Model-Based Testing in Agile Projects: An Approach Based on Domain-Specific Languages*

Aline Zanin[1], Avelino Fracisco Zorzo[1], and Henry Cabral Nunes[1]

PUCRS - Porto Alegre, Brazil
aline.zanin@edu.pucrs.br, henry.numes@edu.pucrs.br and
avelino.zorzo@pucrs.br

Abstract. Model-Based Testing (MBT) can bring several benefits to software quality. However, generally, MBT is applied in traditional software development lifecycle models, with few studies exploring its application in agile software development context. Hence, usually, agile development teams (AT) do not benefit from the advantages that the MBT technique provides, for example, reuse of artifacts and traceability between requirements and test artifacts. Thus, this article presents an approach for applying MBT in agile software development teams. This approach is based on the use of a semi-natural language to write scenarios for the automatic generation of models and test scripts. To exemplify the application of this approach, we also present a Domain-Specific Language (DSL) called Aquila, in which new functional test related keywords are added to the Gherkin DSL. We also present, based on a literature review, the majors challenges and difficulties of applying MBT in AT. To validate the proposed approach a Focus Group study was used.

Keywords: Software Testing, Agile, DSL, MBT

1 Introduction

The increasing search for quality improvement in software products induces companies to innovate and to optimize their processes, techniques and tools. Model-based Testing (MBT) is one of the techniques that helps such improvement in the context of software testing. MBT focuses on creating test artifacts and on reusing models designed by development teams. In general, these teams create models to perform activities such as requirements and systems analysis. So, MBT uses these artifacts to facilitate the generation and maintenance of test artifacts to improve traceability of system requirements and software testing [5].

MBT is widely used in traditional lifecycle software development, *e.g.* Waterfall lifecycle. However, although the benefits of MBT are already known and disseminated, few studies present the application of this technique in an agile software development context, since applying MBT in an agile context brings

* This study was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior – Brazil (CAPES) – Finance Code 001

new challenges. Those challenges are related to time management and human effort required to create models. Since the teams work with short cycles of continuous delivery, the time available for creating models, executing tests, and correcting possible failures is reduced [11] [10] [9].

Furthermore, according to the Agile Manifest [1], it is very important that Agile Teams (AT) use simple and efficient documentation. One way of doing that could be to use software requirements specifications using a Domain-Specific Language (DSL), for example Gherkin [4]. One the other hand, through the scenarios written in Gherkin and using Behavior Driven Development (BDD) [18], it is possible to generate methods (just skeletons) that will structure the test automation for a specific functionality.

On the other hand, BDD does not completely automate the creation of scripts for test automation, BDD only structures it. Therefore, it is still necessary for the test analyst to code the test scripts, and this is a time-consuming activity that could be used for other activities related to system development. MBT can provide the complete automation of this process, optimizing the human effort and time efficiency.

In this context, the research question of this study is: **How can MBT support the automated generation of test scripts in the agile software development context?**

In order to provide answers to the above question: 1) we analyzed the difficulties and challenges to apply MBT in AT reported by researchers in the literature; 2) based on the difficulties and challenges identified by these studies, we propose an approach for adopting MBT in AT; 3) taking into consideration Steps 1 and 2, we designed a new DSL, called Aquila; and, 4) to provide an evaluation of this approach and language, we conducted a survey in which participants had the opportunity to experiment Aquila, create a test scenario, and generate test scripts. To confirm this evaluation we conducted a Focus Group study where we discuss the applicability of Aquila.

This paper is organized as follows. Section 2 describes related work. Section 3 presents the challenges and difficulties to apply MBT in AT and also a set of practices to facilitate the adoption of MBT in AT. Section 4 presents a DSL that allows the use of MBT in AT. Section 5 describes the validation of our approach with AT members. Finally, Section 7 presents the conclusion of this study.

