Automated Acceptance Tests as Software Requirements: An Experiment to Compare the Applicability of *Fit Tables* and *Gherkin Language*

Ernani César dos Santos(✉) and Patrícia Vilain(✉)

Federal University of Santa Catarina, Florianópolis, Santa Catarina, Brazil
ernani.santos@posgrad.ufsc.br,
patricia.vilain@ufsc.br

Abstract. It is estimated that 85% of the defects in the developed software are originated from ambiguous, incomplete and wishful thinking software requirements. Natural language is often used to write software requirements specifications as well as user requirements. However, natural language specifications can be confusing and hard to understand. Some agile methodologists consider that acceptance tests are more precise and accurate sources of information about the customer's needs than descriptions in natural language. Several studies have addressed the use of acceptance tests as software requirements specification. Therefore, none of the previous studies has performed experiments to compare the applicability of different acceptance testing techniques in order to support an organization in the selection of one technique over another. This paper addresses this problem reporting an experiment conducted with undergraduate students in Computer Science. This experiment compares the applicability of two acceptance testing techniques (Fit tables and Gherkin language) as software requirements specification. This research tries to answer three questions: (a) Which technique is the easiest to learn in order to specify acceptance test scenarios? (b) Which technique requires less effort to specify acceptance tests? (c) Which technique is the best one to communicate software requirements? The results show that there is no sufficient evidence to affirm that one technique is easier to specify test scenarios or better to communicate software requirements. Whereas, the comparison of effort in terms of time to specify acceptance testing shows that the mean time to specify test scenarios using Gherkin language is lower than Fit tables.

Keywords: Acceptance testing · Software requirements · Fit tables
Gherkin language · FitNesse · Cucumber · TDD · ATDD · BDD

1 Introduction

Software system functionalities are specified through requirements engineering artifacts, which are a valuable starting point for the software development [1]. Natural language is often used to write system requirements specifications as well as user

J. Garbajosa et al. (Eds.): XP 2018, LNBIP 314, pp. 104–119, 2018.
https://doi.org/10.1007/978-3-319-91602-6_7

requirements [2]. According to [1], most of the software requirements specifications are written in natural language.

However, natural language specifications can be confusing and hard to understand. Various problems can arise when requirements are written in natural language, for example, readers and writers can use the same word for different concepts, or even, it is possible to express the same concept in completely different ways [2]. In addition, it is estimated that 85% of the defects in the developed software are originated from ambiguous, incomplete, and wishful thinking software requirements [3].

Some agile methodologists utilize acceptance tests as a way to specify software requirements [3–6] instead of using more common artifacts based on natural language. They consider that acceptance tests are more precise and accurate sources of information about the customer's needs than descriptions in natural language [7].

Besides the improvement over requirements specification expressed in natural language, acceptance tests also collaborate to the requirements gathering process, because they promote integration between stakeholders and software engineers during the writing of test scenarios of the application to be developed.

Several studies have addressed the use of acceptance tests as software requirements specification. However, none of the previous studies has performed experiments using more than one technique to compare the applicability of them as software requirements in the same project in order to support organizations in the selection of one technique over another. This paper addresses this problem reporting an experiment conducted with undergraduate students of the Computer Science program at the Federal University of Santa Catarina to compare the use of a tabular notation for acceptance test scenarios versus a textual scenario notation, which are Fit tables and Gherkin language, respectively.

The rest of this paper is organized as follows. Section 2 presents the related works. Section 3 presents an overview of the main concepts related to this paper. Section 4 defines the design of our experimentation and the research questions. In Sect. 5 we propose answers for each research question and discuss the results. Section 6 presents the threats to the validity. Section 7 presents the conclusion and future works.

2 Related Works

In [4] two experiments were conducted using the tables of the Framework for Integrated Test (Fit). The results show that when software requirements are written in natural language and complemented by Fit tables, they become four times easier to understand by developers than when Fit tables are not used. However, the authors claim that Fit tables do not replace textual requirements, but rather, they suggest that these tables bridge the gaps of software requirements specification which are written exclusively using natural language, reducing the ambiguity and misinterpretation of them.

