Check for updates

Received: 31 January 2022 | Revised: 31 January 2023 | Accepted: 1 March 2023

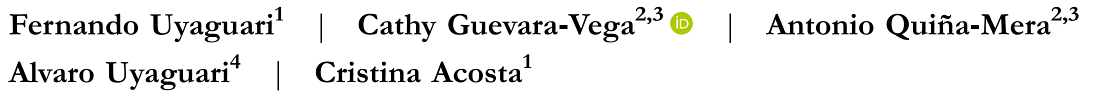
IET Software

DOI: 10.1049/sfw2.12124

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

Effect of requirements specification using native language on external software quality



Fernando Uyaguari[1] | Cathy Guevara-Vega[2,3] | Antonio Quiña-Mera[2,3] | Alvaro Uyaguari[4] | Cristina Acosta[1]

[1]Software Development Department, Instituto Superior Tecnológico Wissen, Cuenca, Azuay, Ecuador

[2]eCIER Research Group, Universidad Técnica del Norte, Ibarra, Imbabura, Ecuador

[3]SCORE Lab, Universidad de Sevilla, Seville, Spain

[4]Software Department, Universidad de Las Fuerzas Armadas - ESPE, Sangolquí, Pichincha, Ecuador

Correspondence

Cathy Guevara-Vega.
Email: cguevara@utn.edu.ec

Funding information

Universidad Técnica del Norte

Abstract

In the context of requirements specification in Global Software Development, aspects such as differences in culture, language and schedule affect software development teams; however, we do not know the effect of these issues. Compare the native language requirements with the foreign language requirements concerning external quality. We conducted a controlled experiment of one-factor two treatments within-subjects with 17 experimental subjects. Wilcoxon test indicates that there is evidence to reject the null hypothesis (p-value = 0.008); there is a statistically significant difference. The external quality value obtained with native language requirements is superior to the external quality produced with the foreign language. The effect size equals an absolute value of 0.45, which corresponds to a medium effect. The language used in the requirements specification influences the external quality; using the native language in the requirements specification significantly increases the external quality. The result obtained in this research should be considered to evaluate the roles and English language skills of GSD team members and their effect on external software quality. We also suggest considering the English language skills of the experimental subjects in the experimental laboratories since language could influence the results of the experiments.

KEYWORDS

software engineering, software quality

1 | INTRODUCTION

Globalisation demands constant challenges in the software development process; one of them is software requirements management [1]. Requirements Engineering (RE) is responsible for managing, discovering, analysing, documenting and verifying software requirements [2], which involves the joint work between several professionals fulfiling different roles in the organisation. Therefore, communication, understanding and documentation of requirements are fundamental to carrying out this initial phase [3]. RE is a complicated process in itself, but it is even more so when groups of professionals specify requirements across cultural, linguistic and time boundaries; these aspects impact requirements elicitation and validation by affecting knowledge transfer to developers [4, 5].

In the scientific literature, on the one hand, we observe that Global Software Development (GSD) teams use a common language, usually English. Still, problems arise in spoken and written communication [6]. Moreover, team members have differences in their language skills in a language other than their native language, which leads to misunderstandings between team members, and its effect on the software product is unknown [7, 8]. On the other hand, we found a controlled experiment where the author in Ref. [9] studies the impact of native language on the quality and correctness of used case

Fernando Uyaguari, Cathy Guevara-Vega, Antonio Quiña-Mera, Alvaro Uyaguari and Cristina Acosta contributed equally to this work.

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2023 The Authors. *IET Software* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

modelling and determines that native language helps to improve the functional correctness of a used case model.

In this study, the authors studied Requirements Engineering in the context of GSD, where the participants are in different countries or continents [10, 11], where we identify problems related to cultural, linguistic and temporal differences. Still, these aspects are not addressed in greater depth. For example, the impact of issues related to language differences on the software product is unknown. Therefore, in our study, we are motivated to analyse language's impact on software's external quality through a controlled experiment [12].

We report this experiment according to the guidelines for reporting experiments in Software Engineering by Jedlitschka et al. [13]. The rest of this document is organised as follows: Section 2 presents the existing literature related to our research; Section 3 establishes the experimental design; Section 4 explains the experimental execution; Section 5 defines the experimenalt results and analysis; Section 6 describes the actions taken to counter threats to validity; Section 7 presents the discussion; Section 8 explains the conclusions; Section 9 offers the future work; and Section 10 provides the replication package for this research.

2 | RELATED WORK

2.1 | Requirements in Global Software Development

Global Software Development is the model of software development in a global context, where participants are located in different countries or continents, whose differences include culture, language, time zone and other factors [10, 11]. In this sense, these differences challenge RE in GSD projects [4, 5].

In the scientific literature, we find several reviews that address the problem of the difficulties of RE in GSD [7, 14–16], where we highlight the main issues related to the context in which it develops:

- Language
- User interface
- Local standards
- Laws and regulations
- Cultural differences
- Regional differences
- Problems related to the education and work context
- Environmental conditions

In addition, the studies identify essential risks that influence some aspects of RE in the GSD, such as knowledge sharing, customer relationships, supplier relationships and socio-cultural communication problems [7, 14]. But they do not focus on each particular aspect, do not evaluate the impact and do not provide concrete solutions. Instead, they show some recommendations, such as using a common language, creating intensive communication and directing communication from the beginning of the project.

2.2 | The requirements language in GSD

Natural Language (NL) is complex and usually generates diverse interpretations from the people involved [17]. Some of the difficulties originate from ambiguities (i.e., when a statement can have different interpretations); vagueness (i.e., when a statement can have a continuum of interpretations); and uncertainty (i.e., when a statement weakens its truth proposition). These interpretations are considered phenomena that affect the comprehension and understanding of the meaning of information [18]. In the RE domain, these phenomena are generally grouped in the class of ambiguity [19]. Tjong & Berry in 2013 explain that ambiguity in RE has dangerous consequences for a software development project. If requirements are misapplied or misunderstood, they cause high rework costs and delays in product releases [19].

RE is a challenge that becomes even more complicated when team members do not have a common language. This problem even occurs within some countries with more than one language, as is the extreme case of India, which has more than 20 languages and 11 scripts [20]. In this regard, Derkacz et al. in 2018 identified problems in the comprehension and understanding of people searching, documenting and analysing information in a language other than their native language [8].