2 Related Work

This section describes some of the papers that were analyzed in a systematic literature review (SLR). We analyzed papers that present some discussion on MBT and AT, and also describe some challenges and difficulties faced by AT when applying MBT. The literature analysis is based on a previous SLR [2]. That SLR mapped all the studies that present approaches, strategies, techniques, methods and methodologies for MBT adoption and were published up to 2016. We analyzed the papers selected for that study [2], and by reading the titles and abstracts, we extracted the papers that present MBT specifically applied to

agile context [10] [17] [8] [13]. Later, based on the related work of these selected papers, other papers were identified. The main papers directly related to this work are mentioned next.

Törsel [22] presents a study that uses Domain-Specific Language (DSL) to create models for test scripts generation. This study is different from ours, because the DSL used for creating models is not semi-natural language, it is similar to a programming language.

Yue *et al.* [19] present an approach to perform MBT based on DSL. The major difference between our approach and the one proposed by Yue *et al.* is that we use an extension to Gherkin to design scenarios and automatically generate models. Besides that, using our approach it is easier to understand how the mapping from the scenario to the model is performed, allowing to implement graphics extension for visual display of this model. In Yue's work, a behavioral model is not generated, and it is difficult to analyze the flows generated by the MBT technique.

Entin *et al.* [8] introduce an approach to use semi-natural language and Gherkin to generate models to apply MBT. However, the scenarios must be exclusively written using Gherkin, which does not have enough details to generate a complete model that allows the generation of test scripts. For this reason, human interference is required to create the model, and no strategy is proposed to automate the generation of test scripts from these models.

Li *et al.* [12] present an approach to generate Gherkin scenarios from models. This approach is similar to ours because both use Gherkin. However, our approach generates models from scenarios and test scripts from models. So, it is possible to create models and generate test scripts only using semi-natural language in our approach.

Dwarakanath *et al.* [6] and Thummalapenta *et al.* [21] propose approaches where DSLs are used to automatically generate test scripts. The DSLs used in those works are similar to the one proposed in our approach, since all of them use semi-natural language based on English. However, Dwarakanath and Thummalapenta do not use the MBT technique. Thus, those works do not benefit from the advantages provided by MBT. For example, in our approach, it is possible to combine alternative system flows to generate test scripts that validate all possible flows of the System Under Test (SUT).

3 Applying MBT in AT: Using a DSL

The main difficulties and challenges for using MBT in AT that we identified through our systematic literature review, mentioned in Section 2, are summarized as: 1) Models created by AT rarely contain enough details to apply MBT [11] [10] [13]; 2) It is difficult to guarantee that all requirements are covered by the generated tests artifacts [17] [11]; 3) AT, in general, prefer easy to use testing tools [11]; 4) Agile process often works with short development cycles [7] [8]; 5) Lack of time to create models [7] [8]; 6) Continuous requirements changes [7] [9] [17] [8]; 7) Difficult test artifact maintenance [9] [8]; and, 8) Professionals

do not have enough expertise in software modeling languages such as Unified Modeling Language (UML) [9] [8] [13].

Based on the difficulties and challenges mentioned above, we can point out to the following improvements in the MBT process to make it applicable by AT:

1. Supporting automatic creation of models, based on the requirements documents usually adopted by AT, such as user stories or scenarios. This is important for reducing the learning curve and simplify the adaptability of the teams to the technique.
2. Supporting requirements changes by modeling adaptation.
3. Supporting automation of script generation, avoiding the generation of manual test cases.
4. Avoiding the generation of excessive documentation.

Based on these suggestions, we propose, in this work, a solution focused on the automation of model creation. To this end, the idea is that the software testers write requirements in a semi-natural language, and from this language, models could be generated automatically.