In [5] an experiment with master students was performed. The experiment aims to verify the use of executable Fit acceptance test scenarios as software requirements in maintenance and evolution tasks. The results indicate that Fit tables help developers to

perform the maintenance tasks correctly and they also show that these tables may be used to perform regression tests.

Melnik et al. [7] have performed an experiment to show that non-technical users, working together with software engineers, can use acceptance test scenarios as a way to communicate and to validate software business requirements. The acceptance testing technique used in this experimentation was the Fit tables. Although the experimentation concludes that non-technical users can specify clearly software requirements using Fit tables, it points out that users have difficulty in learning how to specify test scenarios using this notation. Additionally, this study shows that some non-technical users do not approve the use of Fit tables as an artifact to specify requirements.

A user-centered language called BehaviorMap is proposed in [8]. This language is based on behavior models written in Gherkin language that aims to specify behavioral user scenarios in a cognitive way. In this study, an experiment was conducted with 15 individuals to verify the understandability of the BehaviorMap. The results show that BehaviorMap scenarios are easier to understand in relation to textual scenarios, especially when considering scenarios with higher complexity.

The use of acceptance test scenarios as an artifact to specify software requirements were also analyzed in [9], which performed an experiment to verify the capability of non-technical users in creating user scenarios of a puzzle game using acceptance testing. The acceptance testing technique used in this experimentation was the User Scenario through User Interaction Diagram (US-UID). The experimentation has pointed out that non-technical users could create US-UID scenarios of the application correctly with a few hours of training.

These previous studies have focused on verifying the applicability of acceptance tests as an artifact to clarify software requirements specifications written in natural language or have checked their applicability as software requirements specifications rather than using artifacts such as user stories or use cases. However, none of them compared the applicability of two different notations to express acceptance tests and their adherence to communicate software requirements. This study compares the use of a tabular notation, Fit tables, versus a textual scenario notation, Gherkin language, in terms of: ease of learning, ease of use, effort required to specify acceptance tests scenarios, and capability to communicate software requirements.

3 Background

Test-driven development (TDD) is a software development approach which tests are written before beginning the development of the SUT, this practice becomes widely established after 1999 by Kent Beck. This practice is performed in five steps, as follows [13]:

1. Write a new test case.
2. Run all test cases and see the new one fails.
3. Write just enough code to make the test pass.
4. Re-run the test cases and see them all pass.
5. Refactor code to remove duplication.

In 2002, Ward Cunningham introduced the concept of Fit Tables. In this approach, users write acceptance tests using Fit tables, and programmers write fixtures (glue code) to connect these tables with the future source code of the SUT. The remaining process of this approach is equivalent to steps 2 through 5 of the TDD. This process is called Acceptance test-driven development (ATDD) because acceptance tests are written before the SUT [14].

Acceptance testing is a black box testing that aims to determine if a software system meets customer requirements from user's point of view [3, 7, 9]. As defined in the IEEE Standard 1012-1986 [10], acceptance testing is a "formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system".

Fit is an example of framework to express acceptance test scenarios. Using this framework, the acceptance tests are written in the form of tables, which are called Fit tables. Besides Fit tables are used to represent test scenarios, they are also used for reporting the results of tests [11]. Figure 1 shows an example of Fit report table, which was used to perform several tests in a functionality to calculate discount over an amount. The first column of this table, named amount, represents an input, whereas, the second columns, which name is followed by parenthesis, represent the expected output. When a test fails, the expected and actual output values are showed to the user [11].

CalculateDiscount	
amount	discount()
999.00	0.00
1000.00	0.00 *expected* 50.0 *actual*
1010.00	50.50
1200.00	60.00

Fig. 1. Fit table report sample [11].

Behavior-driven development (BDD) is an agile software development approach that enhances the paradigm of TDD for acceptance testing. In the BDD approach, the behavior of the SUT is described through user stories and acceptance tests before beginning its development. Scenarios representing the user stories are described using BDD languages, such as Gherkin language [12].

