

# An approach based on design practices to specify requirements in agile projects

Juliana Medeiros<sup>1,2</sup>, Alexandre Vasconcelos<sup>2</sup>, Miguel Goulão<sup>3</sup>, Carla Silva<sup>2</sup>, João Araújo<sup>3</sup>

Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB)<sup>1</sup>,

Universidade Federal de Pernambuco (UFPE/CIN)<sup>2</sup>, Universidade NOVA de Lisboa (UNL/FCT/NOVALINCS)<sup>3</sup>,

juliana.medeiros@ifpb.edu.br, amlv@cin.ufpe.br, mgoul@fct.unl.pt, ctlls@cin.ufpe.br, joao.araujo@fct.unl.pt

## ABSTRACT

The agile manifesto highlights a frequent communication with the customer to detail his/her needs and to validate the software requirements through frequent software deliveries. So, the agile methods treat the Software Requirements Specification (SRS) differently from the traditional development methods. User stories are one of the most widely used approaches to specify requirements in agile projects. However, empirical studies in the industry point out that user stories are targeted to customers, only cover simple functional requirements visible to the users, and do not address system and non-functional requirements that are also required for coding, testing, and maintaining. We propose an approach to specify requirements based on design practices targeted to the developer. We conducted an industrial case study during eight months to evaluate the proposed approach. The initial findings indicate that the SRS is closer to what will be implemented, and it meets the developers' expectations.

## CCS Concepts

- Software and its engineering → Specification languages
- Software and its engineering → Agile software development

## Keywords

Requirements Specification, Agile Methods, Design Practices.

## 1. INTRODUCTION

Since the publication of the Agile Manifesto [1] the adoption of the Agile Software Development (ASD) has been growing. A survey involving about 4000 people pointed out that 45% of respondents use agile methods in the majority of projects [2].

Recent studies [3][4][5] have been conducted in the industry to investigate the challenges of the requirements engineering in ASD, such as low availability of the customer, poor quality of Software Requirements Specification (SRS), inadequate management and prioritization of the requirements. According to the Agile Manifesto [1], the validation of requirements must be made through frequent software deliveries. As a result, using the SRS to validate the customer requirements is unnecessary. Thus, the SRS should be used to support the development activities [6]. However, the User Stories (US) are written in the language of the problem domain and its format leads to a high level description of the software requirements, targeting the customer [7]. USs lack detail to support the development team [8]. Even with the

continuous presence of the client during the software development, the design information cannot be gathered because the client is not capable of perceiving it. The design information is neglected in SRS [8], making it difficult the activities of coding, testing and maintaining [9] as well as the knowledge transfer in distributed development and in high turnover teams [4][10].

USs lack expressiveness and capture only simple, customer visible, functional requirements [3]. They do not convey enough information for supporting the design, in complex or hardware-dependent systems. This focus on functional requirements often leads to overlooking technical aspects such as design constraints, making their development harder at later stage [5][11].

Traditionally, an information system is defined regarding two perspectives: one related to its function and the other to its structure [12]. The functional perspective results in a high-level description of the system's functionality from the users' point of view. From the structural perspective, a system is depicted regarding entities and static relationships. Conceptual modeling is the systematic activity of describing some aspects of the structural and social world around for purposes of understanding and communication [13]. According to Olivé [14], conceptual models are needed to achieve a common understanding of the system domain among all stakeholders. Although the conceptual modeling and mockups are established practices in traditional approaches, they are not systematically used in popular agile methods, such as scrum [15] and XP [16].

To address these issues, we propose an approach called Requirements Specification for Developers (RSD) to specify agile requirements using well-established design practices. RSD provides an integrated view of the requirements linking in a systematic way the benefits of the identification of the problem domain concepts (conceptual modeling), the visual representation of interface requirements (mockups), the business rules, nonfunctional requirements-NFR and technical constraints (acceptance criteria). The remainder of this paper is organized as follows: Section 2 summarizes the background about ASD. Section 3 details the RSD approach. Section 4 describes the evaluation of the approach and threats to validity. Section 5 discusses some related work. Finally, Section 6 presents our conclusions and directions for future work.