Communication is one of the biggest challenges in distributed software development; misunderstandings are a source of problems [21]. In addition, team members may not be able to communicate with each other using their native languages, and the language barrier may prevent proper requirements gathering [22, 23]. The lack of a common language between a customer and a supplier can increase the chances of a lack of understanding. GSD teams address the language issue by introducing a reference language, usually English. But it should be noted that using a non-native language will generally create significant overhead for the people [7].

English is the language used worldwide to communicate with people; however, most people have a limited level of understanding [24, 25]. When the software requirements engineering team involved does not have the same English language proficiency (language skills), it affects the understanding of the requirements [26] and could impact the software product. In addition, language skills can impede effective communication; when parties to the communication have different levels of competence, the stronger party occupies a more powerful position and can suppress transmission through unintentional intimidation [27].

While there are studies that indicate that language is a source of problems in RE, there are also studies such as in Ref. [6], which presents a case study where it analyses the cultural aspects between two different countries (Germany and China) and indicates that all team members overcame some language barriers by communicating adequately in English; however, written communication resulted in misunderstandings due to ambiguities in using the English language for both groups. On the other hand, the authors [27] indicate that team members who are unsure of their English language skills could choose another means of communication such as email instead of

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

telephone or videoconferencing; however, they had problems with ambiguities as the authors in Ref. [6] point out. Likewise, the authors of Ref. [28] report problems when transferring to Outsourcing Projects from countries where English is not the native language. Misunderstandings occur as team members tend to translate information into their native language, which can lead to ambiguities when words have multiple meanings.

Culture is another essential aspect addressed by the authors in understanding RE processes, including language and communication style. For example, the use of homonyms ('overlooking the development of a requirement' means overseeing the implementation of the requirement; for others, it means ignoring the requirement) could occur even in two English-speaking stakeholder groups. [5]. The author also indicates that project team members are very likely to introduce misinterpretations of stakeholder requirements at each level of communication due to cultural factors such as language or communication styles. In the GSD domain, Šmite et al. in 2014 consider that team members tend to use their preferred terms; in that sense, they state the importance of making a global taxonomy for software engineering [29].

After performing a literature review in scientific databases ACM, Elsevier, Springer,and IEEE, we observed that several authors address the difficulties of RE in GSD and the difficulties of using a common language differently. For example, we found a study performed in a controlled experiment that concludes that native language helps to improve the functional correctness of a used case model [9]. However, we found no studies that address the effect of requirements language on external software quality. Therefore, in this study, we propose to perform a controlled experiment comparing native language and foreign language (English) development to observe the impact of language on the quality of software developed in the GSD environment. From the results of this experiment, we analyse the importance of language in RE and make suggestions.

3 | EXPERIMENTAL DESIGN

3.1 | Research objectives

To define the research objective, we use Basili's Goal-Question-Metric (GQM) approach [30], where we consider

'Analyse *language of the requirements in the native language and foreign language* for the purpose of *comparing them* with respect to *external quality* from the point of view of the *researchers* in the *academic* context.'

Therefore, we set the following research question:

- RQ: Does native language software requirements increase external quality compared to foreign language software requirements?

3.2 | Factors and treatments

The factor is the language of the requirements.

The treatments applied are:

- Requirements in the native language (Spanish language).
- Requirements in the foreign language (English language).

The native language of the subjects is Spanish, also called Castilian. Spanish is the official language of the country where we conducted the experiment.

The foreign language considered is English. According to the Common European Framework of Reference for Languages, the experimental subjects have an A2, B1 and B2 level of English.

3.3 | Variables

The independent variable is the language of the requirements; the levels are foreign language (English) and native language (Spanish). We specify the requirements of the experimental tasks using NL. That is, we do not use formal language. Some studies indicate that NL is widely used because people understand NL better than formal languages such as UML[1] [32]. We use two experimental tasks, and each task has two specifications, one in the native language and one in the foreign language.

The dependent variable or response variable is the external quality of the software (QLTY). Software quality is the capability of the software product to enable system behaviour that satisfies stated and implied needs when the system is under specific conditions [33]. ISO/IEC 25010:2011 defines the product quality model that combines previous internal and external quality models through a set of characteristics and sub-characteristics [34]. In this study, we use the external quality model; the characteristic of interests is 'functionality,' which refers to the degree to which a product or system provides functions that meet stated or implied needs when used under specified conditions. Functionality has the sub-characteristics presented in Table 1.

The sub-characteristic *functional completeness* refers to how much of the expected functionality has been delivered [35]. This sub-characteristic is not interested in whether the provided functionality is correct or appropriate; it is only interested in whether the functionality is present in the product. The sub-characteristic *functional correctness* replaces the sub-characteristic *accuracy* of the previous ISO 9126 standard [35]. Finally, the third *functional appropriateness* sub-characteristic refers to the fact that the software product is not the end itself. It supports users in doing their actual jobs. The software helps to get the job done faster and more reliably.

We measure the external quality of the software as a combination of functional completeness and functional correctness. In order to calculate the total quality, we use the following Equation (1):

[1]UML: Unified Modeling Language, is a visual language for specifying, constructing, and documenting the artifacts of systems [31].

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 1 Composition of functionality according to ISO/IEC 25010:2011 [34].

Sub-characteristic	Definition
Functional completeness	Degree to which the set of functions covers all the specified tasks and user objectives.
Functional correctness	Degree to which a product or system provides the correct results with the needed degree of precision.
Functional appropriateness	Degree to which the functions facilitate the accomplishment of specified tasks and objectives.

$$QLTY = \frac{\#Assert(PASS)}{\#Assert(ALL)} \times 100 \quad (1)$$

where *#Assert*(*PASS*) represents the total number of assertions passing in the acceptance test suite, and *#Assert*(*ALL*) represents the total of assertions of the acceptance test suite of the experimental task. The authors developed the acceptance test suite which is hidden from the participants.

In other words, QLTY represents the correctness (in percentage) of the code produced corresponding to the tasks addressed by all participants. When tests failed due to trivial errors in the code, we corrected them. We consider a passed assertion the smallest quantifiable evidence of work performed. Therefore, it is the unit of work adopted to measure QLTY. We based on the following studies [36–40] because of the relationship of human-oriented experiments in the Software Engineering area.

3.4 | Hypotheses

Based on the **research question**, we have the following **hypotheses** of the experiment:

- H_0 (Null hypothesis): There is no difference in the effects of requirements language on external quality.
- H_1 (Alternative hypothesis 1): There is a difference between the effects of requirements language on external quality. The quality produced by the native language requirements is higher than the quality produced by the foreign language requirements.
- H_2 (Alternative hypothesis 2): There is a difference between the effects of requirements language on external quality. The quality produced by the native language requirements is lower than the quality produced by the foreign language requirements.

