Requirements Engineering (2021) 26:481–508
https://doi.org/10.1007/s00766-021-00352-6

ORIGINAL ARTICLE

A framework for evaluating and improving requirements specifications based on the developers and testers perspective

Ana Carolina Oran[1] · Gleison Santos[2] · Bruno Gadelha[1] · Tayana Conte[1]

Received: 1 July 2020 / Accepted: 15 April 2021 / Published online: 25 June 2021

Abstract
Requirements specifications are essential to properly communicate requirements among the software development team members. However, each role in the team has different informational needs in order to perform their activities. Thus, the requirements engineer should provide the necessary information to meet each team member's necessities to reduce errors in software development due to inadequate or insufficient communication. Although some research is concerned with communicating requirements among clients and analysts, no related research has been found to evaluate and improve requirements communication within the software development team. With this in mind, we present the ReComP framework, which assists in the identification of problems in the artifacts used to communicate requirements, identification of informational requirements for each role of the development team, and provide improvement suggestions to address requirements communication problems. ReComP was developed using the Design Science Research (DSR) method and this paper presents the results of two DSR cycles considering the use of ReComP for the developer and tester roles by using, respectively, user stories and use cases as requirements specifications. The results provide evidence that ReComP helps software development teams to identify and improve issues in the requirements specifications used for project communication. In two independent studies, ReComP was able to decrease the frequency of problems by 77% in user stories identified by developers and the frequency of all (100%) problems in use cases identified by testers.

Keywords Requirements engineering · Requirements specification · Requirements based on perspectives · Requirements communication · Design Science Research

1 Introduction

Requirements communication plays an essential role in software development projects for coordinating clients, commercial project functions, and software engineers [1]. According to Fricker et al. [2], requirements communication is the process of transmitting a customer's needs to a development team to implement a solution. Successful requirements communication leads to a common understanding among stakeholders and the development team about what the relevant requirements and their meaning for the system to be developed are. Bjarnason et al. [3] state that requirements communication starts with contact with the customer and continues throughout the development of the project, involving different roles, for example, requirements engineers, developers, and testers. The authors also state that the requirements initially elicited need to be communicated and altered to the requirements negotiated and communicated among all affected roles. During software development, the team needs to communicate effectively and share requirements information in order to achieve understanding, consensus, and commitment to the project's objectives. Requirements communication problems can cause productivity losses or even design failures. For example, misunderstood or unreported requirements can lead to software that does

✉ Ana Carolina Oran
ana.oran@icomp.ufam.edu.br

Gleison Santos
gleison.santos@uniriotec.br

Bruno Gadelha
bruno@icomp.ufam.edu.br

Tayana Conte
tayana@icomp.ufam.edu.br

1 Federal University of Amazonas (UFAM), Manaus, AM, Brazil

2 Federal University of the State of Rio de Janeiro (UNIRIO), Rio de Janeiro, RJ, Brazil

not meet the customer's requirements and a subsequent low number of sales or additional costs to alter or recreate the implementation [1, 4]. Méndez Fernández et al. [5] highlight that the critical requirements engineering (RE) problems are related to communication problems and incomplete/hidden or unspecified requirements.

The failure or success of any software depends mainly on the software requirement specification, as it contains all the requirements and characteristics of the product [6]. There are different ways to represent the requirements of a system, from the use of free texts to more structured forms, such as use cases, user stories, and prototypes. The use case is a description that is widely used to specify the purpose of a software system and produce a report in terms of interactions between the actors and the system [7–10]. Differently from user stories, use cases are often used to describe the behavior of a system from a more technical point of view [11]. User stories are a structure most often used in agile software development [12, 13]. User stories can help the development team to understand the requirements from the user's perspective [14]. However, according to Lucassen et al. [15], user stories are often poorly written in practice, and this can create communication problems during development. Prototypes are drawings that show what the system's user interface (UI) should look like during the interaction between the system and the end-user [16]. Prototypes can be used along with use cases and user stories to improve understanding of functional requirements and to gain a better understanding of them.

It is important to represent the requirements in such a way that all involved stakeholders can establish a common understanding of the system's functionalities so that the final product developed meets the customers' expectations. According to Hoisl et al. [17], the way the analyst specifies the requirements information for stakeholders can influence the understanding of what should be developed. Different stakeholders have different roles and tasks within a project [18], as well as different information needs [19]. For this reason, the requirements specification documents (besides communicating the requirements to the customer) must provide each stakeholder with all the information necessary to perform their specific tasks properly. Stakeholders have difficulties related to the validation and understanding of the information found in the requirements specifications [5, 20]. Therefore, it is worth emphasizing the importance of understanding the requirements of each team member since each member has a role in the development of the system.

We conducted an in-depth study of related work [6, 7, 20, 34, 41] on software development teams' requirements communication problems. In the results, we identified the relevance of the problems raised and this motivated us to carry out our research. As such, the problem addressed in this study is related to the difficulty of communicating requirements between the development team members, considering the informational needs of each team member's role. Our objective is to support the improvement of requirements communication through requirements specification artifacts, considering the perspectives of different development team members. To achieve this goal, we created the ReComP framework (a framework for Requirements Communication based on Perspectives) to identify the difficulties of developers and testers in finding requirements information in specifications such as use case, user stories and prototypes, in order to perform their activities within the project. Furthermore, to meet team members' informational needs, ReComP is composed of a set of practical solutions to mitigate or eliminate problems, such as information that has been omitted, is poorly described, or lacking in detail or which has errors.

To conduct our research, we applied a Design Science Research (DSR) approach. DSR has been widely used in information systems to create and evaluate new artifacts [21–23]. In addition, we conducted exploratory studies in order to fully understand the problem and the evolution of the ReComP framework. In this paper, we report two design cycles carried out to evaluate and evolve ReComP for user story artifacts based on the developers' perspective, and use cases based on the testers' perspective.

The results of the two cycles show that ReComP helps software development teams in identifying and improving problems in the requirements specifications used for communication in software projects. In the first cycle, the use of ReComP managed to decrease the frequency of the problems identified by the developers in the user stories by 77%. In the second cycle, ReComP decreased the frequency of all (100%) problems identified by testers in use cases.

The remainder of this paper is organized as follows: Section 2 presents the concepts and related work. Section 3 details the DSR cycles for creating ReComP. Section 4 presents ReComP first version. Section 5 discusses the design of the empirical studies in which ReComP was assessed. Section 6 addresses the execution of the first DSR cycle and improvements that led to the ReComP second version. Section 7 presents the second DSR cycle and the ReComP third version, with new improvements. Section 8 presents the limitations and threats to validity. Finally, Sect. 9 presents conclusions and an outlook on future work.

2 Background and related work

This research involves concepts regarding requirements specifications that can used as a means for improving requirements communication in software development teams. The following sections present the background and related work.

2.1 Requirements specifications

The representation of software requirements is a topic that has been widely addressed in the literature. As such, a variety of methods, techniques and approaches have been applied in different domains [7]. Bjarnason et al. [24] state that requirements specifications are used for different purposes and support the main activities associated with obtaining and validating stakeholder requirements, software verification, tracking and management of requirements, and can also be used for contractual purposes through the documentation of customer agreements. The requirements specification contains the user and system requirements, that is, the specification of functional requirements and non-functional requirements [25].

According to Medeiros et al. [26], there are different ways to specify requirements. Three of the most used in the industry are use cases [8, 27], user stories [28] (due to the growth of agile development), and prototypes [12].

2.2 Use case

The use case description is a way to specify the functional requirements of a software system [29]. It is a technique used to specify the purpose of a software system and produce its description in terms of interactions among the actors and the system in question [7]. However, some common problems, such as ambiguities, incompleteness and inconsistencies, can arise when trying to describe the requirements through use cases. These problems can cause difficulties in understanding the requirements and, consequently, defects in the software system under development [7].

In their work, Tiwari and Gupta [29] identified the existence of different ways of representing use cases, which can be applied to various activities in the software development lifecycle. The authors observed that the use case models share some standard fields, such as the use case name, actors, preconditions, basic restrictions, alternative flows, postconditions. One of the structures widely used by requirements engineers is the structure proposed by Phalp et al. [30], which complements the fields suggested by Tiwari and Gupta [29] with items such as: description, containing a brief description of the use case and description of the purpose of the use case; exception flows, which describe ways to recover from errors that may occur in specific steps of the use case; and business rules, which are policies, procedures, or restrictions that must be considered during the execution of the use case.

2.3 User story

User stories are the artifacts most frequently used in agile software development [12, 28]. They consist of brief descriptions, from the perspective of the end-user, of the desired functionalities, and encompass several aspects of the requirements specification represented in natural language [31]. However, when improperly defined, they can trigger several challenges in agile software development, due to incomplete or incorrect documentation [11, 32].

Gilson and Irwin [13] state that many templates have been proposed over the past years, ranging from templates without restrictions and free format to very stringent ones. However, Cohn's [33] initial suggestion remains the most used model for stories. Cohn proposed the following structure: "As a < type of user > I want < some goal > so that < some reason > ".

According to Soares et al. [34], the lack of a detailed specification in user stories can lead to the development of features that are not properly aligned with the customer's expectations. This limitation of information imposed by the template makes it difficult to understand the requirements that are to be implemented. Thus, to make the specification of user stories more complete, the definition of acceptance criteria becomes necessary. The acceptance criteria describe the limits of a user story, and they are used as a parameter to measure whether a user story has been completed [35]. The acceptance criteria may contain information that was not originally in the user story template, such as business rules, exceptions, specific non-functional requirements, system's screen specification, and other information that the team deems necessary to develop the specified requirement.

2.4 Prototype

The prototype is an excellent way to generate ideas about a user interface (UI) and allows one to evaluate a solution at an early stage of the project [36]. De Lucia et al. [37] recommend the use of prototypes to document the requirements for communication and knowledge sharing between stakeholders and agile teams. For Blomkvist et al. [38], some of the benefits of using prototypes is that the prototypes are easier to interpret, give a clearer overview of the design and function as a stronger means of communication between stakeholders because of interactive qualities inherent in prototypes.

Several authors also use the terms "mockups" and "wireframes" to talk about prototypes [11]. Prototypes can be categorized into low, medium and high fidelity [39]. According to Walker [40], prototypes that are the most similar to the final product are "high fidelity" (e.g., prototypes made in HTML). In contrast, those less similar are "low fidelity" (e.g., paper prototype or sketches). According to Preece et al. [39], the overriding consideration is the prototype's purpose and what level of fidelity is needed to get useful feedback. Low-fidelity prototypes are useful because they tend to be simple, inexpensive, and quick to produce. They serve to identify issues in the early stages of design and, through role interpretation, users can get a real sense of what it will be like to interact with the product. High-fidelity prototyping is useful for selling ideas to people and for testing technical problems.

Prototypes can be used in conjunction with use cases and user stories to simultaneously improve the understanding of functional requirements [7], and allow the representation of non-functional requirements related to the user interface [41]. However, prototypes are not just support for understanding the requirements and are a fundamental part of requirements specifications when they present relevant information which is not documented in use cases or user stories [42].

This study focuses on using low-fidelity prototypes, since the study participants created drawings on paper to represent the system's user interface and the interaction between the user and the system. For Reggio et al. [7], mockups are drawings that show what the system's user interface should look like during the interaction between it and the end-user (user-system interaction). Therefore, throughout this article, we will use the UI Mockups nomenclature to reference the prototypes developed by the participants in the studies.

2.5 Related work

Hess et al. [43] present a comparison between traditional requirements artifacts and agile practices used to document requirements information. They conducted empirical studies to investigate the priorities of agile RE practices most frequently used in projects (user stories, sprint backlog, epics, product backlog, planning meetings, and face-to-face communication, personas, requirements prioritization, time-boxed iterations) from the point of view of members (developer/tester, Scrum master, product owner and requirements engineer). In addition, Hess et al. [43] also investigated the challenges faced by participants in their projects. Analysis of the study data revealed that the relevance of agile RE practices differed among different members of the agile team, and that the biggest challenges in agile projects are insufficient communication with customer(s) due to lack of documentation, and rework due to neglect of non-functional requirements, rework due to inadequate quality of documented requirements, Communication lapse due to unavailability of appropriate customer representative(s) (product owner), insufficient requirements communication in teams due to lack of documentation, and communication lapses due to sudden changes in the requirements.

Soares et al. [34] analyzed the use of agile requirements (user stories) with traditional approaches (use cases) for specifying requirements in software development projects. For this, they carried out a literature review, which identified the main difficulties when working with agile requirements, and an exploratory study that characterized the difficulties in using user stories compared to use cases. The results indicated that the main difficulties in using user stories to specify requirements are related to: sparse detailing of requirements information, difficulty in identifying non-functional requirements, non-definition of dependency between requirements, user dependence, lack of definition of business rules, volatility of requirements, communication and collaboration with users, lack of information for validation of requirements. In view of this, prototypes may be needed to assist in understanding this type of specification. In addition, the study participants' perceptions indicated that, although user stories can provide an initial time gain during the requirements specification activities, difficulties such as the lack of a detailed specification can lead to the development of features that are poorly aligned with customer expectations. Furthermore, using user stories to specify requirements can bring additional challenges to other development activities, such as maintenance and architecture design.

Tu et al. [20] state that the use of more transparent documents, with greater visibility of information for stakeholders, is an essential factor in communication effectiveness in software projects. The authors conducted a study with students and software professionals with different profiles using requirements documents of varying levels of transparency and employed two questionnaires. One had questions about the system described in the received document and required the subjects to identify problems if they could not answer the question about the requirement in the document, and the other asked them to give an opinion on the three attributes of transparency (accessibility, ease of understanding and relevance) in the document they received. The study results showed evidence that participants with the most transparent document spent less time seeking information, answered more questions correctly, and were more confident in their answers than participants with the least transparent document. As such, having a transparent requirements document is useful for communicating requirements to interested parties.

To improve the quality of requirements specification documents, Ali et al. [6] developed a methodology to identify and solve problems with the quality of the requirements specification using four processes to improve different quality attributes. In the first step, in the analysis phase, the input requirements are added, which ensures the integrity of the requirements, especially the domain requirements of the product. Then, the output requirements are inserted in the mapping phase, which acts as a stage for validating and verifying requirements from different stakeholders' perspectives. After removing the incorrect requirements using the mapping process, the requirements are added to the SRS (software requirement specification) and then supplied to the stakeholders for further inspection. After inspecting the

SRS using inspection templates and assigning the total quality score (TQS), the person responsible for the inspection sends a detailed report to the requirements engineering team.

Reggio et al. [7] proposed DUSM (disciplined use cases with screen mockups), a method for describing and refining requirements specifications based on use cases and screen mockups. The results show that, thanks to the screen prototypes, requirements specifications produced with DUSM are easier to understand, less prone to inconsistencies, ambiguities, and do not suffer from incompleteness, thanks to the glossary and many well-formed constraints. In general, DUSM produces "good quality" specifications and is inexpensive to apply.