Gherkin language is a domain specific language (DSL) that can express the behavior and the expected outcome of the SUT [12]. It uses some words as commands, such as Given, When and Then. The word *Given* expresses the inputs or pre-conditions to perform a test, the word *When* expresses conditions or specified behavior, and the word *Then* expresses expected outputs or expected changes due to the specified behavior. As with the Fit tables, Gherkin language also needs a glue code to connect

the features (a set of test scenarios) with the source code of the SUT [12]. An example of the syntax of this language is as follows:

```
Scenario: Pop element
Given a non-empty Stack
When a stack has N elements
And element E is on top of the stack
Then a pop operation returns E
And the new size of the stack is N-1
```

4 Experiment Definition

In this section, we report the experiment definition, design, and planning, following the guidelines proposed in [15, 16], as well the experiments conducted in [3, 4]. The objective of this experimentation is to compare the applicability of Fit tables and Gherkin language to communicate requirements in a software development process regarding specification effort and requirements consistency. The perspective is to adopt Fit tables or Gherkin language to express software system requirements in outsourcing contracts for software development. The context of the experiment consists of undergraduate students (subjects) and a Java application (object). The participants (subjects) involved in the experiment are undergraduate students in the last years of the Computer Science program. The object of this study is a human resource (HR) management application named HRS, which supports functionalities such as compliance, payroll, personnel files, and benefits administration.

4.1 Experiment Participants

The participants were 18 students from a course called Special Topics in Technology Applications I, in the last years of the bachelor's degree in Computer Science at UFSC. The students have already attended courses on software programming and software engineering, and they had a medium knowledge and expertise level in programming and software engineering topics. The most of them have been taking part of trainee programs. The participants have never taken any course or professional experience in Fit or Gherkin language. Although the experiment was conducted as a mandatory activity of the course, the students were not graded based on the artifacts produced, but rather, they were graded based on their participation. Also, students were advised that the activities were parts of an experiment to compare the applicability of two acceptance testing techniques as artifacts to communicate software requirements.

4.2 Experiment Material

The experiment was performed on a set of four requirements for the HRS application. We granted access permission for each participant to access a web directory that contains a textual description of the application, instructions to set up the application project (download the zipped Java project and import it into the Eclipse IDE), a time

sheet and a questionnaire. The timesheet was used by the participants to take note of the time spent in each task of the experiment. The questionnaire is a set of 24 questions to investigate the background of the participants and to perform a qualitative analysis of the performed tasks. The answers for these questions are five-point scales, such as 1 = Strongly agree, 2 = Agree, 3 = Not certain, 4 = Disagree, 5 = Strongly disagree.

The development environment was set up by the participants, who received a tutorial to guide this activity. The tutorial content is:

- Installation and configuration of Java Enterprise;
- Installation and configuration of the standalone version of Fit wiki and its dependencies;
- Installation and configuration of Eclipse IDE with the Cucumber plugin;
- Quick-start examples to validate all development environments.

4.3 Hypothesis Definition

Considering Fit tables and Gherkin language as available acceptance testing techniques, this experiment addresses the following research questions:

- **RQ1.** Which of these acceptance tests techniques is easier to learn?
- **RQ2**. Which of these acceptance tests techniques requires less effort (in time) to specify acceptance test scenarios?
- **RQ3**. Which of these acceptance tests techniques is the best one to communicate software requirements, as a form of expressing consistent requirements?

Once the research questions are formulated, it is possible to turn it into null hypotheses to be tested in the experiment:

- $\mathbf{H_{0a}}$ the correctness of acceptance test scenarios specified by participants who attended a three-hour lecture about Fit tables and Gherkin language is the same for both acceptance testing techniques.
- $\mathbf{H_{0b}}$ the effort to specify acceptance test scenarios is the same for both techniques.
- $\mathbf{H_{0c}}$ the correctness of software functionalities specified using Fit tables and Gherkin language and implemented by the participants is the same using both acceptance testing techniques.