3.5 | Design

The experimental design is one-factor, two-level within-subjects. Each experimental subject applies two treatments: development with native language requirements and development with foreign language requirements.

TABLE 2 Within-subjects experiment design.

Session	Treatment	Task	# Subjects
Session 1	Native language	Tennis game	8
		Bowling scorekeeper	9
Session 2	Foreign language	Tennis game	9
		Bowling scorekeeper	8

In Table 2, we present the experimental design. We have a group of 17 experimental subjects to which we apply the two treatments in two sessions carried out on the same day. We should obtain 34 samples because the experimental design is within subjects; the subjects apply two treatments.

According to the within-subjects design, we need to use two experimental tasks, one for each treatment. The experimental tasks are shown below.

3.6 | Experimental tasks

The experimental tasks are Bowling Scorekeeper and Tennis game. In the following, we present each task.

3.6.1 | Bowling Scorekeeper (BSK)

BSK is an experimental task based on Robert Martin's Bowling Scorekeeper [41]. In BSK, a frame is composed of two shots. A game is made up of a set of frames. The objective of the task is to calculate the score of the game.

Subjects are given the complete specification and a source code template with the names of the classes and methods. The main functionalities of BSK are the following:

- Add a frame
- Add a bonus
- Identify when a frame is a spare
- Identify when a frame is a strike
- Calculate a frame score
- Calculate the score of the game

All operations are essential. It should be noted that the calculation of the game score is the one that demands the most effort since it depends on the type of frame (regular, spare and strike) and the position of the frame in the game and whether the next frame is a strike.

3.6.2 | Tennis game

The tennis game has two players, player A and player B. Player's points are received in a character string, for example, game ('AAB'), game (' '), game ('ABABABA').

Player scores range from zero to three points, described as 'zero,' 'fifteen,' 'thirty,' and 'forty.' The score is written in the

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

format ('player A points,' 'player B points') such as: 'zero, zero,' 'fifteen, zero,' and 'forty, fifteen.'

The rules of the Tennis game are relatively simple:

- If each player has scored at least three points and the scores are equal, the score is "deuce."
- If at least three points have been scored from each side and a player has one more point than his opponent, the score for the game is "advantage" for the winning player. For example, "Advantage Player A."
- A game is won by the first player who has scored at least four points in total and at least two points more than the opponent, then 'player A wins' or 'player B wins.'

In Annexes A, B, C and D, we complete the information on the software requirements presented to the experimental subjects in both native language and foreign language.

We could note that Bowling Score Keeper is the most difficult task. The treatments were applied using both experimental tasks randomly to avoid bias in the experimental task in the results. See Table 3.

3.7 | Subject

The experimental subjects are 17 students of the Master's Degree in Software Engineering at Universidad de las Fuerzas Armadas—ESPE, in Latacunga, Ecuador, who took the course "Experimentation in Software Engineering."

TABLE 3 One factor, two treatments within-subjects design.

Subject	Foreign language treatment	Native language treatment
1	Bowling	Tennis
2	Tennis	Bowling
3	Tennis	Bowling
4	Bowling	Tennis
5	Bowling	Tennis
6	Bowling	Tennis
7	Bowling	Tennis
8	Tennis	Bowling
9	Tennis	Bowling
10	Bowling	Tennis
11	Tennis	Bowling
12	Tennis	Bowling
13	Tennis	Bowling
14	Bowling	Tennis
15	Tennis	Bowling
16	Tennis	Bowling
17	Bowling	Tennis

To ensure proper involvement in the experimental task, the students were told that they would receive extra points to their final grade.

The subjects did not know the objective of the experiment, and neither did the treatments to avoid biases. This activity contributed to the learning of the students since they observed the execution of an experiment from the point of view of experimental subjects.

3.8 | Instrumentation

We used the NetBeans version 8.2 integrated development environment (IDE), and the development language was Java. The experimental subjects had previously installed the development environment on their personal computers.

The researchers developed a template in Java language for each experimental task, which was copied to the computers of the experimental subjects prior to starting the development of the assigned experimental task.

The specification of requirements in the native language and foreign language were developed by the experimenters, printed and given to the experimental subjects at the beginning of the resolution of the experimental task.

We requested specific information from the coordination of the Master's Degree in Software Engineering—ESPE Latacunga—to know the level of English and the professional degree of the experimental subjects.

We obtained the external quality measure through a suite of acceptance test cases developed by a master's student in Software Engineering at the Universidad Técnica del Norte, Ibarra, Ecuador, under the supervision of the researchers. We implemented the test cases in the JUnit testing framework.

3.9 | Data collection

Table 4, shows the schedule planned for the experiment. In session one, we apply the native language treatment, and in session two, we apply the foreign language treatment.

The researchers, authors of this paper, supervised the execution of the experiment, collecting the source code of the experimental subjects and measuring the variable response of the experiment.

TABLE 4 Experiment schedule.

Time	Activity
08h00-08h15	Instructions
08h15-10h00	Session 1
10h00-10h15	Break time
10h15-12h00	Session 2
12h00-12h10	Filling the form
12h10-12h40	Source code collection

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

We obtained the external quality measure using a suite of acceptance test cases for the Bowling and Tennis game tasks. Table 5 shows the total number of test cases for each experimental task. Annexes E and F show the test cases that are used to obtain the external quality measure.

To extract the quality measure, the researchers loaded the source code of the experimental subjects into the development environment and then loaded the suite of test cases and executed the test cases. The result of the execution of each test case was recorded in an Excel sheet. This procedure was applied to obtain the quality measure of each task for each subject.

3.10 | Analysis

The statistical analysis of the experiment was carried out by using IBM SPSS Statistics Version 22, the normality of the data was obtained and the Wilcoxon test was applied for the hypothesis test.

4 | EXPERIMENT EXECUTION

This section presents a concise explanation of the context and setting in which the experiment was conducted. It also describes the selected sample, that is, the number of experimental subjects who participated and their level of English skills. In addition, it details the preparation tasks the participants followed before the experiment and the data collection process of the tasks performed by the participants.

The experiment was carried out on Saturday, 9 March 2019, on a cold morning in the city of Latacunga, located in the Andean mountain range at the height of 2800 m above sea level.