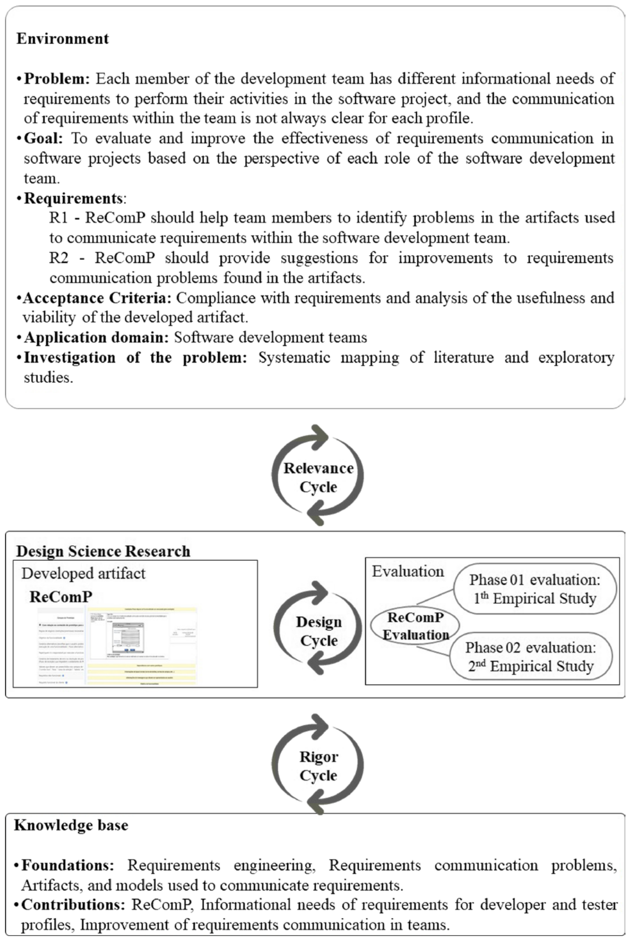
2.6 Discussion

In the related work, we identified that the main problems in requirements communication are related to incomplete, inaccurate, or incorrect requirements specifications [3, 5] and a lack of standardization of terminologies, models, and documents used to communicate the requirements [5, 20, 25]. We also identified that different roles have different informational needs in relation to the requirements in the software development project [3, 19, 43]. According to Bjarnason et al. [3], the communication gaps between the requirements team and the testers and developers result in problems in specifying unclear, ambiguous, and non-verifiable requirements and subsequent problems when implementing and verifying them. However, the solutions to the problems in the requirements specifications presented by these studies did not consider the difficulty that the team members have in identifying the requirements information that is necessary to carry out their project activities.

In view of this, we identified an opportunity to create the ReComP framework to evaluate and improve requirements communication. This is addressed to requirements specification artifacts and considers the informational requirements needs of the developers and testers in order to execute activities on the project. Since the primary artifacts used to specify requirements are use cases [7, 8, 13, 19, 27, 29], user stories [12, 13, 28] and prototypes [12, 19, 37, 38], we have limited the evaluation and improvements in these types of specifications.

3 Applying DSR to develop ReComP

This section presents ReComP and how it was developed by following the Design Science Research (DSR) method. DSR aims to assist in the creation and evaluation of new artifacts in a given context [21, 22]. DSR seeks to understand the problem and build and evaluate artifacts that allow us to transform situations by changing their conditions to better



• **Problem:** Each member of the development team has different informational needs of requirements to perform their activities in the software project, and the communication of requirements within the team is not always clear for each profile.
• **Goal:** To evaluate and improve the effectiveness of requirements communication in software projects based on the perspective of each role of the software development team.
• **Requirements**:
R1 - ReComP should help team members to identify problems in the artifacts used to communicate requirements within the software development team.
R2 - ReComP should provide suggestions for improvements to requirements communication problems found in the artifacts.
• **Acceptance Criteria:** Compliance with requirements and analysis of the usefulness and viability of the developed artifact.
• **Application domain:** Software development teams
• **Investigation of the problem:** Systematic mapping of literature and exploratory studies.

• **Foundations:** Requirements engineering, Requirements communication problems, Artifacts, and models used to communicate requirements.
• **Contributions:** ReComP, Informational needs of requirements for developer and tester profiles, Improvement of requirements communication in teams.

Fig. 1 Overview of the Design Science Research cycles in this study—based on Hevner and Chatterjee [22]

and more desirable states [44]. According to Hevner [45], DSR is an iterative process that proposes three interlinked research cycles: the relevance cycle, the design cycle, and the rigor cycle.

Figure 1 presents the Design Science Research applied in this research. The design cycle considers the relevance cycle and the rigor cycle. These three cycles must be present and clearly identifiable in any Design Science Research project [45]. The inputs for Design Science Research are requirements of the relevance cycle and the theories and methods of design and evaluation that are extracted from the rigor cycle. The relevance cycle links the application domain to the DSR effort, suggesting requirements and, specifically, requiring the artifact to be applied to the application domain to validate its practical use. The design cycle is repeated in two DSR activities: construction and evaluation. Finally, the rigor cycle bases the other cycles on the existing knowledge base and, due to research activities, determines which new knowledge should contribute to the growing knowledge base [45].

The purpose of the relevance cycle is to define the problem to be addressed, the requirements for the new artifact, and to define acceptance criteria for evaluating the research results [45]. To highlight the relevance of the present work, we conducted an in-depth study of papers related to the objective of identifying requirements communication problems within development teams and artifacts used for requirements communication. Furthermore, in this cycle, we conducted the following two exploratory studies to more fully understand the problem:

1. A comparative study between types of the specification: we compared the issuance and reception of requirements using a use case and user story, and evaluated and compared the degree of correction of the specifications and the UI mockups created. The study comprised three steps: In step 1, groups received the scenarios to create the specifications with user story and use case. In step 2, the main activity was constructing UI mockups simulating the real system based on the specifications created. It is noteworthy that each group received a different use case and user story specification than the one specified in step 1. In step 3, the UI mockups were inspected by the groups that created the specification using the inspection checklists. To assess the correctness of the study participants' specifications, the researchers carried out inspections on the specifications by using the inspection checklists to assist in the verification of defects in the use case specification and user stories. From a quantitative and qualitative analysis of the data [46], we observed that different specification formats could provide similar results when communicating requirements, but we should not ignore the human factor. We also noted that the impact and number of defects found in the requirements specification and the construction of UI mockups are not sufficient to determine which of the two specifications is better or worse for communicating requirements between software development teams. So, in this perspective, software development teams that are in doubt about which of the specification forms to adopt can choose to use both user story and use case.
2. Observation study with use cases: We improved the understanding of the needs of developers in the construction of UI mockups. We evaluated the difficulties faced by developers when building UI mockups that employ use cases. In this study, participants received a use case specification developed by professional requirements analysts for a real industry system to construct prototypes, and simulate the real system. Participants also inspected the specification to detect defects that hindered understanding the requirements for constructing the prototype. Finally, they highlighted what information specified in the use case was necessary for the construction of the prototype. From a quantitative and qualitative analysis of the data [47], we observed that the four reasons why developers do not follow the specifications of use cases. These are the existence of specification errors, ambiguous information, lack of detailed specification or incomplete information and due to suggestions for improvement. In addition, the types of defects that most impacted the creation of UI mockups were cited as being due to omission, ambiguity, and incorrect fact.

In these studies, it was possible to identify some requirements information needed in the use case specifications and user stories to help participants build UI models simulating real system construction. It was also possible to identify different formats adopted by the participants to specify use cases and user stories and to create prototypes.

Different studies were necessary for a better understanding of the requirement communication problems within software development teams and the diverse informational needs of the different roles of the members in the development teams. Therefore, the problem addressed by the present study involves the need to improve the communication of requirements within development teams based on the perspective of team members who use the requirements specification documents as input for the execution of their activities.

Thus, our main motivation for developing ReComP is related to the difficulty in communicating requirements between members of the development team. Although there are different ways of representing the requirements of a system, which range from the use of free texts to more structured forms, problems with requirements communication may arise due to the specification model chosen for the system development process [48]. Moreover, since each development team member has different informational needs to execute their activities in the software project problems may arise with the communication of requirements due to the specification model chosen for the system development process [46] and also due to the non-fulfillment of the different informational requirements of the team [19]. These problems occur because requirements engineers do not consider the information needs of each member of their development team when specifying the requirements.

After considering the identified problem, we decided to assist requirements engineers in identifying the flaws in existing requirements information in their artifacts, from the team members' point of view, and propose improvements according to the development team members' need for information. We thus defined two requirements for ReComP:

- R1—ReComP should help team members to identify problems in the artifacts used to communicate requirements within the software development team.

- R2—ReComP should provide suggestions for improvements to requirements communication problems found in the artifacts.

These requirements were established based on the aspects identified in the literature and exploratory studies. R1 requirement was defined based on Liskin [19] and Tu et al. [20]. They state that the requirements artifacts are used by different people, with different roles and different needs to carry out their activities throughout the project. The authors also claim that there is often no perfect type of artifact that suits the needs of all participants, which makes it necessary to adopt a variety of different artifacts. The R2 requirement was defined based on Méndez Fernández et al. [25], who state that the lack of an adequate template in the requirements specification can lead to failures in communication. We present other metrics for the evaluation of ReComP in Sect. 5.4.

The purpose of the design cycle is to develop a solution to the problem raised in the previous cycle and evaluate the solution against the requirements until reaching a project that is considered satisfactory [45]. We conducted two design cycles (empirical studies) to evaluate the use of ReComP. In the first design cycle, we generated and evaluated the first version of ReComP (v1) through a case study at the Federal University of Amazonas and at the Federal University of the State of Rio de Janeiro. Further information about this version is presented in Sect. 4.

In the second design cycle, we evolved the second version of ReComP (v2) and conducted a new empirical study at the Federal University of Amazonas. Further information about ReComP (v2) is presented in Sect. 6.3. This paper presents the results of both evaluations. Sections 7.3 and 7.4 presents the improvements made to the second version of ReComP that led to the creation of the third version.

Finally, the purpose of the rigor cycle is the use and generation of knowledge [45]. According to Thuan et al. [49], the rigor cycle bases the other cycles on the existing knowledge base and, due to research activities, determines that new knowledge should add to the knowledge base. In this study, the main fundamentals are the knowledge related to the communication problems within the development teams, the requirements communication artifacts, the specific information that each member within the development team needs to perform their activities, and the evaluation method (the case study).

The main contribution to the knowledge base is ReComP itself, as a new framework to aid in the identification of requirements communication problems in the artifacts used by software development teams. Also, studies conducted to evaluate ReComP can serve as examples for others in the application of ReComP. We have also contributed with knowledge regarding: (1) the main artifacts used to communicate requirements within software projects; (2) the requirements communication problems within software projects; (3) informational needs of requirements from the perspective of the role of developer and testers; (4) aspects to be considered when creating the requirements specification for software development teams.

4 ReComP—framework requirements communication based on perspectives (v1)

ReComP was built using the results found in the related work and the results of the exploratory studies. The ReComP framework[1] is a structure that supports the improvement of requirements communication through the supply of artifacts capable of assisting the requirements engineer in the identification of problems in the specification and suggested improvement of the identified problems. Table 1 presents the results found in the relevance cycle that supported the creation of the ReComP framework.

We divided ReComP into two specific artifacts: TAX (Team Artefact eXperience) and TAI (Team Artifact Improvement), as described below. In its current version, there are different adaptations of TAX and TAI to assess various artifacts and these are specific to each member that will make the assessment. In other words, both artifacts support two independent, but complementary perspectives for the roles of developer and tester and the specification artifacts, UI mockups, use cases, and user stories:

1. Team Artifact eXperience (TAX)—Support for evaluating the experience of team members regarding the artifacts used to communicate requirements during the software project.
2. Team Artifact Improvement (TAI)—Support for improving the artifacts, proposing improvement suggestions to solve or mitigate the problems in the artifacts used to communicate requirements, with the aim of meeting the informational needs of requirements. The suggested patterns consist of the adoption of templates or elements in the artifacts to present the necessary information for the different roles of the members of the development team.

In a similar manner to the framework definition used by Lucassen et al. [15] and Jiang and Eberlein [50], the ReComP framework contains specific artifacts addressed

[1] We defined ReComP framework according to the Cambridge dictionary, “a supporting structure around which something can be built” and “a system of rules, ideas, or beliefs that is used to plan or decide something”.

Table 1 Results that aided in ReComP's creation

Results found in the relevance cycle	Influence on the creation of ReComP
Source: related work	
Main artifacts used by the team to requirements communicate: use cases [7, 8, 13, 19, 27, 29], user stories [12, 13, 28] and prototypes [12, 19, 37, 38]	1. ReComP supports the following types of requirement specification: use cases, user stories and prototypes.
Specification problems with poorly described information, lack of detail and errors [3, 5]	2. ReComP is composed of a guided form to inspect specifications, emphasizing the difficulty in finding the information needed to perform the team members' activities. 3. We defined a set of fields (data) to be inspected in the specifications, based on the specification's original template.
Problems with the lack of standardization of terminologies, models and documents used to communicate requirements [5, 20, 25]	4. ReComP is composed of guidelines to standardize the requirements specification. 5. We define solution models for the problems inherent to the set of information inspected. These models are based on the original template of the specification.
Informational requirements needs for each member's role in the development team [3, 19, 43]	6. ReComP works from the perspective of the following team members: developers and testers.
Source: exploratory studies [46, 47]	
Difficulty in specifying a use case Difficulty specifying user stories Difficulty in building prototypes	7. We reviewed of the set of fields (data) to be inspected in the specifications. 8. We reviewed the solution models for the problems inherent to the set of fields (data) that were inspected.

to identify informational needs of developers and testers roles, as well as, to assess how requirements specification using user story, use case, and UI mockups meet these needs. Table 2 shows the name of the artifacts.

Because TAX evaluates the team members' experience with the quality of requirements documents, it can be used independently of TAI. That said, TAI does need the results obtained in assessing TAX. In this way, the organization can only use TAX to discover possible problems in communicating requirements within the team and should not adopt TAI for improvements in artifacts. The solutions proposed by TAI are optional, and the requirements engineer decides whether to use them or not.

In the context of this research, we defined the experience of the members of the development team in a similar manner to the definition of Hassenzahl [51] for user experience (User Experience—UX). Hassenzahl [51] states that a good UX is a consequence of fulfilling human needs through interaction with the product or service. Thus, we consider that the experience of the development team members regarding artifacts consists in identifying the informational needs regarding requirements by team members when using a specific artifact as a source of information that aids them in the development of their activities within the project.

The primary users of the framework are requirements engineers, developers, and testers. ReComP aids requirements engineers in evaluating the informational needs of members of the development team and in improving the requirements specifications used in the project. Initially, developers and testers answer a guided TAX form to identify the problems in the specifications used to perform their activities on the project. After that, the requirements engineer has the opportunity to minimize these problems by using the TAI to make improvements in the specification.