One example of semi-natural language used to write scenarios in AT is Gherkin [4]. However, in order to make possible to generate models to apply MBT in AT, the language used to write scenarios, should also provide a simple way of describing, also, functional testing requirements, which currently is not supported by Gherkin, and, therefore, an extended version of Gherkin is needed (see Section 4). Gherkin scenarios are written basically using the keywords GIVEN, WHEN, THEN and AND. GIVEN represents the actual state of the system, WHEN represents one action performed in the system and THEN represents the result of this action. The AND keyword can be used with GIVEN and THEN keywords to complement the information.

Therefore, in order to apply a DSL in MBT and AT, the following guidelines should be observed when designing a new DSL:

1. All scenarios must have, in addition to the status information and expected result, details of actions that must be performed by the tester in the system.
2. Scenarios should not contain too much information, for example, they should not explicitly contain the step-by-step test case describing each of the fields in a form. This is because in this case the improvement in productivity, through automation, would be negatively affected.
3. Scenarios should be complemented with keywords that represent generic behaviors (which often happens in several functionality), for example, filling in all fields in a form with valid information.
4. Scenarios should be complemented with generic result verification information, for example, validating a value in a form.
5. Each scenario should the converted to a model, in order to be able to apply MBT.
6. From the generated model, it should be possible to create test artifacts for any system platform.

7. From the generated model, it should be possible to create test artifacts for any functional testing tool.

From the above mentioned guidelines, a new Domain-Specific Language, called **Aquila**, was created. This new language includes each of the above-mentioned guidelines to Gherkin scenarios. This new DSL is described in Section 4.

4 Aquila - a DSL for Agile Functional Testing

Aquila is a Domain-Specific Language for agile functional testing. Hence, one of the major requirements of Aquila design is that Aquila can be used to write test documentation, *e.g.*, test scenarios. In this context, Aquila was designed as an extension of Gherkin, thus, the main characteristic of Aquila is the insertion of test information in Gherkin scenarios. This test information is represented by adding new keywords in test scenarios. These new keywords are related to the user interaction with the system, for example: input values and click buttons. This test information will allow one to automatically generate a model that will later be translated into test scripts to execute the test automatically. For the definition of these keywords, we performed a mapping of the main actions that a user can perform in a system. This mapping is performed initially through the W3C [23] documentation[1]. We also address some aspects of user interaction based on our experience performing software testing in industry [15] [3] [14].

Each Aquila keyword is associated to a Gherkin keyword. The association between Aquila keywords and Gherkin keywords follows the following pattern: Aquila keywords that represent actions are associated to the Gherkin keyword WHEN; Aquila keywords representing results are associated to the THEN keyword; and, symbols "{ }" and "< >" are used with the GIVEN keyword.

Table 1 shows the selected keywords, the behavior represented by each of the keywords and the associated Gherkin keyword. These keywords are used in the mapping between the scenarios and the models, in which each keyword has a specific representation in the generated models.

Some Aquila scenarios can require input or selection of values, *i.e.*, scenarios that use keywords: put, use-valid-data, checked, choose, and select-data. For this purpose, input tables are created and in these tables the name of the fields and their respective input values are specified. For example, for a scenario that requires input data for the fields "Country" and "State", the following piece of Aquila code would represent the tables that contains field names and their respective values. This example can be seen in Figure 1

For the use-valid-data keyword, it is possible to create a table with all fields, and respective values, as shown in the next piece of Aquila code. This example cam be see in Figure 2.