## 2. BACKGROUND

The Agile Manifesto establishes values and principles to guide the ASD and several practices have been proposed, as shown in "Subway map" Agile Practices [17]. From the fundamental practices, we used *Iterative Development* and *Incremental Development*. From the XP practices, we used *Iterations* and *Frequent Releases*. RSD also includes three design practices: *Modeling Concepts*, *Modeling Mockups* and *Acceptance Criteria*. The latter adds three testing practices: *BDD*, *ATDD* and *Acceptance Testing*. In the scope of this research, the design practices help to bridge the gap between the problem and the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC 2017, April 03-07, 2017, Marrakech, Morocco

© 2017 ACM 978-1-4503-4486-9/17/04...\$15.00.

DOI: <http://dx.doi.org/10.1145/3019612.3019753>

solution domains, providing a better understanding for the developer in charge to implement a feature. According to Bjarnason [9], an SRS closer to what is implemented may reduce the effort required to coding, testing and maintaining.

The requirements approaches employed in ASD, such as USs, focus on functional modeling. There is no concern in defining the conceptual model in a systematic way together with USs. Sometimes, the data entities are generated from the classes defined in the source code. It may end up creating an unstable data model. An inadequate conceptual model can hinder the inclusion of new features, the incorporation of changes and the provision of data for business intelligence systems. Some changes in the conceptual model have an enormous impact on code. Hence the importance of dedicating time to design the conceptual model. The identification of the concepts (actors) and information on the profile of each actor is proposed in [18]. However, it does not treat other concepts involved in the business domain and not address the relationship between these actors, it treats only actors.

*Mockups* are drawings that show how the user interface is supposed to look during the interaction between the software and the end-user [19]. Mockups have proven to be an efficient practice to capture and defining functional requirements. One of their advantages is that they are technically valuable for developers and, at the same time, fully understandable by end-users [20]. Although many tools exist for drawing screen mockups, several professionals prefer to sketch screen mockups on paper. Mockups improve requirements gathering, without implying an additional effort [20]. Many agile projects require user interaction design, but the integration of mockups into ASD is not well understood [21]. The popular agile methods do not use the mockups as part of their process, an exception can be found in Dynamic Systems Development Method (DSDM). So the companies need to adapt their processes to integrate the mockups in ASD.

RSD approach, described in next section, adopts the Acceptance Criteria (AC) used in USs. AC is based on the concept of the Acceptance Tests (AT) from XP [16], which defines constraints for the USs. Since we have changed some original concepts from AC, we defined the AC+ notation with the aim to distinguish it from the AC. AC+ is an atomic statement that defines any need or constraint on the operation of the system. As with AC, an AC+ is generally understood to have a binary result: pass or fail, in which a failure suggests the presence of a defect. However, the AC+ has some differences regarding the AC as show in Table 1.

**Table 1. Differences between AC and AC+**

|             | AC  | AC+   |
|-------------|---|---|
| Link        | Specific to a single user story [22];   | Can be reused by several requirements;  |
| Scope       | Focus on constraints related to the business rules[16];                         | Can be a business rule, interface, validation, technical or any other constraint; |
| Oriented to | Directed to the customer and described at high level, without much detail [23]; | Directed to developer and technical jargon can be used;                           |
| Writer      | Should be written by customers [16];  | Any stakeholder;  |
| Domain      | Problem [7].  | Problem and solution.   |

Like Test-Driven Development (TDD), Acceptance Test Driven Development (ATDD) also involves creating tests before coding, and these tests represent expected behavior of the software [17]. In ATDD, the team creates one or more acceptance-level tests for a feature before implementing it. ATDD changes the purpose of testing by creating concrete examples of business rules for clarifying and documenting requirements.

Behavior Driven Development (BDD) is a synthesis and refinement of practices stemming from TDD and ATDD [17]. BDD focuses on the behavioral aspect while the TDD focuses on the implementation aspect. BDD is usually done in a very English-like language to help the domain experts to understand the implementation rather than exposing the code level tests.

As in ATDD and BDD, AC+ uses concrete examples of complex rules as a strategy to clarify the understanding of it. However, AC+s are defined under the developer's point of view, uses a developer-oriented language and beyond of describing functional requirements, also include rules regarding the NFR and technical aspects. On the other hand, ATDD and BDD only address functional requirements.