4.1 | Sample

At the time of the experiment, the 17 experimental subjects were students of the Experimentation in Software Engineering course of the Master's in Software Engineering programme who attended face-to-face classes on Fridays and Saturdays; most of the subjects lived in the same city, where the experiment was conducted. Thirteen experimental subjects are men and four are women. Fifteen subjects are professionals in Computer Systems Engineering, and two subjects are Software Engineers. The total number of experimental subjects has their profession related to the area of Computer Science.

TABLE 5 Number of test cases per experimental task.

Experimental task	Test cases
Bowling scorekeeper	32
Tennis game	49

Regarding knowledge of the foreign language (English), eight subjects have B2 level knowledge of English, seven subjects A2 level, and two subjects B1 level. All subjects have knowledge of Java programing and professional experience of at least 3 years applied in industry or academy.

4.2 | Preparation

The subjects attended the experiment on time. In the beginning, we gave the subjects instructions for the experimental tasks that they had to carry out.

We ran the experiment according to the established schedule. The experimenters who carried out the experiment attended to queries from the experimental subjects regarding clarifications of the activity. The researchers avoided answering queries regarding the resolution of the experimental task so as not to interfere with the experiment.

Although the time assigned to develop each experimental task was 1 h 45 min, some subjects submitted the source code before that time. The break time was taken inside the same classroom individually so that the subjects did not meet with each other to avoid threats to the validity of the experiment.

4.3 | Data collection

At the end of the experiment, the researchers obtained the source code from the experimental subjects, where the subjects sent the source code to an email address, and the experimenters varied its reception; in some cases, it was necessary to obtain the code directly from the computer of the experimental subject. We collected the source code in a local repository; this activity lasted 1 h, a little longer than the expected time.

5 | RESULTS

This section presents the descriptive analysis of the external quality response variable of the software, a box plot and the results of the hypothesis test that report the statistical significance and the result of the experiment.

5.1 | Descriptive statistics for QLTY

In Table 6 we present the central values (mean along with its 95% confidence interval, trimmed mean and median), dispersion (variance, standard deviation, minimum, maximum, range and interquartile range) and symmetry (skewness and kurtosis) for the QLTY metric for native language and foreign language.

The mean quality for the native language treatment is 14.76 with a confidence interval between 4.53 and 25.00. The mean for the foreign language treatment is 6.65, with a confidence interval between −0.50 and 13.79. The external quality of the software obtained with native language requirements is superior to the external quality of the software obtained with foreign

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

language requirements. The minimum external quality value obtained is the same for both treatments (0). The maximum value of the external quality is higher with the treatment of the native language (65) compared to the maximum value of the foreign language (50).

The variance of the value of the external quality of the native language treatment (396.44) is wide compared to the foreign language treatment (193.11).

Figure 1 shows the box plot, where outliers are observed in the treatment of foreign language and their values below native language reduce the range.

TABLE 6 Descriptive statistics for QLTY.

	Native language	Foreign language
Media	14.76 (4.82)	6.65 (3.37)
Lower bound[a]	4.53	−0.50
Upper bound[a]	25.00	13.79
5% trimmed mean	12.79	4.61
Median	3.00	0.00
Variance	396.44	193.11
Standard deviation	19.91	13.89
Minimum	0	0
Maximum	65	50
Range	65	50
Interquartile range	31	4
Skewness	1.24	2.56
Kurtosis	0.70	6.19

[a]Lower bound and upper bound for 5% Trimmed Mean.

Graphically, we observe that the quality obtained from the software with the requirements in the native language is superior to the quality obtained with a foreign language.

5.2 | Hypothesis testing for QLTY

This subsection presents the hypothesis test for the one-factor, two-level within-subjects experiment.

The data normality test has been performed using Kolmogorov—Smirnov. The significance value is 0.000, therefore, it can be stated that the sample does not meet the normality condition. The Wilcoxon non-parametric test has been applied to contrast the hypothesis with related measures since the experimental subjects have applied both treatments.

The result of the Wilcoxon hypothesis test indicates that there is sufficient evidence to reject the null hypothesis (p-value = 0.008). Given that the p-value is lower than 0.05, there is a statistically significant difference. The value of the external quality obtained by the native language requirements is superior to the quality produced by the foreign language requirements, therefore, the alternative hypothesis H_1 is accepted.

This result affirms that the language in which the requirements are written influences the external quality of the software.

5.3 | Effect size

Regarding the standardised effect sizes in software engineering experiments, it is suggested that an absolute effect size of 1.4 indicates a large effect, 0.6 indicates a medium effect and 0.17 indicates a small effect [42]. We calculated the effect size of the

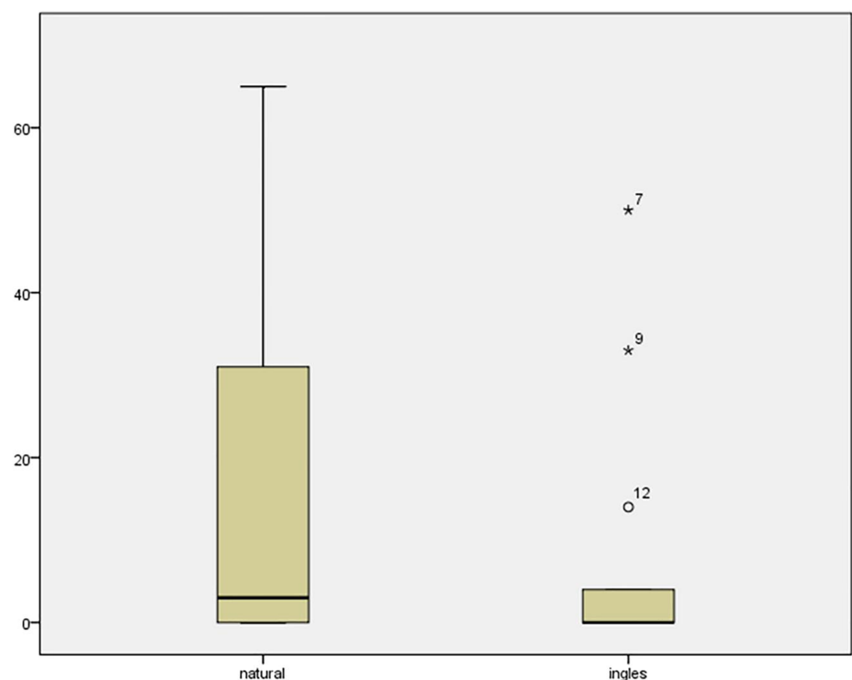


FIGURE 1 Box diagram.