[1] https://www.w3.org/Consortium/

Table 1. Aquila DSL Keywords

Aquila keyword	**Definition**	**Gherkin keyword**
< >	Used to enter a URL	GIVEN
{ }	Used to reference a scenario in another scenario	GIVEN
click[field]	Represents the action of clicking on a specific field	WHEN
click-link[field]	Represents the action of clicking on a link	WHEN
put[field]	Represents the action of inserting values in a specific field	WHEN
use-valid-data	Represents that all fields present on the feature under test and listed in the input table will receive valid values	WHEN
dont-fill-out[field]	Represents that no values will be entered in a specific field	WHEN
checked[field]	Represents the selection of an option in a checkbox	WHEN
choose[field]	Represents the selection of an option in a radio button	WHEN
select-data[field]	Represents the selection of an option in a list	WHEN
mouse-over[field]	Represents the placement of the mouse over a given field	WHEN
enable[field]	Represents verification if a field is active	THEN
disable[field]	Represents verification if a field is not active	THEN
showed[field]	Represents checking for the presence of a particular word or phrase on the SUT	THEN
opened[url]	Represents checking the correct opening of a page	THEN
showed-title[word]	Represents page title verification	THEN

```
When I select-data[Country]
| Country |
| USA     |
And I select-data[State]
| State   |
| Nevada  |
| Texas   |
```

Fig. 1. Select-data keyword example

```
When I use-valid-data:
| Country | State  | Email          | FirstName |
| France  | Loire  | alice@test.com | Alice     |
```

Fig. 2. use-valid-data keyword example

4.1 Aquila DSL Usage

In this section, we describe how a DSL, *i.e.*, Aquila, is used in order to generate and execute test scripts in the context of AT. The complete flow of Aquila utilization is formed by six steps (see Figure 3), in which, two require human interaction (Writing Aquila scenarios and Test Script Improvements), three are automatically executed by Aquila tool (Model Generation, Test Sequence Generation and Test Script Generation) and one is executed by a testing automation tool, for example, Selenium Webdriver. The Aquila tool is available for downloading at: https://github.com/alinnezanin/Aquila/

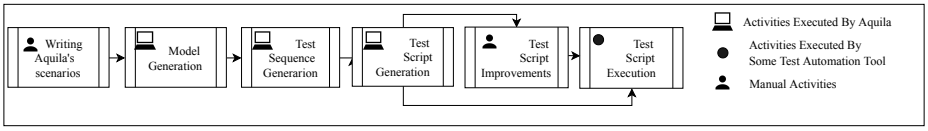

Fig. 3. Aquila steps

1) Writing Aquila's scenarios: This is one of the majors steps of the Aquila usage process. In this step, the scenarios are created. To exemplify each step, we use an online web store, *i.e.*, http://demo.cs-cart.com/. For example, to test this web store we chose the scenario that validates the registering of a new customer and the insertion of a new product into the shopping cart. Figure 4 shows an Aquila code that represents this scenario. In the code, it is possible to see the use of the following tags: click-link, click, use-valid-data, put and showed.

2) Model Generation. The model chosen for MBT application, in this work, is the Directed Acyclic Graph (DAG). In the model generation process, each DAG node represents an Aquila keyword used in the scenario. The exception to this rule is the use-valid-data keyword that generates not only one node, but one node for each field present on the system and specified in the input table. For the scenario in Figure 4, the respective model is represented in Figure 5. In this model, it is possible to realize that there is some alternative flows. These flows represent the second input option for the use-valid-data and put keywords specified in the input table (lines 7,8,13 and 14 of the scenario in Figure 4). It is important to highlight that the number in the DAG nodes is the line number in the scenario code.

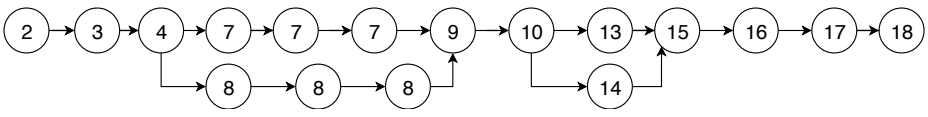

Fig. 5. New User Registration Model

```
1. Scenario:Register Customer and Add Product
2. Given <https://demo.cs-cart.com >
3. When I click-link[my account ]
4. And I click[register ]
5. And I use-valid-data:
6. |email| password| news|
7. |custo@mer.com | c123@ |news|
8. |tes@te.com | a143@ |news|
9. And I click[register ]
10. And I click[search-performed ]
11. And I put[search-performed ]
12. |search-performed|
13. |bike|
14. |game|
15. And I click[search ]
16. And I click-link[Catania ]
17. And I click[addToCard ]
18. Then showed[The product was added to your cart ]
```