The steps for using ReComP are (1) identify the artifacts that will pass the evaluation and are used to communicate the requirements, (2) identify the team member roles that will evaluate the artifact, (3) apply the assessment through TAX, (4) verify the result of the evaluation with the problems identified in the artifacts and, optionally, if you want to solve the problems (5), apply the TAI improvement solutions. If you are going to perform a new evaluation of the improved artifact, you should go back to step (3).

Table 2 ReComP artifacts

Specification artifacts	Role	
	Developers	Testers
User stories	TAX_US_Dev TAI_US_Dev	TAX_US_Test TAI_US_Test
Use case	TAX_UC_Dev TAI_UC_Dev	TAX_UC_Test TAI_UC_Test
UI mockups	TAX_UI_Dev TAI_UI_Dev	TAX_UI_Test TAI_UI_Test

Table 3 Part of the TAX for user story specification and developer role (TAX_US_Dev)

Questions	Type
1—Questions regarding the user stories utilized for the development of the activity	
TX1. Do you have difficulty identifying which customer requirement the user story is describing?	Single Choice (SC)
TX1.1. If it is possible to identify the customer's requirement in the description of the user story, how was it specified?	Open
TX2. When there is a dependency between user stories, is it easy to identify it in the description?	Single Choice (SC)
TX2.1. If it is possible to identify the dependencies between user stories in the description of the user story, how were they specified?	Open
…	
2—Questions regarding the information needs of the user stories for the development of your activity	
TX14. Is the information presented in the user story sufficient for the development of your activities?	Single Choice (SC)
TX14.1. What information do you need to develop your activities that is not described in this type of specification?	Open
TX15. Does the user story contain irrelevant information?	Single Choice (SC)
TX15.1. What information in the user story do you consider irrelevant to the development of your activities?	Open

4.1 Initial proposal for TAX guided forms

TAX is composed of a guided form to be applied by the development team in order to identify problems in the artifacts used to communicate requirements by different roles of the members in the team, and identify the necessary information that each role of the member considers important to carry out their activities in the development of the project.

Table 3 summarizes the TAX questions for the user story specification, from the developer's perspective in a simplified way. We defined the questions for each specification type and member's role that use the artifact to develop activities in the project according to the original templates of the use case specifications, user stories, and prototypes. In addition, we added questions about the informational needs requirements identified in the exploratory studies. The questions were divided into two aspects: (i) evaluation of the difficulty in obtaining information from the artifact used and (ii) evaluation of the needs for information about the artifact for the development of their activity.

It is important to note that the guided forms have requirements information that originally may not have been in the specification template used. However, according to the literature [35] and the result of exploratory studies [46] and [47], this requirement information that is not originally in the template is complementary information necessary for the execution of the activities of developers and testers in software projects.

If the team members have any difficulty in identifying any information related to requirements in the artifact adopted by the team, the requirements engineer can make the improvements suggested by the TAI guidelines presented in the following section.

4.2 The initial proposal for TAI guidelines

The templates used in the requirements documentation may be insufficient to communicate some information to all members of the development team. Thus, there was a need to propose suggestions for improvements to problems identified in the artifacts. With the use of TAX, the objective is to meet the informational needs of requirements for members of the software project development team. We created the proposed improvement suggestions according to the information obtained in publications related to requirements specification problems and exploratory studies. We defined two pieces of information to facilitate the adoption of the standard in the artifact to be improved. These were (1) the description of the problem that can occur in the artifact, and (2) suggestions for improving the artifact. All suggestions for TAI improvements related to the problems identified in TAX. Example: The problem identified in question TX1 has the proposed TI1 improvement solution. Each suggestion from TAI has a usage example to better assist the requirements engineer in changing requirements specifications.

Firstly, the company must use TAX to identify problems in the specification of requirements used in the project and, if the company so desires, it can use TAI as an aid to solve the problems identified. The improvement suggestions proposed are adaptations of the artifacts that already exist in the company so that they meet the needs of the members of the development team.

Table 4 presents part of the TAI guideline for user story specification from the developer's perspective. The TAI and TAX used in the second empirical study can be found in the technical report available in [52].

Table 4 Part of the TAI for user story specification and developer role (TAI_ US_Dev)

Problem	Improvement suggestions for User Story
TI8. I cannot identify which are the *error handling flows* or how to resolve situations that prevent the flow of the user story	*Suggestion:* Create exception scenarios for those described in the user story to describe ways to solve any problems that may occur in the execution of the user story *Example:* … As a product reseller, I want to add products to my order so I can buy Amora products to resell in my store *Exception scenarios* ES1—If the quantity of products ordered exceeds the quantity of stock, the system displays the message MSG2
TI11. I cannot identify the business rules (restrictions/premises) necessary for the operation of the user story	*Suggestion:* Create a field to identify the business rules that must be developed in the user story *Example:* … As a reseller of Amora products I want to add products to my order So you can buy Amora products to resell in my store *Business rules:* BR1—The product order quantity cannot exceed 100 units

5 Evaluating ReComP

To guide the research, we defined the research question: "How to aid the requirements communication based on the perspectives of each member role in the development team's members?" To achieve this objective and answer the research question, we developed ReComP (a framework for Requirements Communication based on Perspectives). It was necessary to evaluate ReComP by applying it to the problem and context, checking whether it produced the desired effects and whether a new interaction and DSR cycle were necessary, while corroborating or questioning the validity of the theoretical assumptions [23]. Thus, to evaluate the use of ReComP in a practical context, we performed two cycles of DSR with an empirical study in each cycle. The following sections present the study planning, execution, and data analysis.

5.1 ReComP evaluation plan

In order to evaluate ReComP (v1) for the user story artifact from the perspective of software developers (ReComP_US_Dev), the first DSR cycle included an empirical study, which was planned based on the research question: "What are the difficulties encountered by developers when building UI mockups using user stories?". In order to evaluate the ReComP (v2) for the use case artifact from the perspective of software testers (ReComP_UC_Test), the second DSR cycle featured yet another empirical study, which was planned based on the research question: "What are the difficulties encountered by testers when building test cases using use cases and UI mockups?".

In both evaluations, initially, the participants create their artifacts from an actual requirements specification, make the evaluation of the specification using TAX, and improve the specification using TAI. After each evaluation, we evolved ReComP to solve the problems found in the study that was carried out. Figure 2 shows the stages of the execution of the two ReComP evaluations.

5.2 Participants

We carried out the first DSR cycle (1st empirical study) in two universities, with 50 undergraduate students taking the Agile Requirements and Systems Analysis and Design class. In total, 37 participants were characterized as novices, since they had only academic experience with the specification of user story requirements. The 13 participants who had already worked with user stories in the industry were characterized as experienced. The participants played the role of developers since they received a requirements specification in the format of user stories to build UI mockups, thus simulating the initial development of a system.

We carried out the second DSR cycle (2nd empirical study) in a university with 37 undergraduate students taking the Analysis and Systems Design class. In total, 32 participants were characterized as novices since they had only academic experience with the use case specification. Five participants had development experience in the industry and were characterized as experienced. Participants played the

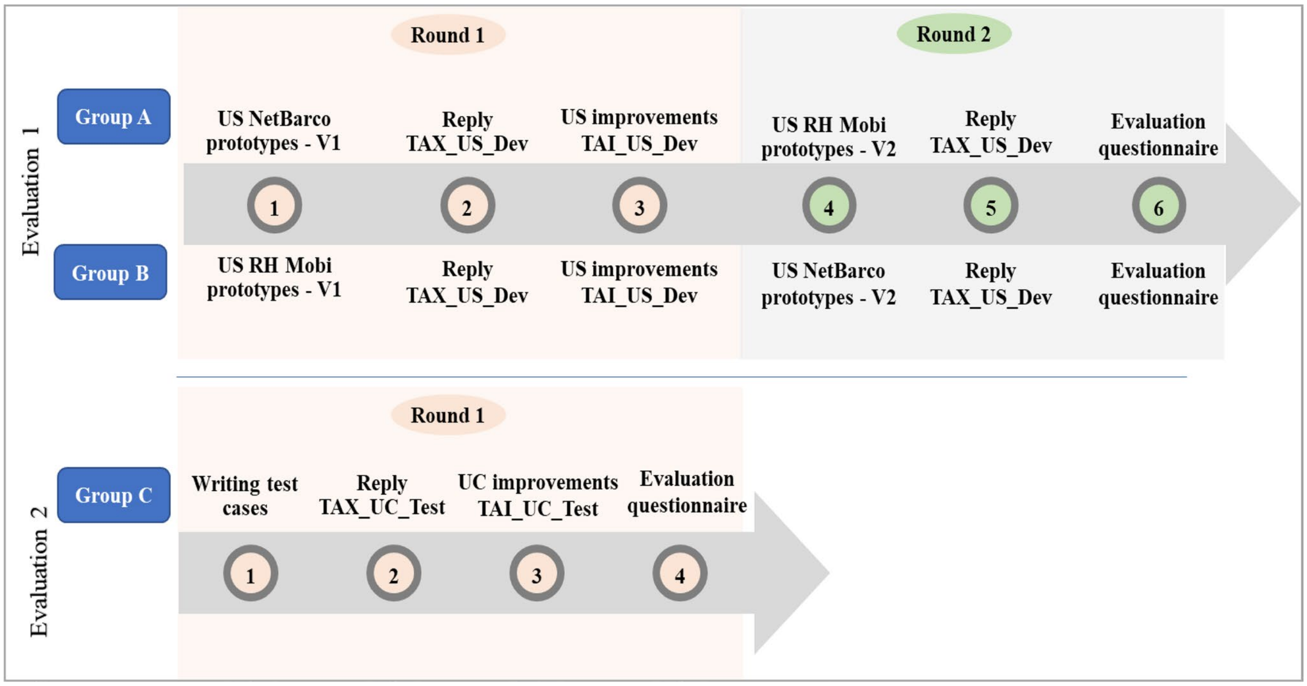


Fig. 2 Procedures followed in the studies

Table 5 ReComP participants and artifacts

Evaluation	Participants			ReComP		
	Quantity	Profile	Role	Profile	Origin Artifact	Target Artifact
1	50	Novice (37) Intermediate (13)	Developer	ReComP (v1)—US_Dev	User Story	UI mockups
2	37	Novice (32) Intermediate (5)	Tester	ReComP (v2)—UC_Test	Use Case, UI mockups	Test Case

role of testers because they received a specification of use cases and UI mockups to build test cases. Table 5 presents the synthesis of the participants and the use of ReComP.

5.3 Study artifacts

All the participants signed the informed consent form, in which they agreed to provide the results for analysis. In addition, the participants answered a characterization questionnaire so that we could verify their experience with software development in the industry and understand their degree of familiarity with requirements specification documents and test cases. At the end of each cycle, all participants answered an evaluation questionnaire about the ease of use and the usefulness of ReComP for improving requirements communication. The other artifacts used during the study were requirements specifications with a similar complexity level and the specific ReComP for each study. Table 6 shows the artifacts used in each cycle.

The final evaluation questionnaire has been defined based on the Technology Acceptance Model (TAM) for utility and ease of use of the indicators [53]. This consists of a 7-point Likert scale that assesses the participant's level of agreement for each statement regarding technology. The defined indicators were (1) perceived usefulness, which defines the degree in which a person believes that the technology could improve their performance at work, and (2) perceived ease of use, which defines the degree in which a person believes that using a specific technology would be effortless. The reason for focusing on these indicators is that these aspects are strongly correlated to the user's acceptance of the technology [53]. The response scale used was as follows: strongly agree, generally agree, partially agree, neutral, partially disagree, generally disagree, and strongly disagree. Table 7 presents the ReComP evaluation questionnaire applied in the cycles.

In addition to these artifacts, the first cycle also used: (a) textual description of two systems in the user story

Table 6 Artifacts used in the studies

Evaluation	Documents	ReComP	Examples of artifacts	
		Components	Version	External artifact
1	Consent form Characterization questionnaire	TAX, TAI	ReComP (v1)—US_Dev	User Story—RH Mobi User Story—NetBarco
2	Final evaluation questionnaire based on TAM	TAX, TAI	ReComP (v2)—UC_Test	Use Case—Document configuration UI mockups—Document configuration

Table 7 ReComP evaluation questionnaire

S1. The TAX questions are easy to understand
S2. The TAX questions are useful for detecting problems in user stories
S3. The TAI improvement suggestions are easy to understand
S4. The TAI improvement suggestions are useful for solving problems in user stories

template with similar levels of complexity, evaluated by the researchers; and (b) TAX_US_Dev and TAI_US_Dev—ReComP for user stories and the role of developers.

In the first cycle, we used two specifications of user stories adopted in real industry projects, one regarding a job offers system and the other about ferry ticket sales. A summary of the scenarios is presented herein:

1. “The App RH Mobi is a system that allows users to view job opportunities offered by an HR consulting company and job seekers to submit their resume for analysis. The objective of this web system is for the HR analysts of the consulting company to register job openings for companies and analyze the resumes sent by the application”.
2. “The NetBarco app is a system developed to run on an Android platform that allows sellers, authorized by boats, to sell tickets to passengers who want to travel from one municipality to another by ferryboat. The NetBarco app has two databases, the local one stored on the smartphone itself and the web one located on one of our servers hosted in the cloud”.

Further details about scenarios and requirements specifications are available in the technical report [52].

In the second cycle, the artifacts used were (a) Problem scenario, a textual description of a use case; (b) UI mockups of the use case; and (c) TAX_UC_Test and TAI_UC_Test—ReComP for specification with use case and tester’s role.

In this cycle, we used a use case specification adopted by a real industry project regarding a document management system to prevent documents from running out of their validity and causing business expenses such as fines and stoppages. The following scenario was presented:

The whole process of the company’s legal documentation control is carried out through the use of a spreadsheet where information is updated manually. This document management model can result in potential risks, such as the expiration of some important tax documents for company compliance with the tax authorities. Thus, the software will be responsible for serving the sectors that need document validity control. It will allow real-time monitoring of the company’s documents to avoid them from running out of their validity and cause expenses to the company (fines and stoppage of activities).

We highlight that the specifications used in the studies, created by software engineers of real projects, had additional information that was not part of the original template of use cases and user stories. Further details of the scenario and specification of requirements are available in the technical report [52].

5.4 ReComP evaluation indicators

To assess whether ReComP achieved its objective, we must consider the two requirements set out in the relevance cycle presented in Sect. 3. The ability of ReComP to aid in the identification and improvement of problems found in the requirements specifications was also evaluated. For this, we considered its viability and utility. In this perspective, ReComP must be considered viable if it can be executed according to its description if it produces what it promises to deliver and if its execution requires a level of effort that is deemed acceptable. On the other hand, the ReComP framework should be considered useful if it provides benefits to the team that is using it. Thus, we defined viability and utility indicators as follows (Table 8).