On the other hand, the alternative hypotheses are:

- $\mathbf{H_{1a}}$ the correctness of acceptance test scenarios specified by participants who attended a three-hour lecture about Fit tables and Gherkin language is different when both acceptance testing techniques are used.
- $\mathbf{H_{1b}}$ the effort to specify acceptance test scenarios using Fit tables is not the same when Gherkin language is used.
- $\mathbf{H_{1c}}$ the correctness of software functionalities implemented by the participants is different when both acceptance testing techniques are used.

The dependent variables of our study are:

- **ATS#.** Acceptance tests of requirement # were specified: {correctly, incorrectly};
- **ATST#.** The participants need {?} minutes to specify acceptance test scenarios of requirement #;
- **AR#.** The delivered source code implemented based on the acceptance test scenarios of requirement # is executable and it was accepted by the business stakeholder: {yes, no};

Where the symbol "#" represents a change software requirement identified from SR1 to SR4 in Table 1, and the symbol "?" represents an integer value.

Table 1. Objects of the experiment

Id	Requirement
SR1	Rectification of personnel profile information
SR2	Calculation of salary bonus per person
SR3	Exclusion of personnel profile information
SR4	Calculation the average of salary bonus per position

The number of acceptance test scenarios that were specified correctly (*ATS#*) was obtained from the evaluation of artifacts delivery by participants. This evaluation was performed by a researcher who is expert in acceptance tests and he does not have any connection with the experimentation. The time needed to specify acceptance test scenarios of each requirement (*ATST#*) has been measured by asking participants to fill it in the timesheet. The number of requirements correctly coded (*AR#*) was obtained from the evaluation of executable source code delivered by participants. This evaluation was conducted by a business stakeholder, who accepted or not the delivered functionality through black box testing. If the business stakeholder accepts the delivered functionality, the coded requirement is considered correct. Otherwise, it is considered incorrect. This person has not been involved with the specification of acceptance test scenarios that were used by participants to develop the set of required software changes.

4.4 Experiment Design

We divided the experiment into two parts. Part 1 addresses the specification of acceptance test scenarios. Part 2 addresses the implementation of new requirements for the HRS application using the acceptance test scenarios to represent the requirements.

In both parts, we have four objects and two treatments. The objects are the new requirements of the HRS application, as shown in Table 2. The treatments are the following:

- **(F)** Software requirements specified as acceptance test scenarios using Fit Tables.
- **(G)** Software requirements specified as acceptance test scenarios using Gherkin language.

The participants were split into two groups, which were identified by the letters A and B. In Part 1, the group A specified two software requirements as acceptance test scenarios using Fit tables, meanwhile, the group B specified two software requirements as acceptance test scenarios using Gherkin language. Table 2 shows, for each group, which treatment was used to specify which software requirement.

Table 2. Experiment design of Part 1 – Specification of acceptance test scenarios. At the top of this table, SR1, SR2, SR3, and SR4 are abbreviations to the objects listed in Table 1.

Participants	Objects and treatments			
	SR1	*SR2*	*SR3*	*SR4*
Group A	(F)	(F)	–	–
Group B	–	–	(G)	(G)

In Part 2 of this experiment, the set of software requirements specified by group A were send to group B, and vice versa. Then, as shown in Table 3, the group A developed requirements SR3 and SR4, which were specified by group B using Gherkin language, whereas, the group B developed requirements SR1 and SR2, which were specified by group A using Fit tables. Before performing this exchange of acceptance test scenarios between the groups, an expert verified the correctness and conciseness of each scenario. Test scenarios that presented problems were replaced by others, which were correct and express the same set of requirements. This intervention was necessary to prevent false negatives in the analysis of capability of executable acceptance tests to communicate software requirements.

Table 3. Experiment design of Part 2 – implementation of new requirements.