Fig. 4. Register a new customer and add a product to the customer cart

3) Test Sequence Generation. A test sequence is generated before the test script creation and describes the path traversed between the initial node and the last node in the graph. To generate a test sequence, we use the Depth-First Search algorithm (DFS) [20]. Using this algorithm, test sequences that combine the inputs used in each field of the system are generated. This ensures that if there are peculiar behaviors related to the combination of two or more inputs, these behaviors are tested.

The sequences are formed by the numbers that represent lines of a scenario and their respective nodes in the DAG. For a scenario line that contains the use-valid-data keyword, this line will be repeated several times in the sequence, because this keyword generates multiple nodes in the DAG, one for each field that will be filled.

For the model in Figure 5, four test sequences are generated. This is because there are two alternative flows in the model, and Aquila combines these flows. The following test sequences are generated:
1) 2 - 3 - 4 - 7- 7- 7 - 9 - 10 - 13 - 15 - 16 - 17 - 18
2) 2 - 3 - 4 - 7- 7- 7 - 9 - 10 - 14 - 15 - 16 - 17 - 18
3) 2 - 3 - 4 - 8- 8- 8 - 9 - 10 - 13 - 15 - 16 - 17 - 18
4) 2 - 3 - 4 - 8- 8- 8 - 9 - 10 - 14 - 15 - 16 - 17 - 18

4) Test Script Generation. The test script is based on the previously generated models and test sequences. For each model, one or more test scripts can be generated according to the number of generated test sequences. Figure 6 shows a snippet of a code generated for the Selenium Webdriver tool. This tool was chosen because it is widely used within IT companies. However, the same

Aquila scenario can be used to generate scripts for different tools or languages. Figure 6 shows the code for the "Register Customer and Add Product" scenario.

```
1     @Test
2     public void newCustomerRegistration() {
3       Webdriver driver = new FirefoxDriver();
4       driver.get("https://demo.cs-cart.com");
5       WebElement linkText = driver.findElement(By.linkText("
            my account"));
6       linkText.click();
7       WebElement register = driver.findElement(By.name("
            register"));
8       register.click();
9       WebElement email = driver.findElement(By.name("email"))
            ;
10      email.sendKeys("cuso@mer.com");
11      WebElement password = driver.findElement(By.name("
            password"));
12      password.sendKeys("c123@.com");
13      WebElement register2 = driver.findElement(By.name("
            register"));
14      register2.click();
15     (...)
16    }
```

Fig. 6. Generated code for "Register Customer and Add Product" scenario

5)Test Script Improvements. These improvements might be necessary when a test engineer uses some field name different from the one used in the application, to correct some mistake in the scenarios, or to create a script for an action that is not completely covered by Aquila. In these cases a partial script is generated, and the test engineer will have to complete the generated script. The partial script is generated into a library and can be edited at any time. This allows the test engineer to customize their tests for peculiar situations not covered by Aquila tags.

6) Test Script Execution. To execute the test scripts, it is necessary to use a traditional testing automation tool, for example: Seleniun Webdriver. This tool should be compatible with the syntax of the generated script.

5 Aquila Validation: a Focus Group Study

In order to validate the applicability of Aquila, we used a Focus Group method. Hence, we established two research questions (RQ) that we wanted to answer by the end of the Focus Group study: *RQ1*) What is the specialists perception on the productivity that can be achieved when using Aquila, compared to manual test scripts creation for test automation? *RQ2*) What are the experts' perceptions

regarding the learning curve on the use of Aquila, when compared to the learning curve for the manual test scripts creation for test automation?