For analysis purposes, when assessing the specification carried out by the participants, the following interpretations were considered regarding the responses indicated in the guided TAX forms and shown in Table 9.

To analyze the participants’ perceptions regarding their use of ReComP, the following interpretations were considered regarding the responses indicated in the final evaluation

Table 8 Viability and utility indicators

Measure	Description	Evaluation
Viability = applicability and effectiveness and ease of use		
Ease of application	Ability to run the ReComP, as described in Sect. 4	Did ReComP run adequately without the need to create new steps or change the order of execution of the steps previously described?
Effectiveness	ReComP will be considered effective if the number of problems identified decreases after the suggested improvement	Number of problems identified in the evaluation N + 1 < Number of problems identified in the evaluation N
Ease of use	ReComP must be considered easy to use by participants	Evaluation questionnaire (based on TAM) after using ReComP (Questions 1 and 3 in Table 6)
Usefulness = the use of ReComP provides benefits for the organization		
Benefits	ReComP must be considered useful by participants	Evaluation questionnaire (based on TAM) after using ReComP (Question 2 and 4 in Table 6)

Table 9 Analysis of the specification assessment

Responses marked by participants	Interpretation
"Yes" and "In some cases"	The participant faced problems identifying the information
"No"	The participant had no problems identifying information

questionnaire based on TAM. These are presented in Table 10.

6 First DSR cycle (1st empirical study)

In the first cycle, we carried out the following 6 steps (Fig. 2): (1) building UI mockups using the user story, (2) application of TAX_US_Dev, (3) improvement of user story with TAI_US_Dev, (4) improvement of UI mockups with user story, (5) reapplication of TAX_US_Dev, and if its execution application of TAX_US_Dev and (6) final evaluation of ReComP. From this point on, we will adopt the following nomenclatures: Round 1 for the first evaluation using the original specification without changes and without worrying about the team role; and Round 2, for the second evaluation using the specification with improvement concerned with the informational need of the team role.

Thus, the execution of the study was divided into two rounds:

- Round 1: using the original specification without changes (traditional)—without worrying about the team role. In Step 1, we divided the participants randomly into two groups, one group received the specification from RH Mobi, and the other group received the specification from NetBarco. In Step 2, the participants used the ReComP artifact TAX_US_Dev to evaluate the specification received to construct the UI mockup. In Step 3, the participants improved the specification received based on the problems they pointed out in the assessment. For this, they used the improvement suggestions provided by TAI_US_Dev.
- Round 2: using the specification with the improvement suggested by TAI. In this scenario, the participants used the specified document regarding the developer role. In Step 4, participants received improved specifications to create the UI mockups. In this step, the participants did not make the UI mockup of the specifications they had received in Step 1. For this, we ensured the rotation of specifications among the participants of Round 1 and Round 2. For example, those who used the RH Mobi user story in Step 1, used the NetBarco user story (improved) in Step 4. Thus, we endeavored to reduce the learning bias in the type of specification.

In Step 5, participants received the TAX_US_Dev guided form and re-evaluated the specification received. It is worth

Table 10 Analysis of the post-ReComP questionnaire

Responses marked by participants Interpretation	Interpretation
"Strongly agree" and "Generally agree"	The participant agrees with the statement
"Partially agree," "Neutral" and "Partially disagree"	The participant has no clear opinion or was neutral
"Generally disagree" and "Strongly disagree"	The participant does not agree with the statement

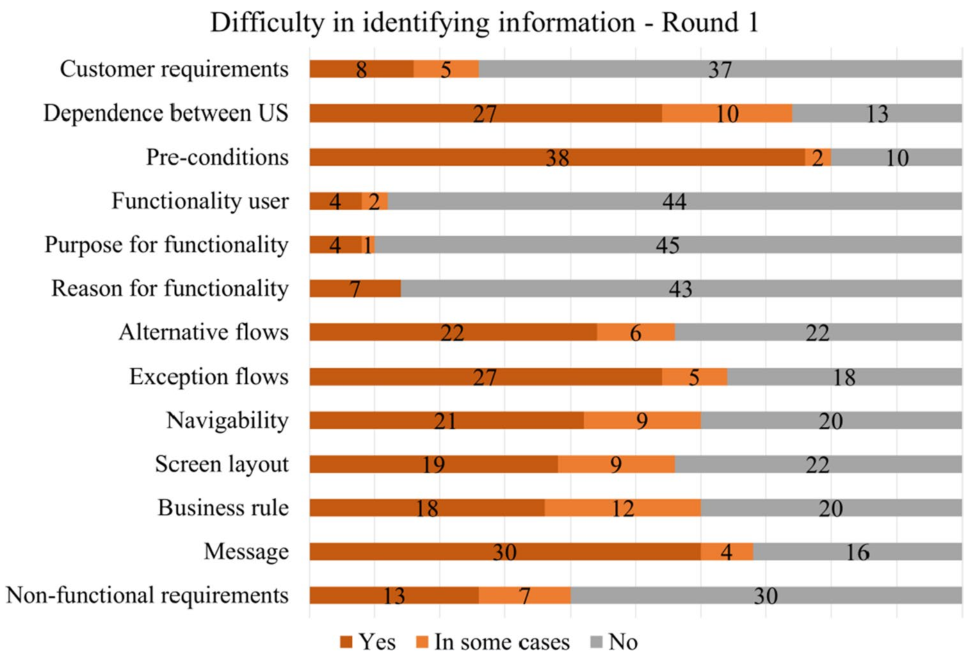


Fig. 3 Difficulty in identifying information in the user story—Round 1

mentioning that at this stage, the option "I do not need this information" was added to the guided TAX form so that participants had the option to indicate the information that they considered irrelevant to the development of their activities. Finally, in Step 6, a questionnaire was applied to assess the communication of requirements to the participants.

6.1 ReComP_US_Dev study results

In this section, we present the quantitative and qualitative results regarding the analysis of the difficulties encountered by developers building UI mockups using user stories. We also analyzed the informational requirements needs to build the UI mockup by the developer. In addition, we analyzed the developers' perceptions regarding the ReComP ease of use and usefulness.

6.1.1 Requirements specification evaluation

The guided TAX_US_Dev form applied in Round 1 allowed us to analyze the participants' perceptions concerning the fields of the user story specification. In this round, the user stories of the RH Mobi or NetBarco systems were created without considering the needs for a developer's role. Figure 3 presents the result of the problems encountered by the participants when building UI mockups using a user story in Round 1.

The information that most participants pointed out as having difficulty in identifying was related to the following: precondition (80%), the dependence between user stories (74%), messages (68%), and exception path (64%). On the other hand, the information that the participants had less difficulty in finding in the user story was related to customer requirements (26%), the reason for the functionality (14%), the user of the functionality (12%), and purpose of the functionality (10%).

6.1.2 Requirements specification improvements

As for the improvements made in the user story using TAI_US_Dev, we analyzed the quality of the improvement made by the participants. We checked the information registered in the TAX_US_Dev guided form and checked if the participants improved the US.

Analyzing the user stories, we classified them as: "Great" if the participants made all the improvements pointed out in the TAX (100% improvement); "Good" if the participants made most of the improvements pointed out in the TAX (51%-75% improvement); "Bad" if the participants made half or less of the improvements pointed out in the TAX (<=50% improvement); "Not recommended" if the participants made incorrect improvements (changes that modified the specified requirement instead) or made no improvements noted and, therefore, it was not possible to build UI mockups from the modified US (0% improvement). Figure 4 presents the evaluation of user stories.

Only 11 (22%) user stories were evaluated as "Great", 27 (54%) user stories evaluated "Good", 7 (14%) were evaluated as "Bad" and 5 (10%) user stories were evaluated as "Not recommended".

The guided TAX_US_Dev form applied in Round 2 allowed us to analyze the participants' perceptions regarding the fields of user story specification improved by other participants in Round 1 (RH Mobi or NetBarco—based on

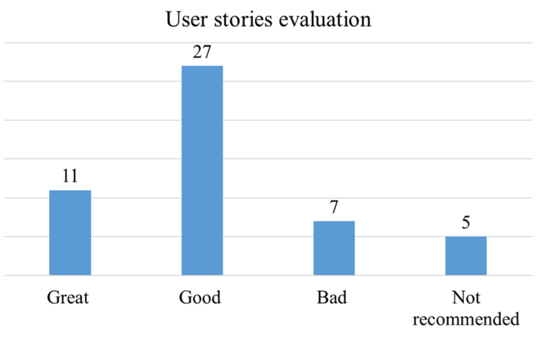
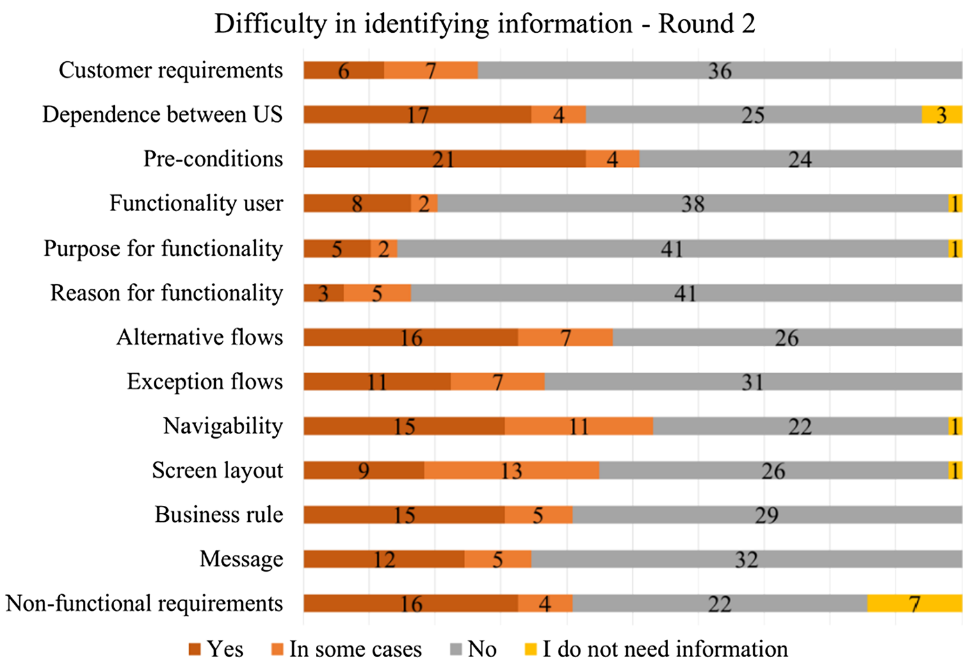


Fig. 4 User stories evaluation

the developer's perspective) that they had difficulty in identifying for the UI mockup development. Figure 5 presents the result of the problems found by the participants when building UI mockups using user stories in Round 2.

During the execution of Round 2, the number of participants was reduced to 49 because one participant did not show up at the study site. We observed that the information that most of the participants indicated they had difficulty in identifying were precondition, the dependence between user stories, and alternative paths. The information that the participants had less difficulty finding in the user story were the following: the reason for functionality, purpose of

Fig. 5 Difficulty in identifying information in the user story—Round 2



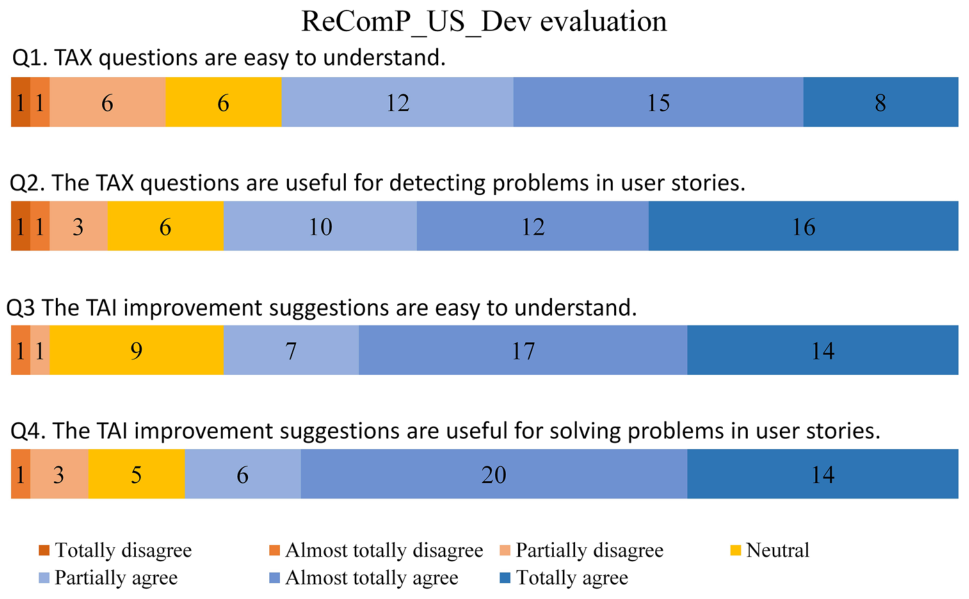


Fig. 6 ReComP_US_Dev evaluation

functionality, and customer requirements. Some participants indicated that they did not need to carry out their activities, such as dependence between the US (3), the user of the functionality (1), the purpose of the functionality (1), navigability (1), screen layout (1), and non-functional requirements (7).

6.1.3 Final analysis of the participants' perception when using ReComP_US_Dev

In this section, we present the results of the analysis of the participants' responses (P) to the questionnaire applied after the study was carried out. Our objective was to investigate the acceptability of ReComP. In this questionnaire, the participants answered regarding their level of agreement in relation to ReComP's usefulness and ease of use. Figure 6 shows the participants' perceptions of ReComP_US_Dev.

The results shown in Fig. 6 reveal mainly positive perceptions and few negative perceptions. There were few participants who disagreed about the ease of use and usefulness of ReComP for improving the communication of requirements in the team. Further on in the paper, the participants' perception of TAX and TAI will be detailed.

6.1.3.1 TAX_US_Dev Regarding the TAX_US_Dev ease of use, we analyzed the answers regarding the first statement (*S1. The TAX questions are easy to understand*), we can observe in Fig. 6 that 47% of the participants considered the questions in the guided TAX_US_Dev form easy and clear to understand. Also, the participants reported that the guided form helped them to identify problems in user stories.

> They have a vocabulary that is easy to understand because they clearly specify the objective of the question and use the same terms used in the user story.—P8
> "Questions can clearly address what they want to be specified"—P7

Only 4% of the participants disagreed that the questions found in the guided TAX_US_Dev form were easy to understand.

> "The questions presented at TAX were not entirely clear, but they are important questions"—P13
> "The questions could be asked more informally; they had a negative impact when reading for the first time"—P5

Figure 6 shows that 49% of the participants had no clear opinion or were neutral about the TAX_US_Dev's ease of use.

Regarding the TAX_US_Dev's usefulness, we analyzed the answers to the statement (*S2. The TAX questions are useful for detecting problems in user stories*), and it can be observed in Fig. 6 that 57% of the participants agreed that the questions in the guided TAX_US_Dev form were useful for detecting problems in user stories.