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

experiment which corresponds to an absolute value of 0.45; therefore, it corresponds to a small to medium effect.

5.4 | Interpretations

After conducting the experiment, we found that the quality of the software developed with requirements written in the native language far exceeds that of the software developed with the requirements written in a foreign language. In terms of understanding or comprehension of the requirements, we observed that the requirements written in a foreign language represent a greater difficulty to develop the experimental tasks than the requirements presented in the native language.

We show that English-level subjects have limited their understanding or comprehension in the analysis of the requirements in a foreign language. When working with the native language, the external quality of the software is increased. That is to say that the language of the requirements influences the quality of the software product.

6 | THREATS TO VALIDITY

6.1 | Conclusion validity

To ensure the conclusions derived from the analysed data, we have taken the following measures:

- We selected two experimental tasks whose specifications can be written in the native language and in a foreign language.
- To increase the number of samples, we defined a within-subjects design for each subject to apply both treatments.

6.2 | Internal validity

To ensure the conclusions derived from the measurements, we considered the following aspects:

- We performed all measurements with the source code created by the subjects during the experiment; we did not reuse any source code created outside of the experiment.
- The measurer was a master's student in the Software Engineering Programme at the Universidad Técnica del Norte, in Ibarra, Ecuador who carried out the measurements under the supervision of the authors of this research.
- The measurer acted independently, avoiding bias since he was unaware of the objectives of the experiment.

6.3 | Construct validity

To ensure that the conclusions obtained are connected to the operationalisation of the dependent and independent variables and that the variables represent the constructs correctly, we took into account the following considerations:

- For the operationalisation of the response variable, we adhered to the ISO/IEC 25000 software product quality standard.
- We elaborated the suite of acceptance test cases to measure the response variable of this experiment.

6.4 | External validity

To ensure the generalisation of the research results to other populations, we carry out the following actions:

- The experimental subjects are master's level students in the Software area, mostly with professional experience.
- The experimenters who designed and executed this experiment are subjects with experience in industry, academia and currently in research.

7 | DISCUSSION

In the context of requirements engineering in the globalisation of the software industry, the literature refers to aspects such as cultural boundaries, language and time zones that could influence software products. In this research, we studied language and its impact on the external quality of the software. Through this experiment, we evidenced problems in understanding the software requirements, especially in a foreign language and their negative influence on the software quality. A shared spoken or written language is not enough since language interpretations could cause misunderstandings. Therefore, it is essential to assess the language skills of team members in different roles, which would favour communication and positively impact software quality in the GSD domain.

We suggest considering language skills in assigning tasks to team members in software development projects. For example, analyst team members with lower language skills should not address higher-impact requirements. The author in Ref. [6] reaffirms our suggestion because he identified that representatives born/raised in two different cultures and languages have a deeper understanding than representatives who have lived and studied for several years in a different culture. Because these representatives can help translate both language and cultural mindsets, this could influence software quality improvement.

In addition, we encourage the results of this research to be taken into account by software laboratories conducting controlled experiments with subjects from different countries. We believe that the influence of language may influence the results of the experiments.

8 | CONCLUSIONS

In this experiment, we show that by using the native language to specify requirements, the external quality of the software product increases significantly, suggesting that this activity is

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

essential in the development process and should preferably be performed in the developers' native language to improve software quality. Native language has a positive effect on external quality, and foreign language has a negative effect on external quality.

This aspect is remarkable in the context of the globalisation of the software industry, as team members are increasingly composed of people from different countries and often speak different languages. However, GSD teams use a common language, usually English. Therefore, the language and language skills of the development team members in using a foreign language could influence the external quality of the software.

9 | FUTURE WORK

In future work, we suggest conducting experiments to measure the impact of other factors on software quality, such as developer culture and time zones. We also recommend experimenting with representatives as defined by the author [6] and with subjects with different foreign language skills through external replication [43]. Finally, we also propose to experiment with assigning roles to team members based on their language skills and observe the effect on software quality.

10 | VERIFIABILITY

For verifiability purposes, we publish as complementary material the replication package (RP) for "Effect of requirements specification using native language on external software quality" in Ref. [44], which is structured as follows:

- Descriptive RP (experiment description).
- README file (replication package description).
- Requirements document (software requirements for the experimental task development environment).
- Software Testing - Bowling and Software Testing - Tennis files (measurement instruments).
- Template-Bowling and Template-Tennis folders (source code template).

AUTHOR CONTRIBUTIONS

Fernando Uyaguari: Investigation; Project administration; Supervision; Writing – original draft. **Cathy Guevara Vega**: Conceptualisation; Funding acquisition; Investigation. **José Quiña Mera**: Investigation; Visualisation; Writing – review & editing. **Cristina Acosta**: Data curation; Resources; Writing – original draft.

ACKNOWLEDGEMENTS

Universidad Técnica del Norte.

CONFLICT OF INTEREST STATEMENT

The authors have declared no conflict of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in 'Zenodo' at https://doi.org/10.5281/zenodo.4950367.