Once the research questions were defined, we set the profile and the number of specialists that would participate in the Focus Group execution. There were eight people participating in this Focus Group session. Table 2 presents an overview on the subjects profile. The Focus Group session was performed based on following questions:

Table 2. Individual Subjects Profile

S1	Professional with more than 2 years of programming experience, initial knowledge in test automation, business and Gherkin and no management knowledge.
S2	Professional with up to 2 years of Gherkin test automation experience and more than 2 years of programming, business and management experience.
S3	Professional who has up to 2 years of Gherkin experience, initial knowledge in test and business automation and no programming knowledge and agile project management.
S4	Professional with initial knowledge in test automation, business, management and Gherkin and no programming knowledge.
S5	Professional who has more than 5 (five) years of testing experience and business experience, has up to 2 (two) years of Gherkin experience and has initial knowledge in programming and agile project management.
S6	Professional with up to 2 (two) years of experience in test automation and programming, with more than 2 (two) years of business and management experience and with more than 5 (five) years of experience with Gherkin.
S7	Professional with more than 5 (five) years of programming experience, up to 2 (two) years of experience in test automation and initial knowledge in business, management and Gherkin.
S8	Professional with up to 2 (two) years experience with Gherkin test automation and more than 5 (five) years experience with programming, business and management.

1. Do you believe that using Aquila can improve team productivity? If yes, how?
   - *Discussion analysis*: For this topic it can be said that the group considered that Aquila influences positively the team productivity, provided that some conditions are respected: the tester must have a minimum knowledge on test automation; the business person needs to be involved with the creation of the scenarios; and, the project must be organized for this framework of scenario utilization and test automation. These conclusions are reinforced by the following statements:

     "I have a feeling that if the tester is a beginner on test automation, it is a lot easier, but it is a feeling [S4]."

     "If we take this to the business area, for them to write the test scenarios, and someone else just complement them, it is much faster for them to type keywords than it is to write Java code [S2]."

2. Compared to the manual creation of scripts for test automation, would you consider Aquila more productive, less productive or indifferent? Justify.
   - *Discussion analysis*: In this topic, there was a consensus among participants that Aquila could present an improvement in productivity when compared to the manual creation of scripts for test automation. However, some limits were pointed out by the participants that should be the focus of the researchers in the development of future work related to Aquila. These conclusions are reinforced by the following statements:
     "But compared to manual test scripts creation I believe it would increase productivity for the reasons already mentioned [S4]."
     "It could be productive to some extent depending on what would have to be changed manually [S7]."
3. What is your perception regarding the learning curve of the use of Aquila in relation to the learning curve for manual creation of scripts for test automation?
   - *Discussion analysis*: Analyzing the participants comments, we can conclude that Aquila could easily be learned and used by professionals. Like any innovative tool, Aquila requires a learning curve and a period of adaptation, but the discussions indicate that the curve is smaller than the curve that usually exists for learning, for example, a programming language and that Aquila can easily be used in conjunction with any other automation tool. Furthermore, Aquila will generate the code that represents actions, and a beginner programmer could see this code generated, being able to analyze, understand and memorize it. These findings are reinforced by the following statements from the participants.
     "In my opinion Selenium (manual creation) is more complex, Aquila does not have many commands like Selenium, so it is easier [S6]."; "I think it is simpler because it is closer to a natural language [S8]."
     "I think the Aquila learning curve is less steep, but a person needs to know how to write tests manually to be able to edit them [S5]."
4. Would you apply Aquila on a test project that is working with agile software development? Why?
   - *Discussion analysis*: In this topic several opinions were expressed by the participating subjects. No participating subject stated that they would not use Aquila, however, several points were raised that would be considered by professionals when deciding to use Aquila or not. The main point, which was quoted by several participants, is that Aquila application is better recommended in projects that are already using the BDD technique or writing requirements using Gherkin. Another factor considered to be determinant in the choice of whether or not to use Aquila is the amount of changes that need to be made to the Aquila generated scripts, and this amount of changes is highly related to system complexity and type of features that the system contains. Analyzing the subjects comments, we can say that Aquila can be applied to the context that it was proposed to. It was designed for AT that use Gherkin for writing requirements, or AT that are looking for using this approach. As for the

amount of scripts changes needed, Aquila is prepared to automate most of the actions that can be performed on the system for the domain to which it was proposed to.