> "(…) I can identify problems more efficiently and safely"-P1
> "TAX focuses on the main points that the US should contain, so if those points are not present or if they are confused, it means that there is a problem in the description, and TAX helps to identify precisely those problems".—P17B.

Only 4% of the participants disagreed that the questions found in the guided TAX_US_Dev form were useful.

> "For the questions to be useful, it would be necessary for those who were applying the technique to have good knowledge of user story and prototyping."—P6

The number of participants who had no clear opinion or were neutral about the usefulness of TAX_US_Dev is 39%.

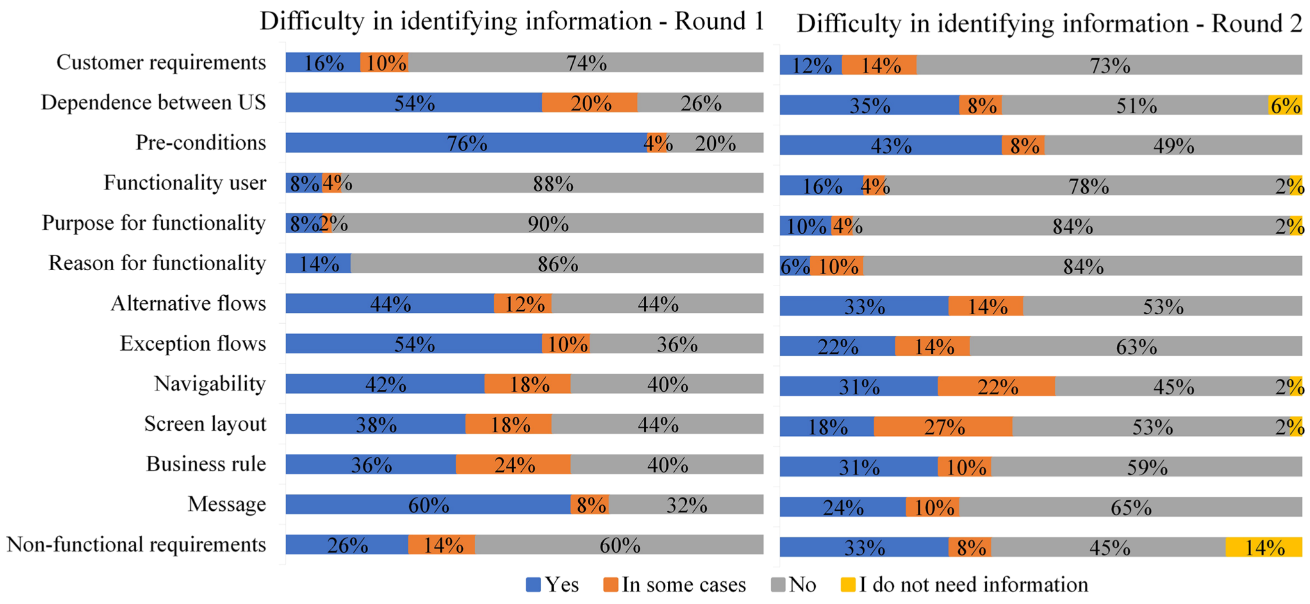
6.1.3.2 TAI_US_Dev Regarding the ease of use of TAI_US_Dev, we analyzed the answers regarding the statement (*S3. The TAI improvement suggestions are easy to understand*), in Fig. 6, we can observe that 63% of the participants agreed that the suggestions for improving TAI_US_Dev were easy to understand. A total of 35% of the participants had no clear opinion or were neutral about the ease of understanding TAI_US_Dev. Only 2% of the participants disagreed with the statement. As for the positive points of TAI ease of use the, we can highlight the following quotes from the participants:

> "The suggestions for improvement are simple to understand, and it is easy to implement them with the aid of examples"- P7
> "It is possible to understand all the issues necessary for a better understanding of the requirements, and the examples presented further help in understanding."—P2

Regarding the negative points of the ease of use of TAI, we can highlight the following quotes from the participants:

> "It points out what should be done to correct errors, but at times I found the suggestions too generic"—P6
> "Some explanations of errors are very formal, thus making the suggestions less useful."—P20B

As for the usefulness of TAI_US_Dev, we analyzed the answers to statement (*S4. The TAI improvement suggestions are useful for solving problems in user stories*), we observed that 69% of the participants considered the suggestions provided by TAI_US_Dev useful for improving user stories. A total of 29% of the participants had



Round 1	Yes	In some cases	No
Customer requirements	16%	10%	74%
Dependence between US	54%	20%	26%
Pre-conditions	76%	4%	20%
Functionality user	8%	4%	88%
Purpose for functionality	8%	2%	90%
Reason for functionality	14%		86%
Alternative flows	44%	12%	44%
Exception flows	54%	10%	36%
Navigability	42%	18%	40%
Screen layout	38%	18%	44%
Business rule	36%	24%	40%
Message	60%	8%	32%
Non-functional requirements	26%	14%	60%

Round 2	Yes	In some cases	No	I do not need information
Customer requirements	12%	14%	73%	
Dependence between US	35%	8%	51%	6%
Pre-conditions	43%	8%	49%	
Functionality user	16%	4%	78%	2%
Purpose for functionality	10%	4%	84%	2%
Reason for functionality	6%	10%	84%	
Alternative flows	33%	14%	53%	
Exception flows	22%	14%	63%	
Navigability	31%	22%	45%	2%
Screen layout	18%	27%	53%	2%
Business rule	31%	10%	59%	
Message	24%	10%	65%	
Non-functional requirements	33%	8%	45%	14%

Fig. 7 Difficulty in identifying information in the user story in Round 1 and Round 2

no clear opinion or were neutral about the usefulness of TAI_US_Dev, and only 2% of the participants disagree that the suggestions for improvement are useful. Regarding the positive points of TAI's usefulness, we can highlight the following quotes from the participants:

> "It is very useful to find missing data at each stage of creating the story."—P2
> "It helped to rewrite the critical points and those that were in doubt on how to redo."—P1B

Regarding the negative points of the usefulness of TAI, we can highlight the following quotes from the participants:

> "I was not able to apply the suggestions very well because I had a hard time transferring the examples to my case."—P6B
> "At the time of applying the TAI, the technique seemed too vague. When correcting the user story, I used my knowledge about user story more than the technique."—P6

It is worth mentioning that we considered all the opinions of the participants regarding the ease and usefulness of TAX as a point of improvement in the ReComP version.

6.1.3.3 General perception of ReComP As for the general perception of ReComP in the evaluation and improvement of user stories, the participants acting in the role of developers highlighted that they would use it again, since ReComP was useful and straightforward, helped in the writing of user stories, served as an inspection guide, and presented itself as a manual for construction and error prevention. Below are some quotes from participants regarding the ease and usefulness of ReComP.

> "It is useful because you do not need to review the entire description because the problems have already been identified by TAX, and TAI already shows the suggestion for improvement in the specific problem."—P7B
> "They are easy because if we identify a problem with TAX, the TAI solution already indicates exactly where and why the problem occurred."—P17B
> "All the problems that I identified in TAX and that I looked for in TAI were easy to implement."—P22B

We observed that the participants approved of the way of applying TAX to assess the quality of the requirements specification and TAI to solve the problems with the improvements suggestions presented. Thus, ReComP proved to be effective in its purpose.

6.1.3.4 Improvement suggestions Regarding suggestions for improvements in ReComP_US_Dev, the participants mainly pointed out improvements in the TAX guided form. The first suggestion for improvement refers to the need for improvement in the questions in the guided TAX form. In relation to this, participant P2 said that "… *In TAX, it would be good to standardize the questions,*" and P9 complemented the suggestion by saying that "*A better description of the TAX questions is necessary*."

It was also suggested that some fields questioned in TAX should be part of TAI. Regarding this suggestion, P9 commented that: "… *the main path and acceptance criteria should be part of TAI*". In addition, according to P13, TAX is "incomplete" and they mentioned that "*TAX lacks several important issues for the user story. In addition, there are poorly worded questions*". They also added that in *TAI "the questions presented are easier to understand.*"

6.2 Discussion

According to the analysis carried out on the results obtained in Round 1 (first assessment using the actual specification without changes and without considering the role) and in Round 2 (second assessment using the specification with improvement considering the role), we can see in Fig. 7 that, out of 13 problems found in the specification of the user story in Round 1, 10 (77%) problems decreased their frequency in Round 2 after the use of ReComP. Only 3 (23%) problems in the specification increased their frequency in relation to Round 1 (the user of the functionality, the purpose of the functionality, and non-functional requirements).

We observed that, in Round 2, 10 (20%) participants said they had problems identifying the information "functionality user", 7 (14%) participants had problems identifying the information "purpose for functionality," and 20 (41%) participants had trouble identifying the "non-functional requirements" information. These participants confronted the problem of receiving user stories with improvement problems in Round 2. In other words, the participants who should have improved the US in Round 1 with the use of TAI_US_Dev did not do it correctly, creating difficulties for the participants who used the US in Round 2.

Due to the addition of the field "I don't need this information" in the guided TAX form applied in Round 2, we can see that only 6 fields of information were considered unnecessary for the execution of activities. Only 1 (2%) participant pointed out that they do not need the functionality user information, the purpose of the functionality, navigability, screen layout. Another 3 (6%) participants pointed out the dependency information among user stories as unnecessary and, finally, 7 (14%) participants said that information on non-functional requirements is not necessary for the performance of their activities.

Regarding the improvements made in user stories through the application of TAI_US_Dev, 38 (76%) participants improved the user stories according to the proposed suggestions and only 12 (24%) participants had their improvements in the user story considered "bad" and "not recommended." The number of negative improvements may have occurred due to the difficulty in editing the user story that was carried out manually by the participants. However, it is worth mentioning that the improvements made by the participants minimized the occurrence of problems found in the specifications of the second round, showing that ReComP was effective in identifying and improving the problems in user stories.

From the analysis of the user's perception, we can see that, in general, most participants agreed with the statements about ease of use and usefulness in identifying problems in the user stories (TAX_US_Dev) and suggestions for improving the user stories (TAI_US_Dev). These results show evidence of ease of use when applying ReComP (v1). The fact that ReComP was widely accepted by the participants may indicate that this technique is also suitable for development teams that want to evaluate and improve their user stories in order to communicate requirements.

Regarding the negative points of the ReComP's utility mentioned by the participants, we highlight the need to have a minimum amount of knowledge of the specification template used for best use of the framework. Moreover, the application of the improvement suggestions presented in TAI is optional. However, the problems pointed out by TAX must be considered to meet the needs of the specification user.

The participants pointed out some difficulties in using ReComP in this study. Based on these difficulties, some improvements were made to ReComP, with the aim of improving its usefulness, ease of use, and effectiveness.

6.3 ReComP improvements (v2)

The main problems pointed out by the participants related to the guided TAX_US_Dev form were that the questions were very general, non-standardized and very extensive, which made their use difficult. Through this, we realized that R1—ReComP should allow team members to identify problems in the artifacts used to communicate requirements within the software development team—and this was not fully achieved in the study.

Therefore, we revised the guided form and made it more compact, removed the open questions from all fields, standardized it, and gave the questions a more direct approach. In addition, we added the option for the participants to indicate that they "do not need the information" to gain insights from what was described in a specification that is not necessary for a given role, thereby helping to reduce irrelevant information in the specification.

Regarding R2, ReComP should provide suggestions for solutions to improve the requirements communication problems found in the artifacts. As such, we noticed that some information was missing from the improvement standards and the use, as well as TAX_US_Dev being confusing. Therefore, we reviewed all the ReComP TAI guidelines and added all the suggestions to improve the fields corresponding to the problems found in TAX and added the instructions

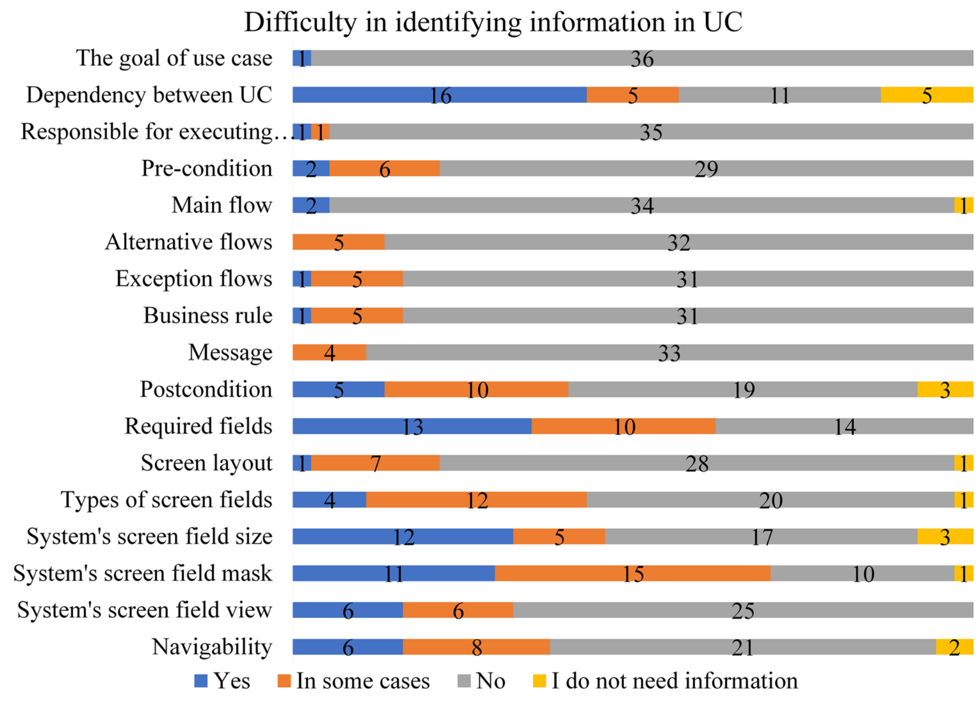


Fig. 8 Difficulty in identifying information in UC

on how to use the ReComP to help in its use. ReComP (v2), which was used in the second empirical study, can be found in the technical report available in [52].

7 Second DSR cycle (2nd empirical study)

In the second DSR cycle, we conducted only one round of evaluation and improvement of ReComP (v2) that led to ReComP third version. As shown in Fig. 2, we performed this study in the following 4 steps: (1) Specification of the test cases, (2) Application of TAX_UC_Test, (3) Improvement of the use case with TAI_UC_Test, (4) ReComP final evaluation.

In Step 1, the participants received the use case specification—UC01: Configure Document and the use case UI mockup. In Step 2, they used ReComP artifact TAX_UC_Test to evaluate the specification received in order to construct the test cases. In Step 3, the participants improved the specification received based on the problems identified in the assessment. For this, they used the TAI_UC_Test improvement suggestions. In Step 4, we applied an evaluation questionnaire to assess the participants' opinions regarding the communication of the requirements.