Participants	Objects and treatments			
	SR1	*SR2*	*SR3*	*SR4*
Group A	–	–	(G)	(G)
Group B	(F)	(F)	–	–

4.5 Training

Participants have been trained in meaning and usage of the following subjects:

- a half-hour lecture about acceptance testing, TDD, ATDD, and BDD;
- one-and-a-half-hour lecture about Fit tables and FitNesse, including how to configure this framework and practice exercises;
- one-and-a-half-hour lecture about Gherkin language and Cucumber, including how to configure this framework and practice exercises.

4.6 Experiment Procedure

The experimentation was carried out as explained in the following. First, the participants were given a short introduction to the experimentation, then they were randomly assigned to one of the two groups. After this, they received the timesheet and the questionnaire, described in Sect. 4.2.

Then, the experiment was conduct according to the following steps:

- (1) Participants had 20 min to check if their environment to specify acceptance tests, which was previously set up in the training section, was working. In this step, they also answered the six first questions of the questionnaire.
- (2) Participants read an overview of the HRS application and received a document with the description of two new requirements for this application.
- (3) For each requirement:
- (3.a) Participants filled the start time in their time sheets.
- (3.b) Participants had to understand the requirements; if they had any doubt a business stakeholder was available to clarify them.
- (3.c) Participants had to specify the requirement using the acceptance testing technique assign to them.
- (3.d) When finished, participants had to mark the stop time on their time sheet.
- (4) Then, participants had to answer the next eight questions of the questionnaire and to send the produced artifact to a web repository. The artifacts were identified by a random numeric id and only the researchers knew who uploaded them.
- (5) An expert in acceptance testing evaluated all uploaded artifacts and marked the ones that were inconsistent or incomplete. This mark was visible only to the researchers.
- (6) In the sixth step, the second part of our experiment was started. Participants had 20 min to check if their environment to develop the next tasks was working. After this, they had to download acceptance tests artifacts produced by a participant of the other group.
- (7) Then, participants had to answer two questions in the questionnaire related to their view about the downloaded artifacts.
- (8) The researchers had to verify which participants downloaded acceptance tests that, according to the expert evaluation, were incorrect. Then, they exchanged the incorrect acceptance tests by correct tests that express the same requirements using the same acceptance testing technique.
- (9) Then, for each acceptance test scenario (requirement):
- (9.a) Participants had to fill the start time in their time sheets.
- (9.b) Participants had to understand by themselves the acceptance test.
- (9.c) Participants had to develop the new requirement of the HRS application expressed by the acceptance tests.
- (9.d) When finished, participants had to mark the stop time on their time sheet.
- (10) Then, participants had to answer the next eight questions of the questionnaire and to send the produced source code to a web repository. The artifacts were identified by a random numeric id, and only researchers knew who uploaded them.

This procedure was carried out in two sections. The first with three and a half-hour of duration and the second one with two hours of duration.

5 Results and Data Analysis

In this Section, we show and discuss the results achieved from the experiment. The experiment data and charts are available at www.leb.inf.ufsc.br/index.php/xp2018/.

5.1 Consistency and Correctness of the Acceptance Test Scenarios

Table 4 is the contingency table for the dependents variables (see Sect. 4.3) from **ATS***SR1* to **ATS***SR4*. The first line of this table shows the number of tasks performed using Fit tables as acceptance testing technique: 17 tasks were completed, and only one task failed. The second line shows the number of tasks performed using Gherkin language as acceptance testing technique: 13 tasks were completed, and five tasks failed. The tasks performed by the same participant were considered as independent measures.

We applied the Fisher's test in the data presented in Table 4. This test returned a p-value of 0.1774. The result is not significant at $p < 0.05$. Thus, $\mathbf{H_{0a}}$ is accepted, there is no statistically significant influence of the treatment on the acceptance test scenarios specification.

Table 4. Contingency table for correct specifications of acceptance tests

	Acceptance test scenarios were specified:	
Treatment	Correctly	Incorrectly
Fit Tables (T)	17	1
Gherkin Language (G)	13	5

Although we cannot obtain the answer of RQ1 with Fisher's test, we suppose that extra training sections and practical exercises could decrease the number of incorrect specification around zero because the errors identified in the test scenarios specified by participants in both techniques are basic mistakes. Thus, we found that the complexity to learn both acceptance testing techniques by software developers is the same.