ORCID

Cathy Guevara-Vega https://orcid.org/0000-0002-2470-8287

REFERENCES

1. Calefato, F., et al.: Assessing the impact of real-time machine translation on multilingual meetings in global software projects. Empir. Softw. Eng. 21(3), 1002–1034 (2012). https://doi.org/10.1007/s10664-015-9372-x
2. Dick, J., Hull, E., Jackson, K.: Requirements Engineering. Springer, Switzerland (2017). 239. https://doi.org/10.1007/978-3-319-61073-3. https://www.springer.com/gp/book/9783319610726
3. Calefato, F., et al.: An empirical simulation-based study of real-time speech translation for multilingual global project teams. In: International Symposium on Empirical Software Engineering and Measurement (2014). https://doi.org/10.1145/2652524.2652537
4. Damian, D.E., Zowghi, D.: Requirements Engineering challenges in multi-site software development organizations. Publ. App. Req. Eng. J. 8(3), 149–160 (2003). https://doi.org/10.1007/s00766-003-0173-1
5. Damian, D.: Stakeholders in global requirements engineering: lessons learned from practice. IEEE Softw. 24(2), 21–27 (2007). https://doi.org/10.1109/ms.2007.55
6. Brockmann, P.S., Thaumuller, T.: Cultural aspects of global requirements engineering: an empirical Chinese-German case study. In: 2009 Fourth IEEE International Conference on Global Software Engineering, pp. 353–357. IEEE (2009)
7. Schmid, K.: Challenges and solutions in global requirements engineering–a literature survey. In: International Conference on Software Quality, pp. 85–99. Springer (2014)
8. Derkacz, J., et al.: Definition of requirements for accessing multilingual information opinions. Multimed. Tool. Appl. 77(7), 8359–8374 (2018). https://doi.org/10.1007/s11042-017-4737-3
9. Mahmood, S., Ajila, S.A.: Software requirements elicitation–a controlled experiment to measure the impact of a native natural language. In: 2013 IEEE 37th Annual Computer Software and Applications Conference, pp. 437–442. IEEE (2013)
10. Binder, J.: Global Project Management: Communication, Collaboration and Management across Borders, p. 312. Routledge. London (2016). https://doi.org/10.4324/9781315584997
11. Sangwan, R., et al.: Global Software Development Handbook, p. 288. Auerbach Publications, New York (2006). https://doi.org/10.1201/9781420013856
12. Guevara-Vega, C., et al.: Empirical strategies in software engineering research: a literature survey. In: International Conference on Information Systems and Software Technologies (ICI2ST). IEEE Press, Quito, Ecuador (2021)
13. Jedlitschka, A., Ciolkowski, M., Pfahl, D.: Reporting experiments in software engineering. In: Guide to Advanced Empirical Software Engineering, pp. 201–228. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5_8
14. Nicolás, J., et al.: On the risks and safeguards for requirements engineering in global software development: systematic literature review and quantitative assessment. IEEE Access 6, 59628–59656 (2018). https://doi.org/10.1109/access.2018.2874096
15. Shafiq, M., et al.: Effect of project management in requirements engineering and requirements change management processes for global software development. IEEE Access 6, 25747–25763 (2018). https://doi.org/10.1109/access.2018.2834473
16. Khan, H.U., et al.: Empirical investigation of critical requirements engineering practices for global software development. IEEE Access 9, 93593–93613 (2021). https://doi.org/10.1109/access.2021.3092679

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

17. Chetan, A., et al.: Automated checking of conformance to requirements templates using natural language processing. IEEE Trans. Softw. Eng. 41(10), 944–968 (2015). https://doi.org/10.1109/TSE.2015.2428709
18. Cruz, B.D., et al.: Detecting vague words & phrases in requirements documents in a multilingual environment. In: Proceedings–2017 IEEE 25th International Requirements Engineering Conference, RE 2017, pp. 233–242 (2017). https://doi.org/10.1109/RE.2017.24
19. Tjong, S.F., Berry, D.M.: The design of SREE – a prototype potential ambiguity finder for requirements specifications and lessons learned. In: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 80–95 (2013). https://doi.org/10.1007/978-3-642-37422-7_6
20. Bhatia, M., Vasal, A.: Localisation and requirement engineering in context to indian scenario. In: 15th IEEE International Requirements Engineering Conference (RE 2007), pp. 393–394. IEEE (2007)
21. Paasivaara, M., Lassenius, C.: Could global software development benefit from agile methods? In: 2006 IEEE International Conference on Global Software Engineering (ICGSE'06), pp. 109–113. IEEE (2006)
22. Akbar, M.A., et al.: A systematic study to improve the requirements engineering process in the domain of global software development. IEEE Access 8, 53374–53393 (2020). https://doi.org/10.1109/access.2020.2979468
23. Nidhra, S., et al.: Knowledge transfer challenges and mitigation strategies in global software development—a systematic literature review and industrial validation. Int. J. Inf. Manag. 33(2), 333–355 (2013). https://doi.org/10.1016/j.ijinfomgt.2012.11.004
24. Fügen, C., et al.: Efficient handling of multilingual language models. In: IEEE 2003. Workshop on Automatic Speech Recognition and Understanding, pp. 441–446. IEEE, USA (2003). https://doi.org/10.1109/ASRU.2003.1318481
25. Calefato, F., Lanubile, F., Prikladnicki, R.: A controlled experiment on the effects of machine translation in multilingual requirements meetings. In: Proceedings - 2011 6th IEEE International Conference on Global Software Engineering, ICGSE 2011, pp. 94–102 (2011). https://doi.org/10.1109/ICGSE.2011.14
26. Condori-Fernández, N., et al.: Practical relevance of experiments in comprehensibility of requirements specifications. In: Proceedings - 1st International Workshop on Empirical Requirements Engineering, EmpiRE 2011, pp. 21–28 (2011). https://doi.org/10.1109/EmpiRE.2011.6046251
27. Noll, J., Beecham, S., Richardson, I.: Global software development and collaboration: barriers and solutions. ACM Inroads 1(3), 66–78 (2011). https://doi.org/10.1145/1835428.1835445
28. Betz, S., Oberweis, A., Stephan, R.: Knowledge transfer in it offshore outsourcing projects: an analysis of the current state and best practices. In: 2010 5th IEEE International Conference on Global Software Engineering, pp. 330–335. IEEE (2010)
29. Šmite, D., et al.: An empirically based terminology and taxonomy for global software engineering. Empr. Softw. Eng. 19(1), 105–153 (2014). https://doi.org/10.1007/s10664-012-9217-9
30. Basili, V.R.: Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Encyclopedia of Software Engineering (1994)
31. Object Management Group: Unified Modeling Language. SpringerReference 3, 226 (2011). https://doi.org/10.1007/springerreference_63554
32. Nikora, A., Hayes, J., Holbrook, E.: Experiments in automated identification of ambiguous natural-language requirements. In: o Appear, Proceedings 21st IEEE International Symposium on Software Reliability Engineering. IEEE Computer Society, San Jose (2010)
33. ISO/IEC: Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE (2005)
34. ISO: ISO/IEC 25010: Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models (2011)
35. Hass, A.M.J.: Guide to Advanced Software Testing. Artech House, Inc., USA (2008)
36. Santos, A., et al.: A family of experiments on test-driven development. Empir. Softw. Eng. 42(3), 26 (2021). https://doi.org/10.1007/s10664-020-09895-8
37. Santos, A., et al.: Increasing validity through replication: an illustrative TDD case. Softw. Qual. J. 28(2), 371–395 (2020). https://doi.org/10.1007/s11219-020-09512-3
38. Dieste, A.O., et al.: Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study. Empir. Softw. Eng. 22(5), 2457–2542 (2017). https://doi.org/10.1007/s10664-016-9471-3
39. Tosun, A., et al.: An industry experiment on the effects of test-driven development on external quality and productivity. Empir. Softw. Eng. 22(6), 2763–2805 (2017). https://doi.org/10.1007/s10664-016-9490-0
40. Fucci, D., et al.: An external replication on the effects of test-driven development using a multi-site blind analysis approach. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '16). Association for Computing Machinery, vol. 3, pp. 1–10 (2016). https://doi.org/10.1145/2961111.2962592
41. Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall PTR, USA (2003)
42. Kampenes, V.B., et al.: A systematic review of effect size in software engineering experiments. Inf. Softw. Technol. 49(11-12), 1073–1086 (2007). https://doi.org/10.1016/j.infsof.2007.02.015
43. Cruz, M., et al.: A model-based approach for specifying changes in replications of empirical studies in computer science. Computing 26(4), 1573–7616 (2021). https://doi.org/10.1007/s00607-022-01133-x
44. Uyaguari, F., et al.: Replication package—effect of requirements specification using native language on external software quality. Zenodo (2021). https://doi.org/10.5281/zenodo.4950367