7.1 ReComP_UC_Test study results

In this section, we present the results regarding the analysis of the difficulties encountered by testers when building test cases using use cases and UI mockups. We also analyzed the testers' informational requirements needs to build test cases. In addition, we analyzed the perception of ReComP's ease of use and usefulness.

7.1.1 Requirements specification evaluation

The result of the guided TAX_UC_Test form allowed us to perform an analysis of the participants' perceptions concerning the fields of the use case specification (not based on the tester's perspective) in which they had difficulty in identifying in order to create the test cases. Figure 8 shows the result of the problems encountered by the participants when creating test cases using use cases.

Most participants pointed out that they had difficulty in identifying some information, such as screen field mask (70%), mandatory fields (62%), dependencies among use cases (57%), and size of screen fields (46%). The information that the participants had less difficulty finding in the use case were in regard to exception flows (16%), business rules (16%), main scenario (5%), and purpose of the use case (3%). In some cases, problems were attributed to alternative flows (14%) and messages (11%). Some participants pointed out that they do not need some information to perform their activities, such as main scenario (3%), screen layout (3%), types of screen fields (3%), screen fields mask (3%), navigability (5%), post-condition (8%), size of screen fields (8%), and dependencies among use cases (14%).

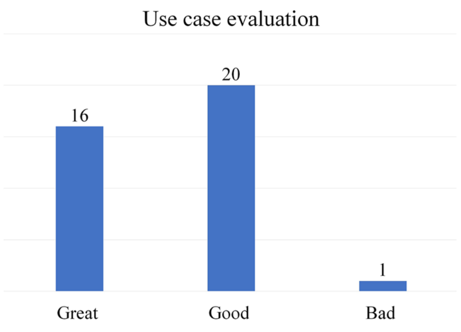


Fig. 9 Use case evaluation

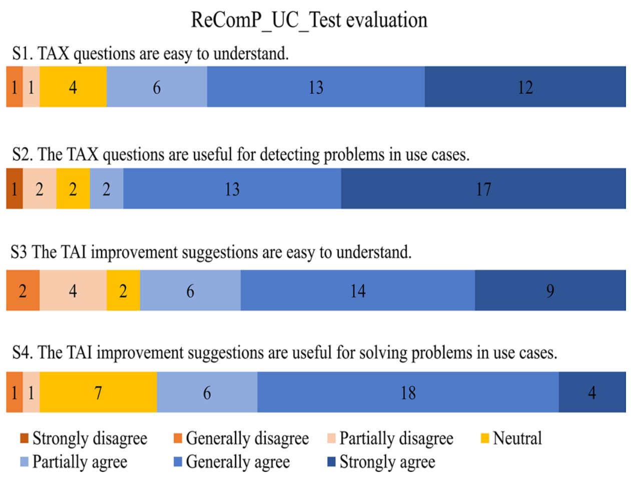


Fig. 10 ReComP_UC_Test Evaluation

7.1.2 Requirements specification improvements

As for the improvements made in the use cases using the TAI_US_Test, we analyzed the quality of the improvement made by the participants. We checked the information marked in the guided TAX_US_Test form and checked to see if the participants made any improvement in the use case.

By analyzing the use cases, we classified them as: "Great" (76%–100% improvement) the use cases that the participants made all the improvements pointed out in the guided form, "Good" (51%–75% improvement) the use cases that the participants failed to make a few improvements, in other words, they indicated the problem in the TAX, "Bad" (<=50% improvement) for the use cases that the participants made half of the improvements pointed out, and "Not recommended" (0% improvement) for the use cases that participants made incorrect improvements (improvements that modified the specified requirements) or did not make any of the improvements pointed out. Figure 9 shows the evaluation of the use cases.

Overall, all participants made improvements in the use cases using the TAI_UC_Test; 16 (43%) use cases were evaluated as "Great", 20 (54%) use cases evaluated "Good", only 1 (3%) use case was evaluated as "Bad". No use cases identified as "Not recommended" were identified.

7.1.3 Final analysis of the participants' perception when using ReComP_UC_Test

After the quantitative analysis, we evaluated ReComP's acceptability. In this questionnaire, the participants provided their degree of agreement concerning the usefulness and ease of use of ReComP. Figure 10 shows the participants' perceptions of ReComP_UC_Test.

The results shown in Fig. 10 reveal mainly positive perceptions and a few negative perceptions. There is a slightly negative perception that points to some participants who felt that ReComP is neither easy nor useful for improving requirements communication. In the following section, the participants' perception of TAX and TAI will be discussed in detail.

7.1.3.1 TAX_UC_Test Regarding the ease of use of TAX_UC_Test (*S1. TAX questions are easy to understand*), we can see in Fig. 10 that 60% of the participants found the questions in the guided TAX_UC_Test form easy to understand. Furthermore, the participants reported that the guided form helped them to identify problems in the use cases through objective and precise questions, as shown by the following quotes:

> "Short, straightforward sentences, directly addressing the problem. It made it easier."—P5
> "The questions were objective and helped me to find the errors."—P28

Only 3% of the participants disagreed that the questions found in the guided TAX_UC_Test form were easy to understand.

> "… maybe the questions do not help you very much if you do not understand the subject well"—P10
> "… it ends up being annoying to have to read many questions"—P32

Figure 10 also points out that 37% of the participants had no clear opinion or were neutral about the ease of using TAX_UC_Test.

As for the usefulness of TAX_UC_Test, we analyzed the answers to statement (*S2. The TAX questions are useful for detecting problems in use cases*), and the results showed that 62% of the participants agreed that the questions in the TAX_UC_Test were useful for detecting problems in use cases. A total of 33% of the participants had no clear opinion or were neutral about the usefulness of the TAX_UC_Test.

Only 5% of the participants disagreed with the statement that the questions found in the TAX_UC_Test were useful. Regarding the positive points of TAI's usefulness, we can highlight the following quotes from the participants:

> "An excellent guide, it basically works as a work instruction."—P3
> "The questions direct the tester to points that are essential in any project."—P20

Regarding the negative points in relation to the usefulness of TAX, we can highlight the following quotes from the participants:

> "Useless, this document must be delivered before the TAI, there is just one more questionnaire to be answered"—P9.
> "… TAX depends a little on the opinion of those who are reading the use case, that is, if I don't think something is a problem, I will not mark it as a problem on TAX. So, the usefulness of TAX depends on the perception of those who use it."—P27.

7.1.3.2 TAI_UC_Test Regarding the ease of use of the TAI_UC_Test, we analyzed the answers to statement (*S3. The TAI improvement suggestions are easy to understand*). The results showed that 81% of the participants agreed that the suggestions to improve TAI_UC_Test were easy to understand. Only 16% of the participants had no clear opinion or were neutral in regards to how easy it is to understand the TAI_UC_Test, and 3% of the participants disagreed with the statement. As for the positive points relating to the ease of use TAI, we can highlight the following quotes from the participants:

> "As TAI is practically a direct mapping of TAX, it is straightforward to understand and apply the improvements."—P10
> "TAI presents solutions clearly and objectively, so it is elementary to understand and know how to proceed to improve the use case."—P30

Regarding the negative points of the ease of use TAI, we can highlight the following quotes from the participants:

> "In certain instances, TAI only indicates that we have to create a new field, but it does not help us to identify for sure who the actors are, for example."—P16
> "I would like you to have more solution options in each section, to cover more cases and choose from possible solutions that better solve the error."—P23

As for the TAI_UC_Test's usefulness, we analyzed the answers to the statement (*S4. The TAI improvement suggestions are useful for solving problems in user stories*). The results showed that 68% of the participants found the TAI_UC_Test improvement suggestions useful for improving use cases. Only 29% of the participants had no clear opinion or were neutral as to the usefulness of TAI_UC_Test, and 3% of the participants disagreed with the statement that the suggestions for improvement were useful. Regarding the positive points in relation to TAI's usefulness, we can highlight the following quotes from the participants:

> "They are handy for solving problems once they have been identified in TAX."—P11
> "TAI shows exactly where in the UC, the changes should take place."—P14

Regarding the negative points in relation to the usefulness of TAI, we can highlight the following quotes from the participants:

> "I do not usually need TAI since, in general, for the problems I identify, I immediately think of a solution."—P3
> "Generally, the solutions seem to be more complicated than we think. The examples seem to increase the complexity of the use case instead of showing a simpler path."—P13

7.1.3.3 General perception of ReComP As for the general perception of ReComP for evaluating and improving the use cases, the participants who played the role of testers pointed out that it was useful and straightforward. Furthermore, ReComP helped in writing of use cases, served as an inspection and error prevention checklist, as well as making the specification more complete and simpler, as shown in the quotes below:

> "I found it a good tool in the process of refining the use case and also fixing all my knowledge related to it. However, I found the methodology, with regard to the structure (on paper, the spreadsheet), in a way, too rigid. TAX and TAI could be unified in software where these mappings could be done in a more automated way. I think it would make it a lot easier, less paper, etc. And it would be good, since it would already be in the environment that the developer uses, i.e., the computer."—P15.
> "I would use TAI again, as it helped to correct errors, but it works better with TAX, which helps to find errors. With both, we can make a better use case."—P25

7.1.3.4 Improvement suggestions Regarding suggestions for improvements in ReComP_UC_Test, the participants pointed out that it would be better if the ReComP were digital, as we can note in their quotes below:

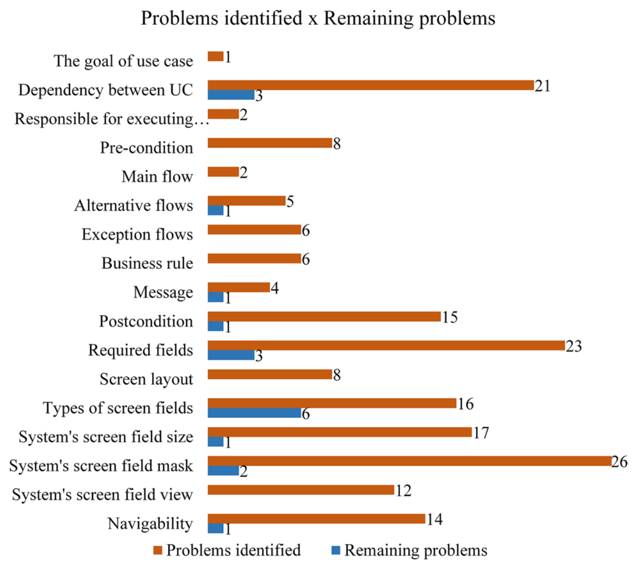


Fig. 11 Problems identified vs. Remaining problems

"TAI is important to allow people who are "lost" in not knowing how to solve some problems, but it is not very practical, perhaps because it is a table full of information thrown at you on a piece of paper. Perhaps it would be more relevant if it were digital, and it was possible to search only what is of interest to you. Perhaps even with some suggestions for more common mistakes."—P16.
"… it would be easier to be an online tool."—P29

7.2 Discussion

According to the analysis of the results achieved with TAX (using the real specification without changes and without worrying about the team member's role) and in the evaluation with TAI (using the specification with improvements concerning the role), we can see in Fig. 11 that for all 17 problems found in the specification of the use case in Round 1, the frequency decreased after the improvements made in the user stories (100%). More specifically, it is noteworthy that 8 (47%) problems were completely remedied after their identification using TAX_US_Dev and the application of improvements using TAI_US_Dev. However, 9 (53%) problems were still identified in Round 2, quantifying a much smaller amount than in Round 1.

Analyzing the remaining problems, the information that participants had trouble identifying in the use case specification is typically not part of the standard use case template. Information such as the screen field size or its mask does not need to be included in the UC. They could be available in another specification artifact, for example, in the database, UI mockups, or architecture document.

Regarding the information needed for testers, we can see that only eight fields of information were considered unnecessary for the execution of activities. Only 1 (3%) participant pointed out that he does not need the primary scenario information, screen layout, types of screen fields, the mask of screen fields. Another 2 (5%) participants pointed out that navigability information is unnecessary, 3 (8%) participants said that post-condition information and screen field size is unnecessary and, finally, 5 (14%) participants said that information dependency between use cases are not necessary for the performance of their activities.

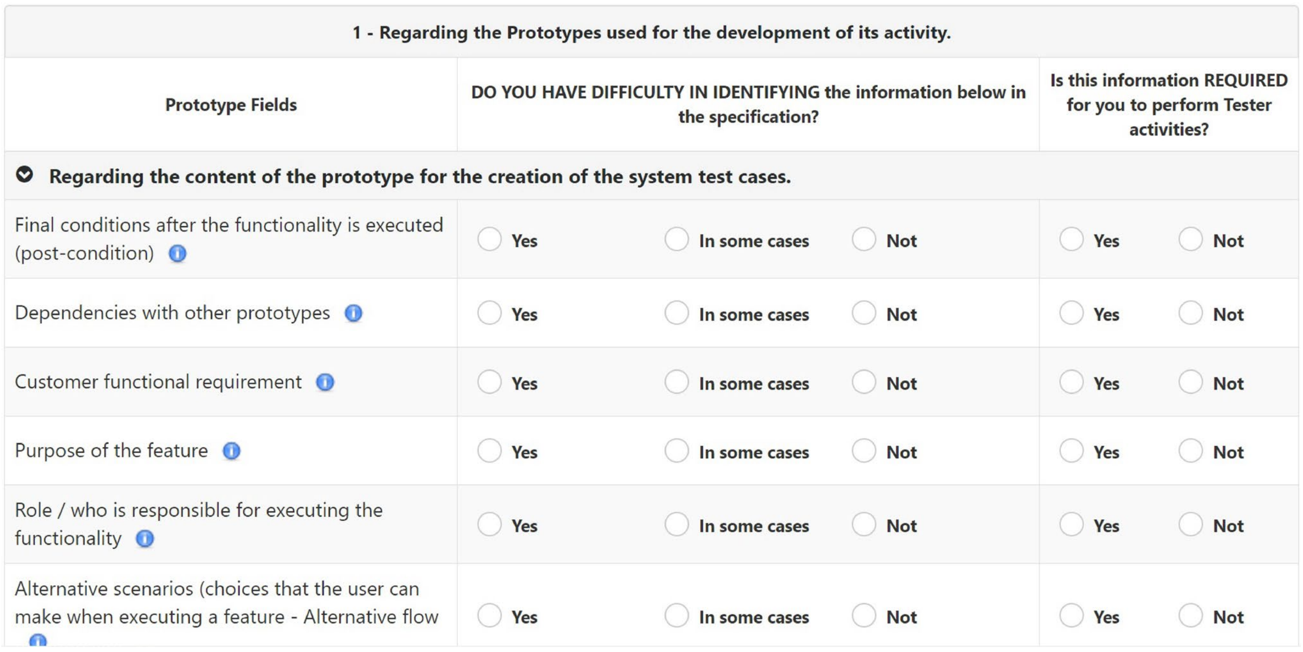
Regarding the use case improvements using the TAI_UC_Test, we found that 36 (97%) participants made improvements considered to be "excellent" and "good" in the use case, and only 1 (3%) participant made the improvements in the use case considered "bad." The high rate of positive improvements can be explained by the ease of editing of the use case being performed on the computer and the availability of the original version. The participant who did not make all the improvements pointed out in the use case generated 3 of the remaining problems: message, post-condition, and dependency among UCs.