5.2 Time to Complete Acceptance Test Scenarios Specifications

Table 5 presents the time, in minutes, spent by participants to complete the specification task of each requirement. The underlined time values in this table refer to test scenarios that were specified incorrectly by participants. The tasks performed by the same participant were considered as independent measures, and the distribution of requirements and treatments were conducted as shown in Table 2.

We used the Shapiro-Wilk normality test to check if the data collected from the experiment for the two treatments have a normal distribution. Then, we performed a

Table 5. A list of time spent to develop new software requirements in the HRS application.

Treatment	Time list (in minutes)
Fit tables (F)	{20, 21, 21, 23, 35, 36, 40, 45, 50, 65, 66, 77, 79, 88, 108, 120, 126, 135}
Gherkin Language (G)	{15, 16, 17, 15, 39, 30, 30, 30, 45, 35, 60, 28, 40, 40, 70, 57, 84, 75}

t-test that returned a p-value of 0.0291. We also performed the same test excluding the underlined values and we obtained a p-value of 0.0334. The results are significant at $p < 0.05$. Thus, in both tests $\mathbf{H_{0b}}$ is rejected and the alternative hypotheses are accepted. Therefore, there is a difference, in terms of the mean time spent to specify acceptance tests, between Fit tables and Gherkin language.

Answering the RQ2, we found that the effort, in terms of the meantime, to specify acceptance tests using Gherkin language (40 min) is lower than using Fit Tables (64 min).

5.3 Applicability of Acceptance Test Scenarios to Communicate Software Requirements

Table 6 is the contingency table for the dependents variables (see Sect. 4.3) from ARSR1 to ARSR4. The first line of this table shows the number of tasks implemented based on the requirements specified using Fit tables: 13 tasks were successfully completed, and five tasks failed. The second line shows the number of tasks using Gherkin Language: 10 tasks were successfully completed, and eight tasks failed.

We applied the Fisher's test in the data presented in Table 6. This test returned a p-value of 0.4887. The result is not significant at $p < 0.05$. Thus, H0c is accepted, therefore, there is no statistically significant influence of the treatment on the development of software requirements expressed by acceptance test scenarios using both techniques.

Table 6. Contingency table for correct development of software requirements expressed by acceptance tests

	The delivered source code implemented based on the acceptance test scenarios is executable and it was accepted by the business stakeholder:	
Treatment	Yes	No
Fit Tables (T)	13	5
Gherkin Language (G)	10	8

Then, addressing the RQ3, we cannot assume based on the Fisher's test result that a technique is better than another to communicate requirements. In addition, in the same way that we suppose that extra training could improve acceptance test scenarios specification, it also could improve requirements communication. However, despite these experimentation evidences, we claim that Gherkin language scenarios communicate requirements better than Fit tables because we observe that tables are weak in details and depending on the software requirement a complementary textual description is required to communicate a requirement completely, whereas, in Gherkin language the acceptance test scenarios are complemented on default by a textual description.

5.4 Experiment Questionnaire

In this section, we discuss six questions of the questionnaire that we applied in the experiment. The questions were answered on a five-point scale, where one maps to Strongly agree, two maps to Agree, three maps to Not certain, four maps to Disagree, and five maps to Strongly disagree. Question 1 (Q1) and question 3 (Q3) were applied to group A, whereas question 2 (Q2) and question 4 (Q4) were applied to group B. Questions 5 (Q5) and 6 (Q6) were applied to both groups.

Q1. I experienced no difficulty in specifying acceptance test scenarios using Fit tables. Half of the participants strongly agree (22.22%) or agree (27.78%) with this statement and 22.22% are not certain, whereas, the rest of the participants disagree (5.56%) or strongly disagree (22.22%).

