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

Jéssica Lasch de Moura and Andrea Schwertner Charão

Computer Science Nucleus
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 different versios of 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 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 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 [8]. 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 [9]. 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 [10], 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.

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

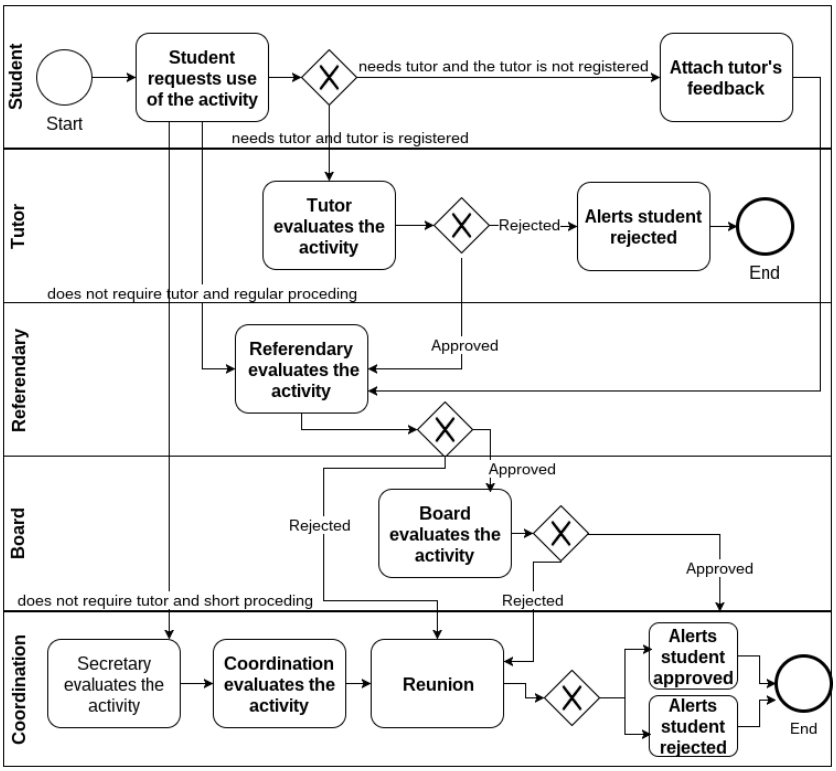

Fig. 1. BPMN process diagram

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

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" of this project: a public educational institution.

Among the analyzed tools (TIBCO[2], Activiti[3], Process Maker[4], Intalio[5]), none offered clear support for automated testing. Just Activiti tool cited the

[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

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, 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.

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. 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, 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

[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.

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, it was set up a scenario in which the student logs in the application and fills 2 forms as a first task of the process and after that another user logs in and runs the second task of the process available to him. For this scenario, methods were created by varying the entries on the forms. As the tested process is the same in all BPMS, the test scenario is also the same.

In a previous work [7], 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 scenario used in the tests with the four versions.

```
Feature: Testing BPM Process

Scenario Outline: Cenario 1
Given I start the process
When I select the First 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 Secretaria Avalia ACG task is ready

Examples:
  |row_index|
  |1|
  |2|

Scenario Outline: Cenario 2
Given I select the task Secretaria Avalia ACG available
When I fill the forms of Secretaria Avalia ACG task with excel row "<row_index>" of dataset
Then The next task is ready

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

Fig. 2. Test Scenario

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. It was 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 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, 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.

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.

4.2 Load Tests

In the load testing 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.

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/

	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

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 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.

In the tests in version 7.1.2 of Bonita, capture 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, it was impossible to find where the identifying keys are generated.

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, it was decided to verify the existence of a way to perform load testing through the information obtained from the Selenium.

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 it was chosen to perform load testing with Selenium. TestNG is a framework commonly used to automate testing with Selenium [11]. 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 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 [16]), 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 [17] to allow the browser to perform the test without opening a new window.

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 part of the load test code simulating 200 virtual users in the first task of the process can be seen in Figure 3. It was possible to run the load test using the code obtained in Selenium combined with TestNG. With this test, we were able to analyze the performance of two versions of the both

[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.

```
@Test(invocationCount = 200, threadPoolSize = 200)
public void SeleniumMasterTest() {
        //create Virtual Frame Buffer
        String Xport = System.getProperty("lmportal.xvfb.id", ":10");
        FirefoxBinary firefoxBinary = new FirefoxBinary();
        firefoxBinary.setEnvironmentProperty("DISPLAY", Xport);
        WebDriver driver = new FirefoxDriver(firefoxBinary, null);

        baseUrl = "http://[redacted]";

        driver.get(baseUrl + "/activiti-explorer/ui/");
        driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
        driver.switchTo().frame(driver.findElement(By.name("PID3")));

        //login
        driver.findElement(By.name("username")).clear();
        driver.findElement(By.name("username")).sendKeys("[redacted]");
        driver.findElement(By.name("password")).clear();
        driver.findElement(By.name("password")).sendKeys("[redacted]");
        driver.findElement(By.cssSelector("span")).click();

        //processes
        driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
        driver.switchTo().defaultContent();// inseri manualmente
        driver.findElement(By.xpath("//div/div/span/span[contains(.,'Processos')]")).click();

        driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
```

Fig. 3. Selenium Load Test Source Code

BPMS tools. In Table 2 may have seen an overview about which load tests could be performed.

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 2. Overview of the load tests

Load Test Results Due to problems reported in the previous section, load tests with JMeter could only be achieved in the implementation created with the version 5.7.2 of Bonita. In order to test the system behavior with different load levels, tests were performed with 1, 50, 100 and 200 virtual users and were analyzed 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 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 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 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 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 3. Average response times, in milliseconds

Overall, the load testing with JMeter helps explain the overload that occurred with the application in production, when many students 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 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.

On the other hand, 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 more about client behavior than server.

However, it was possible to analyze the server behavior 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 only test that returns no errors in this case was the test with the Activiti tool in version 19.5. 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 server 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 server 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 was observed in tests with JMeter. 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. However, 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 Considerações Finais

In this paper we explored automated test solutions at an application of BPMS. In the absence of supporting the functional and load tests in Bonita BPMS and Activiti was 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.

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 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.: 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. 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)
8. Winter Green Research.: Business process management (BPM) cloud, mobile, and patterns: Market shares, strategies, and forecasts, worldwide, 2013 to 2019.
9. Bourque, P. e Fairley, R. E.: Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 3.0. IEEE Computer Society (2014)
10. Chetty, N. K.: How to perform workflow testing for BPM applications (2014), `http://www.evoketechnologies.com`
11. Bindal, P. and Gupta, S.: Test Automation Selenium WebDriver using TestNG. In *Journal of Engineering Computers & Applied Sciences*, pp. 18–40, (2014)