The results of this empirical study allowed us to note that, in general, ReComP (v2) was well-received by the majority of participants in terms of ease of use and utility in identifying problems in use cases (TAX_UC_Test) and in regards to suggestions for improvement in use cases (TAI_UC_Test). The fact that the participants demonstrated positive acceptance of ReComP may indicate that this technique is also suitable for development teams that want to evaluate and improve their use cases to communicate requirements.

One of the prerequisites for using ReComP is the knowledge that the team must have of the template adopted for the requirements specification. It is worth mentioning that the reason for applying the ReComP framework in such a way as we do (by first applying TAX and then TAI) is because it was not created to impose a template on the teams. Its purpose is to verify whether the specification used in the project meets the informational requirements of requirements or not and, if so, to propose improvements to meet them.

For beginner teams, TAI can do the work of an experienced requirements engineer, since it has a set of the main problems found in communicating requirements and suggestions to solve these problems. On the other hand, TAX can be considered a quality assurance analysis, as it provides the necessary means to analyze the artifacts used in the communication of requirements and indicates whether they contain all the information necessary for a given role within the team.

Despite the wide acceptance of ReComP (v2) by the participants, we considered the negative points and suggestions



1 - Regarding the Prototypes used for the development of its activity.

Prototype Fields	DO YOU HAVE DIFFICULTY IN IDENTIFYING the information below in the specification?			Is this information REQUIRED for you to perform Tester activities?	
Regarding the content of the prototype for the creation of the system test cases.					
Final conditions after the functionality is executed (post-condition)	○ Yes	○ In some cases	○ Not	○ Yes	○ Not
Dependencies with other prototypes	○ Yes	○ In some cases	○ Not	○ Yes	○ Not
Customer functional requirement	○ Yes	○ In some cases	○ Not	○ Yes	○ Not
Purpose of the feature	○ Yes	○ In some cases	○ Not	○ Yes	○ Not
Role / who is responsible for executing the functionality	○ Yes	○ In some cases	○ Not	○ Yes	○ Not
Alternative scenarios (choices that the user can make when executing a feature - Alternative flow	○ Yes	○ In some cases	○ Not	○ Yes	○ Not

Fig. 12 ReComP screen—Team view

for improvement reported by the participants for a new version of ReComP (v3).

7.3 ReComP improvements (v3)

The main problems identified by the participants concerning the TAX_UC_Test were as follows: the guided form is too long and difficult to use because it is not a software. Thus, we realized that Round 1 (ReComP should allow team members to identify problems in the artifacts used to communicate requirements within the software development team) had not been fully achieved. Therefore, we revised the guided form, developed the ReComP software, taking care to reduce the guided form and make it more objective. In addition, we improved the dynamics of the ReComP application.

Regarding Round 2 (ReComP should provide suggestions for improvements to the requirements communication problems found in the artifacts), we noted that there is a need to present TAX and TAI together to facilitate their use. Therefore, we reviewed the dynamics of the ReComP application and made the application via software simpler.

7.4 ReComP-web

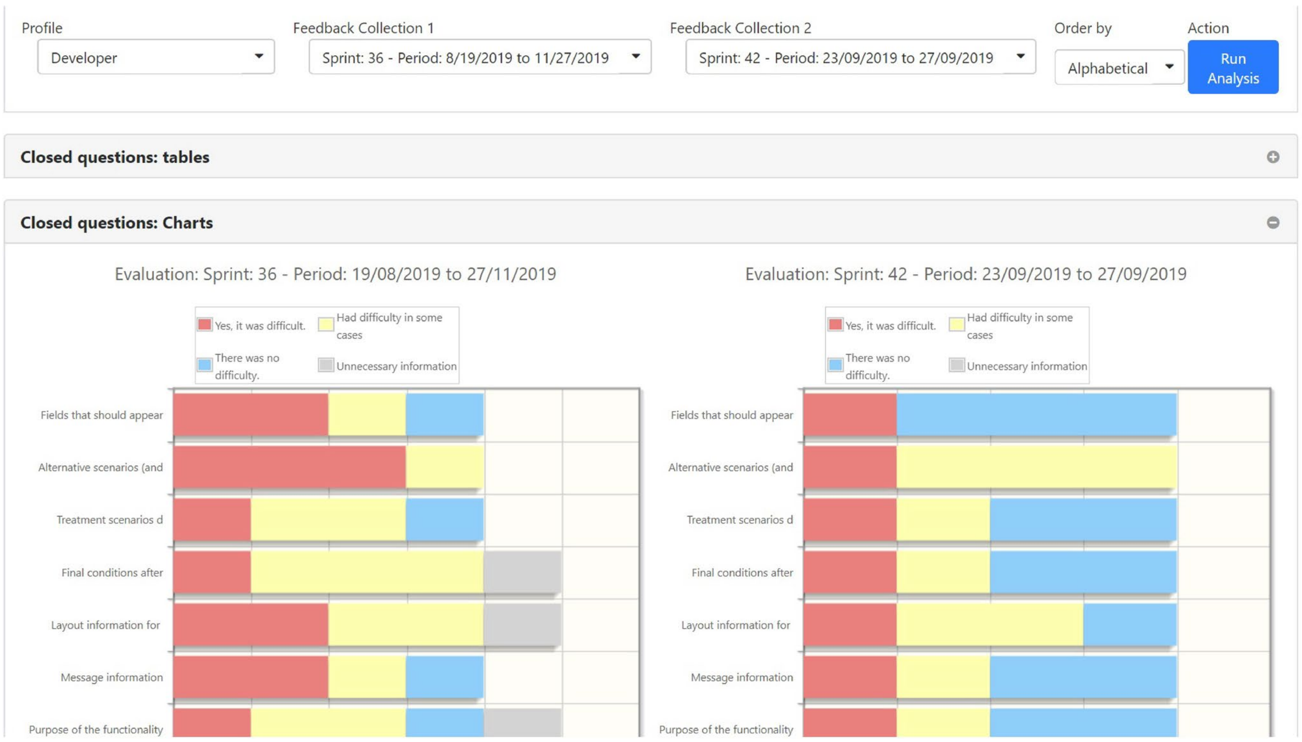
In the two ReComP design cycles, we applied the framework manually, using several artifacts defined in each study. To meet the improvement requested in the last cycle (Sect. 7.3), a web tool was created to automate the process of applying ReComP based on the artifacts that support TAX and TAI.

Along with the development of the tool, we designed a new application dynamic for the new version of ReComP. The new dynamic was divided into three steps: (1) Creation of the evaluation of the requirements specification artifact; (2) Evaluation of the requirements specification artifact by the team members (TAXs), and (3) Verification of the results of the team evaluation and suggestion for improvements (TAIs).

In Step 1, the requirements engineer creates a new assessment. In this step, the requirements engineer registers the project data, current assessment data, and which team members will be part of the assessment. In Step 2, the participants defined in the "Team members" section will receive a link, via email, to access the guided TAX form corresponding to their role within the team, and the artifact used to specify the requirements in the project.

In Step 3, the requirements engineer receives the answers given by the team members and evaluates the problems pointed out by them in the specification. Furthermore, he/she receives the problems sorted by name or frequency. The requirements engineer also has, on the same screen, access to improvement suggestions, provided by TAI, which correspond to each problem pointed out in the artifact analyzed by the team members. Based on this information, requirement engineers adjust the requirements of communication artifacts to minimize the highlighted problem. These adjustments may or may not follow the improvement suggestions provided by the tool.

To check whether the artifact has improved from the team's point of view, Steps 1 and 2 must be redone, and


Profile: Developer

Feedback Collection 1: Sprint: 36 - Period: 8/19/2019 to 11/27/2019

Feedback Collection 2: Sprint: 42 - Period: 23/09/2019 to 27/09/2019

Order by: Alphabetical

Action: Run Analysis

Closed questions: tables

Closed questions: Charts

Evaluation: Sprint: 36 - Period: 19/08/2019 to 27/11/2019

Evaluation: Sprint: 42 - Period: 23/09/2019 to 27/09/2019

Fig. 13 ReComP screen—Requirements engineer view

Step 3 should give a new result regarding the ability of the artifacts to communicate the requirements to the members of the development team. It is worth mentioning that, if needed, the three steps can be repeated in specification improvement cycles. The number of cycles can be repeated as many times as the person responsible for the evaluation (requirements engineer) deems necessary.

Figure 12 presents part of the ReComP-web screens with more objective questions for the team members. Figure 13 presents part of the evaluation report of the requirements specification artifact used in the project. It is noteworthy that we intend to evaluate ReComp-web in future studies in the industry.

8 Validity threats and limitations

As in all studies, some threats can affect the validity of the results. In this section, we discuss the existing threats in the two DSR cycles and, when possible, the manner by which they have been mitigated. The main threats, based on Wohlin et al. [54], were as follows:

1. The validity of the artifact evaluated as a representative artifact—this was minimized with the use of a use case specification developed by industry professionals and corresponding to a real system;
2. Representativeness of the participants—the study participants were undergraduate students and did not play the developers' and testers' role in the industry. However, studies, such as Salman et al. [55], Höst et al. [56] and Runeson [57], show that students can adequately represent a population of industry professionals. The results found in this study are encouraging and indicate that ReComP could similarly be useful for professionals. Therefore, we believe that ReComP is also suitable for professionals, although a study with that goal is still to be executed. The results may not be generalizable to experienced developers and testers.
3. Sample size: due to the limited number of participants, there is a limitation in the conclusion of the results and, as such, they are considered indicative and not conclusive.
4. Evaluation apprehension—this threat was mitigated by conducting evaluations without personal questions, only about requirements specifications used to perform the activities. Additionally, we carried out the entire evaluation anonymously, and all participants were volunteers and could withdraw from the study at any time.
5. Subject drops out of the study—To minimize the impacts on the results, we discarded incomplete data of the subjects who withdrew from the study.

6. Researcher's influence on the results—to minimize this threat, all data collected were reviewed and analyzed jointly with two other researchers.

In addition to the threats to the validity of the study results, it is worth mentioning that the proposed ReComP has three limitations:

1. It does not evaluate the quality of the requirements elicitation. It does not verify whether the elicitation was carried out correctly. ReComP focuses on evaluating and improving the communication of software requirements between members of the development team, considering the artifacts used, as well as the informational needs of each role involved.
2. Guided evaluation forms are limited to the roles of developers and testers. The reason for this limitation was due to the number of publications found involving the informational needs of these two roles.
3. The proposed standards are specific to the following artifacts: UI mockups, use case description, and user story. The reason for choosing these three artifacts was due to the number of publications found that were related to the subject and because they are widely used by the software industry to communicate requirements.

9 Conclusion

This paper introduced ReComP, a framework for evaluation and improvement of artifacts used by development teams to communicate requirements in software projects. ReComP was developed following the Design Science Research method. The framework aims to support requirements communication based on the perspective of the software development team members, by: (a) assisting in identification of problems in the artifacts used for communicating requirements within the software development team, (b) identifying the informational needs for each role of the development team, and (c) proposing suggestions for improvements to problems found in the requirements communication. We conducted two studies to assess the use of ReComP from the perspective of developers by using user stories (first DSR cycle) and testers by using use cases (second DSR cycle).

In the first DSR cycle, we carried out an empirical study to verify the difficulties faced by developers when building software UI mockups using user stories. For this, we used ReComP to evaluate the specification (TAX_US_Dev) and to suggest improvements for the artifact (TAI_US_Dev). TAX_US_Dev proved to be effective in helping to identify problems in the specification. TAI_US_Dev proved to be effective in helping to improve the specification to meet the developer's informational needs.

With the application of just two rounds of ReComP, we could observe a decrease in the number of problems found by developers in the requirements specification. This result motivates us to hypothesize that creating specifications by considering the roles of users of this specification (developers) tends to decrease the number of defects in the development, since the information necessary for the developer to perform his activities will be present in the specification. In this study, ReComP_US_Dev had a positive acceptance regarding its ease of use and usefulness. Some improvement points were suggested and were implemented in the second version of ReComP.

In the second DSR cycle, we conducted another empirical study to verify the difficulties faced by testers when constructing test cases using use cases and UI mockups. For this, we used ReComP to evaluate the specification (TAX_UC_Test) and to suggest improvements in the artifact used for requirements communication (TAI_UC_Test). The TAX_UC_Test proved to be effective in helping to identify problems in specifications used by testers. The TAI_UC_Test proved to be effective in helping to improve the specification to meet the informational needs of testers.

In this last study, we only performed one round. The results showed that due to the use case specification having more information than the user story, it thus presents a higher level of detail for the tester and, consequently, there was a low rate of problems identified by the application of TAX_US_Test. Furthermore, the ReComP_UC_Test has also received positive acceptance for its ease of use and usefulness. Additionally, the participants suggested some points of improvement that will be implemented in future versions of ReComP.

Assessing the preference between the ReComP versions (after the improvements presented in Sect. 6.3), we highlight that the second version of the framework proved to be more widely accepted. It is believed that this result is a consequence of reducing the size of the guided forms, and changing discursive questions to closed, standardized, and direct questions.

9.1 Discussion

Requirement information transmitted in an incomplete, inaccurate, or incorrect manner, and undocumented requirements changes often cause incorrect functionality, unimplemented software functionality, and rework. ReComP aids in the improvement of requirements communication through the supply of artifacts capable of identifying problems in the requirement specification by considering the needs of the team (TAX), suggesting improvements for the identified problems (TAI), and evaluation of the results achieved (ReComP-web).

The results presented in Sect. 2 identify problems in the specifications and proposal for improving the specifications without considering the informational needs regarding requirements that the users of the specification (team members) use to carry out their activities in the project. We emphasize the importance of understanding the informational needs related to requirements for each team member, since each member has a role in system development. Given this, we created ReComP to meet the perspectives of developers and testers in the use case, user stories, and prototypes specifications.

Nevertheless, it is essential to establish a good practice of knowing the user's needs for requirements information. Detailed knowledge of the team's requirements information needs is a valuable aid in the requirements engineer's decision-making process when eliciting and specifying software requirements. Once the team's needs are known, the requirements engineer can obtain information from the customer and transmit the necessary requirements information to the team so that it can perform its activities more clearly and effectively.

ReComP can be used in different software development processes since its objective is to evaluate and improve the requirements specification artifacts used to communicate requirements between the development team members. ReComP does not analyze the artifacts concerning the rules and standards established by the development processes, but it does analyze whether the specifications can communicate the requirements informational needs that members of the team need to perform their activities on the project.

In both DSR cycles, ReComP proved viable in application, effective in evaluating and improving requirements specifications, and it was considered easy to use by most participants. The results also show that its application has benefited the participants in improving the software development team's requirements communication.