Although the results presented in Sect. 5.1 shows that the major part of the acceptance tests was specified correctly, we observed, through this questionnaire, that participants had difficulty to specify the acceptance tests. So, we realized that in a next experiment we should dedicate more time performing training lectures intending to decrease the time spent by participants to specify acceptance tests and to increase the quality of acceptance test scenarios using Fit tables.

Q2. I experienced no difficulty in specifying acceptance test scenarios using Gherkin language. This result was different than we expected. The percentage of participants who answered that they strongly disagree (16.67%) or disagree (16.67%) with this statement is greater than the percentage of who answered that strongly agree (5.56%) or agree (16.67%). The rest of participants (44.44%) are not certain about this statement.

Our initial belief was that participants who used Gherkin language had experimented less difficulty to create acceptance test scenarios than ones that used Fit tables because Gherkin language is similar to English spoken language. However, the results obtained from the questionnaire tend to be the opposite. As concluded in Q1, the participants should spend more time with the lecture and exercises to improve their experience with Gherkin Language.

Q3. I experienced no difficulty in implementing new requirements in the HRS application, which specification were expressed as acceptance test scenarios writing through Fit tables. The major part of participants reported that they had difficulty in the implementation tasks, 55.56% answered that strongly disagree

(27.78%) and disagree (27.78%) with this assertion, whereas 38.89% of the participants are not certain, 5.56% agree and 0.00% strongly agree with this assertion.

Q4. I experienced no difficulty in implementing new requirements in the HRS application, which specification were expressed as acceptance test scenarios writing through Gherkin Language. The major part of participants reported that they had difficulty in the implementation tasks, 27.78% and 33.33% answered respectively that strongly disagree and disagree with this assertion, whereas, 27.78% of the participants are not certain, 5.56% agree and 5.56% strongly agree with this assertion.

Although only a few participants agree or strongly agree with this assertion, the percentage is two times bigger than the percentage of the same group in Q3. We assign this to the fact that acceptance test scenarios written in Gherkin language are more verbose than tables, which becomes Gherkin language easier to understand than Fit tables. However, as presented in Sect. 5.3, there is no evidence that one technique is better than another to communicate software requirements.

Q5. I will use acceptance testing to validate software in future projects. The number of participants (27.78%) that agree with this assertion is greater than the number of participants that disagree (11.11%) or strongly disagree (11.11%). However, 50.00% are in doubt about using acceptance testing to validate software.

Q6. I will use acceptance test scenarios to communicate requirements for future projects. 33.33% of the participants did not approve the use of acceptance tests as requirements and they would not like to take part in projects that use this approach. 38.89% of the participants are not certain about this assertion, the rest of the participants, 27.78%, agree with this assertion.

In both Q5 and Q6, the number of participants that are in doubt about using acceptance testing to validate software or to communicate requirements is greater than the number of participants that are certainly that will use or will not use it in the future. We think that the high number of participants that are in doubt is due to the inexperience in acceptance testing, which is in agreement with the background questionnaire where 100% of participants answered they had never seen it before.

6 Threats to the Validity

Although we assigned different requirements to groups A and B in the same part of our experiment, we choose requirements that have similar business logic and complexity. However, the complexity of the requirements could affect results, mainly regarding time.

An expert in acceptance testing verified the artifacts produced by participants in part 1 and a business stakeholder verified the artifacts produced in part 2. These two individuals had an important role in our experimentation because they decided what is or not correct. However, we could carry out our experiment without these individuals in a different way:

- using the acceptance tests specified by an expert as input for part 2, avoiding that some mistakes in the acceptance test scenarios created by other participants were unnoticed by the expert;
- using automated acceptance tests, or even JUnit tests, to validate if the code implemented by the participants meets the user needs.

However, we choose this approach to approximate our experiment to a real-world scenario, where there is a variation on the style of acceptance tests scenarios written, such as vocabulary, idioms, and level of details.