It is important to emphasize that the participants already had previous contact with these questions. After reading, and answering the questions individually, they discussed the same questions as a group. The conclusions mentioned above are related to the group discussion.

6 Aquila Validation: a Survey with professionals

This study aimed to understand the interference of DSL Aquila on productivity and test automation learning curve, considering the opinion of testing engineers. For this study we brought together a group of professionals who work with software testing on agile teams. These professionals used the Aquila DSL, and based on this usage, they provided feedback about Aquila.

Thirteen professionals participated in this study. All of them work in agile teams (their profile is detailed in Table 3). The profile considers the professionals self-assessment about their previous knowledge according to the following following criteria: Beginner: I have knowledge, I can perform basic tasks; Intermediate: I have knowledge and can perform complex tasks; Advanced: I have knowledge and can perform all kinds of tasks including management; I am not aware: Professional without knowledge on the subject.

Table 3. Profile of the subjects

Knowledge area	**Knowledge level**			
	Beginner	**Intermediate**	**Advanced**	**Not aware**
Test Automation	9	4	0	0
Gherkin	6	4	0	3
Agile Methods	4	7	2	0
Software Test	3	4	0	3

The participants took one hour to write the scenarios and to generate the scripts. After completing the challenge, participants answered questions on: 1) Learning difficulty of DSL Aquila; 2) Possibility to write scenarios and generate scripts using Aquila in a sprint of an agile project; 3) Advantages and disadvantages of using Aquila in an agile project. They were also asked to suggest improvements on the language and tool.

The first question is related to the difficulties encountered in learning to use Aquila. In this sense, ten participants answered that they consider it was easy to learn; two participants considered that the level of difficulty was average, and one participant considered it difficult.

In the second question, a comparison is made between the easiness to learn Aquila against the easiness to learn other testing tools. For this question, seven

participants found it was easier to learn Aquila, four mentioned the learning curve was similar, and two said it was harder. The two participants, who believed it was more difficult to learn Aquila than another testing tool, had never seen Gherkin before, and, therefore, either way, previously knowledge on using a DSL (for example, Gherkin) may have influenced the participants perception.

The third question aimed to understand if it would be possible to complete the DSL Aquila utilization cycle in an agile project sprint. For this question, only one participant said he did not believe using Aquila would help in an agile project sprint. Regarding the advantages, disadvantages and improvements of Aquila, several professionals have pointed out as advantages the easiness of use and the fact that the tool generates scripts. Regarding disadvantages, one participant mentioned that the code generated by the tool presented some faults (this has already been reviewed and the faults corrected). Some of the participants also pointed out that tool usability should be improved.

7 Conclusion

In this paper we presented an approach to use MBT in Agile projects. This approach was validated through a survey in which participants had the opportunity to experiment Aquila, and its tool. In the experiment, all participants were able to complete the full scenario creation and test scripts generation. This produced an excellent view from the participants in the sense that Aquila can positively influence productivity and facilitate the learning of test automation. This result confirms the result that we had already obtained through a focus group study, before the tool was ready to be used. Some improvements are still necessary, as mentioned before, in order to the full adoption of our language and tool in actual agile environments. For example, a framework such as Usa_DSL[16] may be used to help improving Aquila usability.

