();
```

Fig. 3. Selenium Load Test Source Code Fragment

of the both BPMS tools. In Table 3 is possible to see an overview about which load tests could be performed compared to tests performed with JMeter.

BPMS and version	Load Test with JMeter	Load Test 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 3. Overview of the load tests

Test results with Selenium and TestNG In the load test performed with Selenium and TestNG all versions of the tools could be successfully tested. These tests were also performed simulating 1, 50, 100 and 200 virtual users/threads but, because it is a load test conducted in a different way of JMeter (without HTTP requests) could not be analyzed on isolated results of each request or the response time. At the end of the test run, the TestNG generate reports that display the time that each execution (each thread) took to complete and total test execution time.

In the execution time is included the time of the entire interaction with the application, since this is effectively executed by each thread, different from test

with JMeter where the requests are simulated and the response time is measured. For example, the average execution time of each thread in all of the tools tested and for any amount of threads was the same: about 14s. Thus, it is impossible to compare the runtime with the response time obtained in JMeter because it reflects the time that the script itself takes to be processed.

However, it was possible to analyse the application behavior (not performance) front a large number of simultaneous accesses from other types of information. In tests with 200 threads, for example, some threads have failed due to pages that have stopped responding. The server logs were also observed some errors in the lack of response. This behavior is similar to what happened with testing with JMeter. In the Table 4 can be seen in the summary of application behavior for each tool and for each number of virtual users/threads.

Users	Bonita 5.7.2	Bonita 7.1.2	Activiti 5.14	Activiti 5.19
1	No failure	No failure	No failure	No failure
50	No failure	No failure	No failure	No failure
100	Some failure	No failure	No failure	No failure
200	Some failure	Some failure	Some failure	Some failure

Table 4. Summary of application behavior

Thus, it can be said that the load test with TestNG achieved its objectives because it was useful to simulate a large number of users performing actions in the system simultaneously and experienced similar behavior to what we observed in tests with JMeter. Regarding the load test with JMeter testing using Selenium and TestNG have some disadvantages: the cost to run the test is higher because each thread is actually performed on the machine that runs the tests, this cost is somewhat mitigated by the use of a Virtual Frame Buffer; the test takes longer to run; it is not possible to measure the performance/response time (can only be analyzed the behavior through of server information).

On the other hand, the approach using TestNG and Selenium has the advantage of being possible to carry out tests on all the tools analyzed, showing that it is not dependent on the tool structure, different from what we observed in the test with JMeter. This approach can be useful when used BPMS which have much client-side processing and use technologies such as Ajax, for example, which can not be correctly processed by the requests captured with JMeter.

Also, as observed in other tests, the task of running the tests can become cumbersome if you need to test many tasks or many process flows. This approach also has the advantage of being able to reuse code created in the functional test, which allows shorten test creation.

5 Final Considerations

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

In the Functional test, with the approach taken, it obtained greater success in the execution of tests and low dependence was observed in relation to the BPMS because only a few page structures must be observed (names, iframes, etc). Two initial tasks of the process were tested and some changes were necessary for the test run. Thus, because the necessary changes, the creation of functional testing can become very cumbersome, especially when you want to test many tasks and flows that a business process can have.

Regarding the load test with JMeter, it was useful to explain failures observed in previous work with the application created with Bonita 5.7.2. Also proved to be a laborious test, or even impossible, depending on the BPMS used. Experience with two different versions of two BPMS strengthened this conclusion because there were different situations, with Bonita in version 5.7.2 the test was successful, but with the others tools the test could not be executed, since it failed to reproduce all requests.

On the other hand, the load test performed with Selenium and TestNG enabled all analyzed tools to be tested, making it a good alternative for load test allowing to evaluate the behavior of the application. However, this test has the disadvantage of having a higher cost to run, it can also be cumbersome to run (as well as other approaches) and not allow it to be analyzed the response time. Bu this approach can be useful when used BPMS which have much client-side processing which can not be correctly processed by the requests captured with JMeter.

Overall, how lessons learned we 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 under explored 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. van der Aalst, W. M. P.: Business process management: A comprehensive survey. *ISRN Software Engineering*, (2013).
2. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, 2nd edition, (2012).
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.: Gestão de processos de negócio em curso de sistemas de informação: Relato de experiência utilizando software livre. In *IX Simpósio Brasileiro de Sistemas de Informação*, 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) Graham, D. and Fewster, M.
7. Winter Green Research.: Business process management (BPM) cloud, mobile, and patterns: Market shares, strategies, and forecasts, worldwide, 2013 to 2019.
8. Rozinat, A., and van der Aalst, W. M.: Conformance testing: measuring the fit and appropriateness of event logs and process models. In *Business Process Management Workshops*, pp. 163–176, (2005)
9. Duarte Filho, N. F. and Silva, M. T. N. and Padua, C. I. P. : COMPARISON AND EVALUATION OF BPM TOOLS: FOCUS ON YOUR FEATURES. In *International Conference on Information Systems and Technology Management*, (2010)
10. Ferme, V. and Ivanchikj, A. and Pautasso, C.: A Framework for Benchmarking BPMN 2.0 Workflow Management Systems. In Business Process Management. In *Business Process Management* , pp. 251–259, (2015)
11. Van Der Aalst, W. M.: Business process management demystified: A tutorial on models, systems and standards for workflow management. In *Lectures on concurrency and Petri nets* , pp. 1–65, (2004)
12. Bourque, P. and Fairley, R. E.: Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 3.0. IEEE Computer Society (2014)
13. Chetty, N. K.: How to perform workflow testing for BPM applications (2014), `http://www.evoketechnologies.com`
14. Wang, Fei, and Wencai Du.: A test automation framework based on web. In *2012 IEEE/ACIS 11th International Conference on*, pp. 683-687, (2012).
15. BOWERS, A. and BELL, J.: Automated testing with Selenium and Cucumber. In *Agile Conference*, pp. 6-pp, (2006).
16. Pannu, Yadvinder Singh: Test Automation Using Cucumber and Selenium WebDriver.
17. Chiavegatto, R. et al.: Especificação e automação colaborativas de testes utilizando a técnica BDD. In *XII Simpósio Brasileiro de Qualidade de Software*, pp. 334–341, (2013)
18. de Moura, J.L. and Charão, A.: Automação de Testes em Aplicações de BPMS: um Relato de Experiência. In *XIV Simpósio Brasileiro de Qualidade de Software*, pp. 212–219, (2015)
19. Bindal, P. and Gupta, S.: Test Automation Selenium WebDriver using TestNG. In *Journal of Engineering Computers & Applied Sciences*, pp. 18–40, (2014)