Another issue is the time sheets. It is very difficult to ensure that all participants are marking the time spent in each task. During the experiment, we checked if the forms have been filled correctly and asked the participants to fill out the forms very carefully. Finally, the small sample size may limit the capability of statistical tests. In this study, the time was compared using t-test, and for contingency tables, we used Fisher's exact test.

7 Conclusion

In this study, we have experimented to compare the applicability of acceptance tests, which were written using Fit tables and Gherkin language, as software requirements. The results show that there is no sufficient evidence to affirm that one technique is easier to use than another or one technique communicates software requirements better than another. Whereas, the comparison of effort regarding time to specify acceptance testing shows that the mean time to specify test scenarios using Gherkin Language is lower than using Fit tables.

Additionally, the questionnaire applied shows that participants had difficulty to specify and understand acceptance tests writing in both techniques. We assign this difficulty because neither of the participants had used Fit tables and Gherkin language before. Despite only a few participants answered that is easy to understand requirements expressed by acceptance tests, they have pointed out Gherkin language scenarios as easier to understand than Fit tables.

Finally, the number of participants who agreed with the possibility of using these acceptance testing techniques as software requirements in future projects is very similar to the numbers of those participants who disagree with this possibility. We assign this result to the participants' inexperience in acceptance testing, which resulted in a poor impression about the application of these techniques in real-world projects. As future works, we intend to improve our experimental design to carry it out with others acceptance testing techniques and include other personas like non-technical users and software engineers.

References

1. Sarmiento, E., Leite, J.C.S.P., Almentero, E.: C&L: Generating model-based test cases from natural language requirements descriptions. In: 2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET), pp. 32–38 (2014)
2. Sommerville, I.: Software Engineering. 9th edn. Pearson Education, Boston (2015)
3. Torchiano, M., Ricca, F., Penta, M.D.: "Talking tests": a preliminary experimental study on fit user acceptance tests. In: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), pp. 464–466 (2007)
4. Ricca, F., Torchiano, M., Penta, M.D., Ceccato, M., Tonella, P.: Using acceptance tests as a support for clarifying requirements: A series of experiments. Inf. Softw. Technol. **51**, 270–283 (2009)
5. Ricca, F., Torchiano, M., Penta, M.D., Ceccato, M., Tonella, P.: On the use of executable fit tables to support maintenance and evolution tasks. In: Third International ERCIM Symposium on Software Evolution, pp. 83–92 (2007)
6. Clerissi, D., Leotta, M., Reggio, G., Ricca, F.: A lightweight semi-automated acceptance test-driven development approach for web applications. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 593–597. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38791-8_55
7. Melnik, G., Maurer, F.: The practice of specifying requirements using executable acceptance tests in computer science courses. In: Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, pp. 365–370. ACM, San Diego (2005)
8. Wanderley, F., Silva, A., Araújo, J.: Evaluation of BehaviorMap: a user-centered behavior language. In: 2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS), pp. 309–320 (2015)
9. Longo, D.H., Vilain P.: Creating user scenarios through user interaction diagrams by non-technical customers. In: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, pp. 330–335 (2015)
10. IEEE: IEEE Standard for Software Verification and Validation Plans. IEEE Std 1012-1986. IEEE (1986)
11. Mugridge, R., Cunningham, W.: Fit for Developing Software: Framework for Integrated Tests. Pearson Education, Upper Saddle River (2005)
12. Rose, S., Wynne, M., Hellesøy, A.: The Cucumber for Java Book: Behaviour-Driven Development for Testers and Developers. 1st edn. Pragmatic Bookshelf (2015)
13. Beck, K.: Test-Driven Development: By Example. Addison-Wesley Professional, Boston (2003)
14. Deng, C., Wilson, P., Maurer, F.: FitClipse: a fit-based eclipse plug-in for executable acceptance test driven development. In: Concas, G., Damiani, E., Scotto, M., Succi, G. (eds.) XP 2007. LNCS, vol. 4536, pp. 93–100. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73101-6_13
15. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering – An Introduction. Springer, Heidelberg (2012)
16. Juristo, N., Moreno, A.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers, Boston (2001)