How to cite this article: Uyaguari, F., et al.: Effect of requirements specification using native language on external software quality. IET Soft. 17(3), 287–300 (2023). https://doi.org/10.1049/sfw2.12124

APPENDIXES

A | Bowling scorekeeper (BSK) specifications (Spanish)

Requirements Kata Bowling Game *Descripción del problema* Desarrollar un programa que, dada una secuencia válida de Rolls (Tiros) para una línea de American Ten-Pin Bowling, produzca la puntuación total para el juego. El juego presenta ocho reglas para determinar el puntaje final: 1. Cada juego, o "línea" de bolos, incluye diez frames para el jugador. 2. En cada frame, el jugador tiene hasta dos intentos para derribar todos los bolos. La puntuación de cada intento va de cero a diez. 3. Si en dos intentos, no logra derribarlos, su puntaje para este frame es el número total de bolos derribados en sus dos intentos. 4. Si en su primer intento en el frame, derriba todos los bolos, esto se llama un "strike". Su frame ha terminado, y su puntaje para el frame es diez más el simple total de los pines derribados en sus siguientes dos tiradas. 5. Si en dos intentos los derrota a todos, esto se llama "spare" y su puntaje para el frame es diez más el número de pines (bolos) derribados en su próximo lanzamiento es decir en su próximo turno. 6. La puntuación del juego es el total de todas las puntuaciones de

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

frame. 7. Si obtiene un "spare" o un "strike" en el último frame (décimo), el jugador de bolos puede lanzar una o dos bolas de bonificación más, respectivamente. Estos lanzamientos de bonificación se toman como parte del mismo turno. Si los lanzamientos de bonificación derriban todos los pines, el proceso no se repite. Los lanzamientos de bonificación solo se utilizan para calcular la puntuación del frame final. 8. La máxima puntuación será trecientos puntos es decir conseguir doce strikes consecutivos.

B | Bowling scorekeeper (BSK) specifications (English)

Requirements Kata Bowling Game *Problem Description* Develop is a programme that, given a valid sequence of Rolls for one line of American Ten/Pin Bowling, produces the total score for the game. The game has eight rules for the determination of the score: 1. Each game, or "line" of bowling, includes 10 turns, or "frames" for the bowler. 2. In each frame, the bowler gets up to two tries to knock down all the pins. The score on each try ranges from zero to 10. 3. If in two tries, he fails to knock them all down, his score for that frame is the total number of pins knocked down in his two tries. 4. If, on his first try in the frame, he knocks down all the pins, this is called a "strike". His turn is over, and his score for the frame is 10 plus the simple total of the pins knocked down in his next two rolls. 5. If in two tries he knocks them all down, this is called a "spare" and his score for the frame is 10 plus the number of pins knocked down on his next throw (in his next turn). 6. The game score is the total of all frame scores. 7. If he gets a spare or strike in the last (tenth) frame, the bowler gets to throw one or two more bonus balls, respectively. These bonus throws are taken as part of the same turn. If the bonus throws to knock down all the pins, the process does not repeat: The bonus throws are only used to calculate the score of the final frame. 8. The maximum score will be three hundred points which are to get 12 consecutive strikes.

C | Tennis game specifications (Spanish)

Requirements Kata Tennis Game *Descripción del problema* Se trata de desarrollar un juego de tenis simplificado donde cada "SET" es un juego. Se tienen dos jugadores playerA y playerB. Los puntos de los jugadores playerA y playerB se reciben en una cadena de caracteres. Ejemplos: game('AAB'), game(''), game ('ABABABA'). El sistema de puntuación tiene 6 reglas que deben considerarse: 1) Las puntuaciones de los jugadores van desde cero a tres puntos son descritos como "cero", "quince", "treinta", y "cuarenta" respectivamente. El marcador se escribe como "cero, cero", "quince, cero", "cuarenta, quince". 2) Si por lo menos se han anotado tres puntos por cada jugador, y las puntuaciones son iguales, el marcador es "deuce". 3) Si por lo menos se han anotado tres puntos de cada lado y un jugador tiene un punto más que su oponente, el marcador del juego es "ventaja" para el jugador que va ganando. Ejemplo. "Ventaja playerA" o "ventaja playerB". 4) El juego se gana cuando un jugador obtiene al menos cuatro puntos en total y al menos dos puntos más que su oponente. Ejemplo: "playerA gana" o "playerB gana".

D | Tennis game specifications (English)

Requirements Kata Tennis Game *Problem Description* Develop, a simple tennis game where each game is a SET. There are two players: playerA and playerB. The points of the players, playerA and playerB, are received in a string of characters. Example: game("AAB"), game(""), game("ABABABA"). The scoring system has six rules that must be considered: 1. The scores of the players ranging from zero to three points are described as 'zero', 'fifteen,' 'thirty' and 'forty' respectively. The marker is written as 'zero, zero,' 'fifteen, zero,' or 'forty, fifteen.' 2. If at least three points have been scored for each player, and the scores are equal, the score is 'deuce'. 3. If at least three points have been scored on each side and a player has one point more than his opponent, the score of the game is an "advantage" for the player who is winning. Example. "playerA advantage." 4. The game is won when a player gets at least four points in total and at least two points more than his opponent. Example: 'playerA wins' or 'playerB wins.'

E | Bowling test cases

Table E1 shows the test cases executed to obtain the external quality measure of the Bowling experimental task.

F | Tennis game test cases

Table F1 shows the test cases executed to obtain the external quality measure of the Tennis experimental task.