References

1. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al.: Manifesto for agile software development. http://www.agilemanifesto.org (2001), access in 03/07/2019
2. Bernardino, M., Rodrigues, E.M., Zorzo, A.F., Marchezan, L.: Systematic mapping study on mbt: tools and models. IET Software **11**(4) (2017)
3. Bernardino, M., Zorzo, A.F., de M. Rodrigues, E.: Canopus: A domain-specific language for modeling performance testing. In: 9th IEEE International Conference on Software Testing, Verification and Validation (ICST) (2016)
4. Cucumber: Gherkin. https://docs.cucumber.io/gherkin/ (2019), access in 03/07/2019
5. Dalal, S.R., Jain, A., Karunanithi, N., Leaton, J., Lott, C.M., Patton, G.C., Horowitz, B.M.: Model-based testing in practice. In: International Conference on Software Engineering (ICSE) (1999)
6. Dwarakanath, A., Era, D., Priyadarshi, A., Dubash, N., Podder, S.: Accelerating test automation through a domain specific language. In: 10th International Conference on Software Testing, Verification and Validation (ICST) (2017)

7. Entin, V., Winder, M., Zhang, B., Christmann, S.: Combining model-based and capture-replay testing techniques of graphical user interfaces: An industrial approach. In: 4th International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (2011)
8. Entin, V., Winder, M., Zhang, B., Claus, A.: A process to increase the model quality in the context of model-based testing. In: 8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (2015)
9. Entin, V., Winder, M., Zhang, B., Christmann, S.: Introducing model-based testing in an industrial Scrum project. In: 7th International Workshop on Automation of Software Test (AST) (2012)
10. Jalalinasab, D., Ramsin, R.: Towards model-based testing patterns for enhancing agile methodologies. In: 14th International Conference on Intelligent Software Methodologies, Tools, and Techniques (SoMeT) (2012)
11. Katara, M., Kervinen, A.: Making model-based testing more agile: a use case driven approach. In: 11th Haifa Verification Conference (HVC) (2006)
12. Li, N., Escalona, A., Kamal, T.: Skyfire: Model-based testing with cucumber. In: 9th International Conference on Software Testing, Verification and Validation (ICST) (2016)
13. Li, N., Escalona, A., Kamal, T.: Skyfire: Model-based testing with cucumber. In: 9th International Conference onSoftware Testing, Verification and Validation (ICST) (2016)
14. de M. Rodrigues, E., Bernardino, M., Costa, L.T., Zorzo, A.F., de Oliveira, F.M.: Pletsperf - A model-based performance testing tool. In: 8th IEEE International Conference on Software Testing, Verification and Validation (ICST) (2015)
15. Rodrigues, E.M., de Oliveira, F.M., Costa, L.T., Bernardino, M., Zorzo, A.F., do Rocio Senger de Souza, S., Saad, R.S.: An empirical comparison of model-based and capture and replay approaches for performance testing. Empirical Software Engineering **20**(6), 1831–1860 (2015)
16. Rodrigues, I.P., Zorzo, A.F., Bernardino, M., de Borba Campos, M.: Usa-DSL: usability evaluation framework for domain-specific languages. In: 33rd Annual ACM Symposium on Applied Computing. pp. 2013–2021 (2018)
17. Sivanandan, S., B, Y.C.: Agile development cycle: Approach to design an effective model based testing with behaviour driven automation framework. In: 20th Annual International Conference on Advanced Computing and Communications (ADCOM) (2014)
18. Smart, J.: BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle
19. Tao, Y., Shaukat, A., Zhang, M.: Rtcm: A natural language based, automated, and practical test case generation framework. In: International Symposium on Software Testing and Analysis (ISSTA) (2015)
20. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM journal on computing **1**(2), 146–160 (1972)
21. Thummalapenta, S., Sinha, S., Singhania, N., Chandra, S.: Automating test automation. In: 34th International Conference on Software Engineering (ICSE) (2012)
22. Törsel, A.: A testing tool for web applications using a domain-specific modelling language and the nusmv model checker. In: 6th International Conference on Software Testing, Verification and Validation (ICST) (2013)
23. W3C: Html input types. https://www.w3schools.com/ (2018), access in 03/07/2019