The use of ReComP can benefit software development companies that want to: (a) meet the informational needs of the team's requirements; (b) evaluate how much the specification used in the project meets the identified needs; (c) find solutions to improve the problems encountered, and (d) adjust the project specification to meet the team's demand.

9.2 Future works

To consolidate the results that were found and improve the framework, in the future, we intend to test ReComP-web in software development companies in order to identify other difficulties faced by developers and testers in the requirements specification artifacts used by them in the software development project. As such, we intend to benefit the participating companies by identifying problems in their specifications and improving them to meet their team members' informational needs. In addition, we intend to extend the ReComP framework to support other specification artifacts (e.g., data model and UML diagrams), considering the perspectives of different team members (e.g., architects, designers, and usability specialists).

Concerned with the training of requirements engineers, we also intend to investigate the use of ReComP as an instrument to support teaching requirements specification techniques and document inspection techniques. In addition, one hypothesis to be investigated in the future is whether the application of ReComP can create empathy among future software engineers and the team members by attending to the informational needs in the specifications created by them to communicate requirements.

9.3 ReComP contributions

ReComP can benefit the industry to identify problems in the communication of requirements specifications considering the informational needs of developers and testers in regards to requirements. Furthermore, it can provide improvement suggestions that are commonly used by other companies to solve problems.

Based on related work and improving requirements communication between members of the development team, ReComP presents itself as a tool that is capable of assisting in the identification and improvement of artifacts (use case, user story and UI mockups) used to document and communicate requirements, considering the informational needs of the roles (developer and tester) that participate in a software development team.

Acknowledgments This research, according to Article 48 of Decree nº 6.008/2006, was partially funded by Samsung Electronics of Amazonia Ltda, under the terms of Federal Law nº 8.387/1991, through agreement nº 003/2019, signed with ICOMP/UFAM. This work was supported by the Foundation for Research Support of the State of Amazonas (FAPEAM)—POSGRAD, and the Coordination for the Improvement of Higher Education Personnel—Brazil (CAPES)—Finance Code 001. Besides, we recognize the financial support provided by CNPq through process 314174/2020-6 and FAPEAM 062.00150/2020. UNIRIO through process no. PPQ-UNIRIO 01/2019 e 04/2020. We thank all participants in the empirical study and the researchers of the USES-UFAM research group for their valuable contributions in carrying out this study.

Author contributions Ana Carolina Oran: Conceptualization, Methodology, Formal analysis, Investigation, Data Curation, Writing—Original Draft, Visualization. Gleison Santos: Validation, Writing—Review and Editing, Supervision. Bruno Gadelha: Methodology, Verification, Validation, Writing—Review and Editing. Tayana Conte: Validation, Writing—Review and Editing, Supervision.

Funding Not applicable.

Availability of data and material Technical report available [52].

Declarations

Conflicts of interest The authors declare that they have no conflict of interest.

References

1. Bjarnason E, Sharp H (2017) (2017) The role of distances in requirements communication: a case study. Requirements Eng 22(1):1–26. https://doi.org/10.1007/s00766-015-0233-3
2. Fricker SA, Schneider K, Fotrousi F, Thuemmler C (2016) Workshop videos for requirements communication. Requirements Eng 21(4):521–552. https://doi.org/10.1007/s00766-015-0231-5
3. Bjarnason E, Wnuk K, Regnell B (2011) Requirements are slipping through the gaps—a case study on causes and effects of communication gaps in large-scale software development. In: 19th international requirements engineering conference. IEEE. pp 37–46.
4. Begel A, Zimmermann T (2014) Analyze this! 145 questions for data scientists in software engineering. In: Proceedings of the 36th International Conference on Software Engineering. ACM, pp 12–23. https://doi.org/10.1145/2568225.2568233
5. Méndez Fernández D, Wagner S, Kalinowski M, Felderer M, Mafra P, Vetrò A, Conte T et al (2017) Naming the pain in requirements engineering. Empir Softw Eng 22:1–41. https://doi.org/10.1007/s10664-016-9451-7
6. Ali SW, Ahmed QA, Shafi I (2018) Process to enhance the quality of software requirement specification document. In: 2018 International Conference on Engineering and Emerging Technologies (ICEET). IEEE, pp 1–7. https://ieeexplore.ieee.org/document/8338619/
7. Reggio G, Leotta M, Ricca F, Clerissi D (2018) DUSM: a method for requirements specification and refinement based on disciplined use cases and screen mockups. J Comput Sci Technol 33(5):918–939. https://doi.org/10.1007/s11390-018-1866-8
8. Anda B, Hansen K, Sand G (2009) An investigation of use case quality in a large safety-critical software development project. Inf Softw Technol 51(12):1699–1711. https://doi.org/10.1016/j.infsof.2009.04.005
9. Cockburn A (2001) Writing effective use cases, vol 1. Addison-Wesley, Boston
10. Mohagheghi P, Anda B, Conradi R (2005) Effort estimation of use cases for incremental large-scale software development. Intl Conf Softw Eng. https://doi.org/10.1109/ICSE.2005.1553573
11. Schon E-M, Winter D, Escalona MJ, Thomaschewski J (2017) Key challenges in agile requirements engineering. In: Baumeister H, Lichter H, Riebisch M (eds) Agile processes in software engineering and extreme programming. Springer, Cham, pp 37–51. https://doi.org/10.1007/978-3-319-57633-6_3
12. Schön EM, Thomaschewski J, Escalona MJ (2017) Agile requirements engineering: a systematic literature review. Computer Standards and Interfaces 49:79–91. https://doi.org/10.1016/j.csi.2016.08.011
13. Gilson F, Irwin C (2018) From user stories to use case scenarios towards a generative approach. In: Australasian Software Engineering Conference (ASWEC). IEEE, pp 61–65.
14. Jia J, Yang X, Zhang R, Liu X (2019) Understanding software developers' cognition in agile requirements engineering. Sci Comput Program 178:1–19. https://doi.org/10.1016/j.scico.2019.03.005
15. Lucassen G, Dalpiaz F, van der Werf JME, Brinkkemper S (2016) Improving agile requirements: the quality user story framework and tool. Requirements Eng 21(3):383–403. https://doi.org/10.1007/s00766-016-0250-x
16. Ricca F, Scanniello G, Torchiano M, Reggio G, Astesiano E (2014) Assessing the effect of screen mockups on the comprehension of functional requirements. ACM Trans Softw Eng Methodol (TOSEM) 24:1–38. https://doi.org/10.1145/2629457
17. Hoisl B, Sobernig S, Strembeck M (2014) Comparing three notations for defining scenario-based model tests: a controlled experiment. In: International Conference on the Quality of Information and Communications Technology (QUATIC 2014), pp 180–189. https://doi.org/10.1109/QUATIC.2014.62
18. Gross A, Doerr J (2012) What you need is what you get!: the vision of view-based requirements specifications. In: IEEE International Requirements Engineering Conference (RE). IEEE, pp 171–180. https://doi.org/10.1109/RE.2012.6345801
19. Liskin O (2015) How artifacts support and impede requirements communication. In: International working conference on requirements engineering: foundation for software quality, vol 9013. Springer, Cham, pp 132–147. https://doi.org/10.1007/978-3-319-16101-3_9
20. Tu Y-C, Tempero E, Thomborson C (2016) An experiment on the impact of transparency on the effectiveness of requirements documents. Empir Softw Eng 21(3):1035–1066. https://doi.org/10.1007/s10664-015-9374-8
21. Johannesson P, Perjons E (2014) An introduction to design science. Springer. https://doi.org/10.1007/978-3-319-10632-8
22. Hevner A, Chatterjee S (2010) Design Research in Information Systems. Design Research in Information Systems. Springer, Boston, pp 9–22. https://doi.org/10.1007/978-1-4419-5653-8_2
23. Wieringa R (2014) Design science methodology for information systems and software engineering. Springer, Berlin
24. Bjarnason E, Unterkalmsteiner M, Borg M, Engström E (2016) A multi-case study of agile requirements engineering and the use of test cases as requirements. Inf Softw Technol 77:61–79. https://doi.org/10.1016/j.infsof.2016.03.008
25. Méndez Fernández D, Wagner S, Lochmann K, Baumann A, de Carne H (2012) Field study on requirements engineering: Investigation of artefacts, project parameters, and execution strategies. Inf Softw Technol 54(2):162–178. https://doi.org/10.1016/j.infsof.2011.09.001
26. Medeiros J, Vasconcelos A, Silva C, Goulão M (2018) Quality of software requirements specification in agile projects: a cross-case analysis of six companies. J Syst Softw 142:171–194. https://doi.org/10.1016/j.jss.2018.04.064
27. Lauesen S, Kuhail MA (2012) Task descriptions versus use cases. Requirements Eng 17(1):3–18. https://doi.org/10.1007/s00766-011-0140-1
28. Wang X, Zhao L, Wang Y, Sun J (2014) The role of requirements engineering practices in agile development: an empirical study. In: Proceedings of the Asia Pacific Requirements Engineering Symposium. CCIS Springer 432:195–209. https://doi.org/10.1007/978-3-662-43610-3_15
29. Tiwari S, Gupta A (2015) A systematic literature review of use case specifications research. Inf Softw Technol 67:128–158. https://doi.org/10.1016/j.infsof.2015.06.004
30. Phalp KT, Vincent J, Cox K (2007) Assessing the quality of use case descriptions. Software Qual J 15(1):69–97. https://doi.org/10.1007/s11219-006-9006-z
31. Zeaaraoui A, Bougroun Z, Belkasmi MG, Bouchentouf T (2013) User stories template for object-oriented applications. In: Innovative Computing Technology (INTECH), IEEE, pp 407–410. https://doi.org/10.1109/INTECH.2013.6653681
32. Inayat I, Salim SS, Marczak S, Daneva M, Shamshirband S (2015) A systematic literature review on agile requirements engineering practices and challenges. Comput Hum Behav 51:915–929. https://doi.org/10.1016/j.chb.2014.10.046

33. Cohn M (2004) User stories applied: for agile software development. Addison-Wesley Professional, Boston
34. Soares HF, Alves NSR, Mendes TS, Mendonça MG, Spínola RO (2015) Investigating the link between user stories and documentation debt on software projects. In: International Conference on Information Technology. IEEE, pp 385–390. https://doi.org/10.1109/ITNG.2015.68
35. Knight W (2018) Business Objectives vs. user goals. UX for Developers. https://doi.org/10.1007/978-1-4842-4227-8_3
36. Baumer D, Bischofberger W, Lichter H, Zullighoven H (1996) User interface prototyping-concepts, tools, and experience. In: Proceedings of IEEE 18th International Conference on Software Engineering, pp 532–541. https://doi.org/10.1109/ICSE.1996.493447
37. De Lucia A, Qusef A (2010) Requirements engineering in agile software development. J Emerg Technol Web Intell 2(3):212–220
38. Blomkvist JK, Persson J, Åberg J (2015) Communication through boundary objects in distributed agile teams. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. ACM, 2015. pp 1875–1884. https://doi.org/10.1145/2702123.2702366
39. Preece J, Sharp H, Rogers Y (2015) Interaction design: beyond human-computer interaction, 4th edition. Wiley, Hoboken
40. Walker M, Takayama L, Landay JA (2002) High-fidelity or low-fidelity, paper or computer? Choosing attributes when testing web prototypes. In: Proceedings of the human factors and ergonomics society annual meeting, vol 46(5), pp 661–665. https://doi.org/10.1177/154193120204600513
41. Ferreira J, Noble J, Biddle R (2007) Agile development iterations and UI design. Proc Agile 2007:50–58. https://doi.org/10.1109/AGILE.2007.8
42. Reggio G, Ricca F, Leotta M (2014) Improving the quality and the comprehension of requirements: disciplined use cases and mockups. In: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications. IEEE, pp 262–266. https://doi.org/10.1109/SEAA.2014.79
43. Hess A, Diebold P, Seyff N (2019) Understanding information needs of agile teams to improve requirements communication. J Ind Inf Integr 14:3–15. https://doi.org/10.1016/j.jii.2018.04.002
44. Dresch A, Lacerda DP, Antunes Júnior JAV (2015) Design science research: método de pesquisa para avanço da ciência e tecnologia. Bookman, Porto Alegre
45. Hevner AR (2007) A three-cycle view of design science research. Scand J Inf Syst 19(2): 87–92. https://aisel.aisnet.org/sjis/vol19/iss2/4
46. Oran AC, Nascimento E, Santos G, Conte T (2017) Analysing requirements communication using use case specification and user stories. In: Proceedings of the 31st Brazilian Symposium on Software Engineering. Association for Computing Machinery, New York, NY, USA, pp 214–223. https://doi.org/10.1145/3131151.3131166
47. Oran AC, Valentim N, Santos G, Conte T (2019) Why use case specifications are hard to use in generating prototypes? IET Softw 13(6):510–517. https://doi.org/10.1049/iet-sen.2018.5239
48. Ibriwesh I, Ho S-B, Chai I, Tan C-H (2017) A controlled experiment on comparison of data perspectives for software requirements documentation. Arab J Sci Eng 42(8):3175–3189. https://doi.org/10.1007/s13369-017-2425-2
49. Thuan NH, Drechsler A, Antunes P (2019) Construction of design science research questions. Commun Assoc Inf Syst 44.1:332–363. https://doi.org/10.17705/1CAIS.04420
50. Jiang L, Eberlein A (2008) A framework for requirements engineering process development (FRERE). In: Proceedings of the 19th Australian Conference on Software Engineering, pp 507–516. https://doi.org/10.1109/ASWEC.2008.4483240
51. Hassenzahl M (2008) User experience (UX): towards an experiential perspective on product quality. In: Proceedings of the 20th French-Speaking Conference on Human-Computer Interaction '08, ACM Press, pp 11–15. https://doi.org/10.1145/1512714.1512717
52. Oran AC, Santos G, Gadelha B, Conte T (2020) A framework for evaluating and improving requirement specifications based on the developers and testers perspective—technical report. https://doi.org/10.6084/m9.figshare.12563756
53. Davis FD, Bagozzi RP, Warshaw PR (1989) User acceptance of computer technology: a comparison of two theoretical models. Manage Sci 35(8):982–1003. https://doi.org/10.1287/mnsc.35.8.982
54. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer, Berlin, Heidelberg, p 9783642290
55. Salman I, Misirli AT, Juristo N (2015) Are students representatives of professionals in software engineering experiments?. In: 37th International Conference on Software Engineering (ICSE 2015), pp 666–676. https://doi.org/10.1109/ICSE.2015.82
56. Höst M, Regnell B, Wohlin C (2000) Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. Empir Softw Eng 5(3):201–214. https://doi.org/10.1023/A:1026586415054
57. Runeson P (2003) Using students as experiment subjects—an analysis on graduate and freshmen student data. In: Proceedings of the 7th international conference on empirical assessment in software engineering. Citeseer, pp 95–102