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE E1 Bowling game test cases.

Begin of table

Id	Purpose	Input	Output
CP001	Create set with null values	null, null	Illegal argument exception
CP002	Create a game with correct values	1,1	True
CP003	Consult frames game	1	10
CP004	Generate attempt 1 with 0 pins	1,0	0
CP005	Generate attempt 1 with 9 pins	1,10	9
CP006	Generate attempt 1 with 11 pins	1,11	Incorrect score
CP007	Generate attempt 1 with null	1,null	Incorrect score
CP008	Generate attempt 2 with 0 pins	2,0	0
CP009	Generate attempt 2 with 9 pins	2,10	9
CP010	Generate attempt 2 with 11 pins	2,11	Incorrect score
CP011	Generate attempt 2 with null	2,null	Incorrect score
CP012	Generate attempt 3 with 1 pin	3,1	Incorrect attempt
CP013	Generate try 1 with 0 pins and try 2 with 9 pins	1,0–2,9	9
CP014	Generate try 1 with 8 pins and try 2 with 1 pins	1,9–2,0	9
CP015	Generate try 1 with 4 pins and try 2 with 5 pins	1,4–2,5	9
CP016	Generate try 1 with 5 pins and try 2 with 4 pins	1,5–2,4	9
CP017	Generate try 1 with 0 pins and try 2 with 1 pins	1,0–2,1	1
CP018	Generate try 1 with 1 pins and try 2 with 0 pins	1,1–2,0	1
CP019	Generate try 1 with 0 pins and try 2 with 0 pins	1,0–2,0	0
CP020	Generate attempt 1 with 10 pins	1,10	Strike
CP021	Check frame score with one attempt with 10 pins	1,10	10
CP022	Generate 2 attempts with 10 pins	1,6–2,4	Spare
CP023	Check frame score with two attempts with 10 pins	1,6–2,4	10
CP024	Check game score with zero points	0,0,0,0,0,0,0,0,0,0	0
CP025	Check game score without strike or spare	0,1,2,3,4,5,6,7,8,9	45
CP026	Check game score with intermediate strikes	8,8,8,8,10,10,7,8,8,8	115
CP027	Check game score with intermediate spares	8,8,8,8,10,10,7,8,8,8	100
CP028	Check score game with 1 strike and 1 intermediate spare	8,8,8,8,10,10,7,8,8,8	107
CP029	Consult score game with 1 spare and 1 strike in between	8,8,8,8,10,10,7,8,8,8	108
CP030	Check score game with perfect score	10,10,10,10,10,10,10, 10,10,10–9,9	298
CP031	Check game score with spare at the end	10,10,10,10,10,10,10, 10,10,10–10	290
CP032	Check score game with perfect score	10,10,10,10,10,10,10, 10,10,10–10.10	300

End of table

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE F1 Tennis game test cases.

Begin of table

Id	Purpose	Input	Output
CP001	Create game with PlayerA with null value	Null, PlayerB	Illegal argument exception
CP002	Create game with PlayerB with null value	PlayerA, null	Illegal argument exception
CP003	Create game with players with null value	Null, null	Illegal argument exception
CP004	Generate score with null value	Null	Illegal argument exception
CP005	Generate score with empty string	””	Zero, zero
CP006	Generate score with A string A	A	Fifteen, zero
CP007	Generate score with AA string AA	AA	Thirty, zero
CP008	Generate score with AAA string AAA	AAA	Forty, zero
CP009	Generate score with B string B	B	Zero, fifteen
CP010	Generate score with AB string	AB	Fifteen, fifteen
CP011	Generate score with AAB string	AAB	Thirty, fifteen
CP012	Generate score with AAAB string	AAAB	Forty, fifteen
CP013	Generate score with BB string	BB	Zero, thirty
CP014	Generate score with ABB string	ABB	Fifteen, thirty
CP015	Generate score with AABB string	AABB	Thirty, thirty
CP016	Generate score with AAABB string	AAABB	Forty, thirty
CP017	Generate score with BBB string	BBB	Zero, forty
CP018	Generate score with ABBB string	ABBB	Fifteen, forty
CP019	Generate score with AABBB string	AABBB	Thirty, forty
CP020	Generate score with ABAB string	ABAB	Thirty, thirty
CP021	Generate score with BABA string	BABA	Thirty, thirty
CP022	Generate score with AAABBB string	AAABBB	Deuce
CP023	Generate score with ABABAB string	ABABAB	Deuce
CP024	Generate score with BABABA string	BABABA	Deuce
CP025	Generate score with BBBAAA string	BBBAAA	Deuce
CP026	Generate score with ABABABAB string	ABABABAB	Deuce
CP027	Generate score with BABABABA string	BABABABA	Deuce
CP028	Generate score with AAAABBB string	BBBAAAA	AdvantageA
CP029	Generate score with AAABBBB string	AAABBBB	AdvantageB
CP030	Generate score with ABABABA string	ABABABA	AdvantageA
CP031	Generate score with BABABAB string	BABABAB	AdvantageB
CP032	Generate score with ABABABABA string	ABABABABA	AdvantageA
CP033	Generate score with BABABABAB string	BABABABAB	AdvantageB
CP034	Generate score with AAAA string	AAAA	WinA
CP035	Generate score with BAAAA string	BAAAA	WinA
CP036	Generate score with BBAAAA string	BBAAAA	WinA
CP037	Generate score with BBBAAAAA string	BBBAAAAA	WinA
CP038	Generate score with BBBB string	BBBB	WinB
CP039	Generate score with ABBBB string	ABBBB	WinB

(Continues)

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE F1 (Continued)

Begin of table			
Id	**Purpose**	**Input**	**Output**
CP040	Generate score with AABBBB string	AABBBB	WinB
CP041	Generate score with AAABBBBB string	AAABBBBB	WinB
CP042	Generate score with ABABAA string	ABABAA	WinA
CP043	Generate score with BABABB string	BABABB	WinB
CP044	Generate score with ABABABAA string	ABABABAA	WinA
CP045	Generate score with BABABABB string	BABABABB	WinB
CP046	Generate score with AAAAA string	AAAAA	InvalidA
CP047	Generate score with BBBBB string	BBBBB	InvalidB
CP048	Generate score with AAAAB string	AAAAB	InvalidB
CP049	Generate score with BBBBA string	BBBBA	InvalidA
End of table			

17518814, 2023, 3, Downloaded from https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/sfw2.12124 by Universidad UTE, Wiley Online Library on [09/02/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License