### 3. REQUIREMENTS SPECIFICATION FOR DEVELOPER (RSD)

Literature reviews [3][24] and case studies [6][9] allowed us identifying challenges related to requirements specification in ASD, such as language customer-oriented and reliance on tacit requirements knowledge. We proposed the RSD approach to encompass design practices in the SRS to provide a better understanding to the developer about what will be implemented. In RSD approach, the customer is involved throughout the development process, describing his/her needs, prioritizing and validating the requirements. In this paper, customer is a person who buys, uses, or defines the software requirements.

In the RSD structure, customer needs and system requirements are represented using a single view that integrates three perspectives. The first perspective models the business concepts (entities, attributes, and relationships). The second describes the acceptance criteria that can represent the business rules, but also technical requirements, NFR, or any other constraint. The third describes the visual interface elements between the system and the user (mockups). This provides a wider requirements coverage, when compared to USs, which only addresses user requirements.

The conceptual modeling and the use of mockups are consolidated practices in traditional development. Although not part of the main agile methods, these practices are used in many agile projects, although neither in a systematic way [21], nor integrated into the functional requirements outlined by the US. The innovation of our approach is to systematize the use of these practices in ASD, and integrate the description of the functional and technical requirements in a single view in order to provide a SRS with the information required for coding.

#### 3.1 Metamodel

In the RSD metamodel (**Figure 1**), the *Requirement (functional or non-functional)* is identified by a *label* and has a high-level *description* that succinctly describes a user or system requirement. **Figure 2** shows an example of a requirement specified using the metamodel. Each requirement is detailed through the description of the *Concepts* (from the conceptual model), *Mockup*, and the *Acceptance Criteria* related. The *Product Backlog* (PB) contains a set of requirements that are allocated to *Sprints* according to the customer *Priority*. A requirement may have zero or more related *Mockups*. Mockups are not mandatory, since there may be requirements that do not require a visual interface with the user. An AC+ is produced by a *Stakeholder* (a client or any member of the development team), and it is verified by evaluating its *Quality Attributes*. The following assertives should be considered: (i) AC+ may be applied to more than one requirement; (ii) All

requirements must have at least one associated AC+; (iii) A requirement might have AC+ with different priorities that can be allocated to different sprints; (iv) AC+ does not need to be associated with a Mockup.

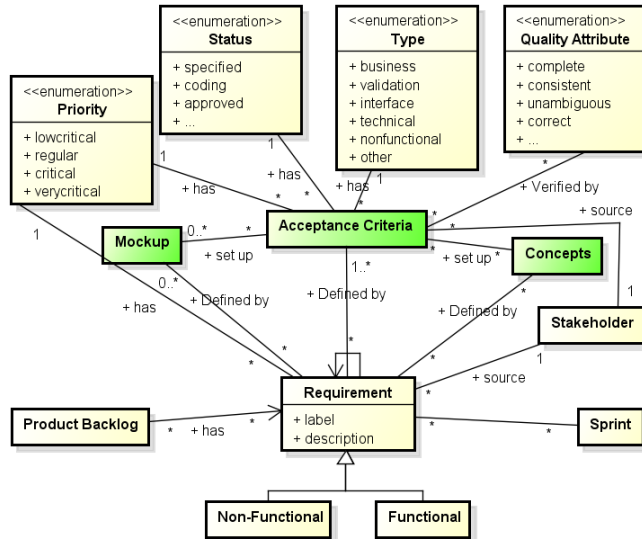


Figure 1. MetaModel of RSD

To have an SRS targeted for the developer, the AC+ includes not only business rules, but also validation rules, interface, technical or any other type of constraint necessary for the system coding. An AC+ can be categorized into six *types* as shown in Table 2.

Table 2. Acceptance Criteria Type

| Type               | Description  |
|--------------------|--|
| Business (B)       | Represents a restriction related to the intrinsic nature of the business.  |
| Validation (V)     | Represents a validation that the application needs to perform but it is not directly related to the core business. |
| Interface (I)      | Represents any restriction related to the user interface.  |
| Technical (T)      | Represents a technical restriction on how the solution should be implemented.                                      |
| Non-Functional (N) | Represents concerns about tracking quality, e.g., performance constraints, reliability constraints.                |
| Other (O)          | When it does not fit in any of the previous types.   |

## 3.2 The Practices of RSD

The practices of RSD aim to detail each requirement. The process of detailing starts with the creation of the conceptual model which should be done considering all requirements of a sprint. Then, the mockups are modeled, and the AC+s are specified, in parallel. Depending on the size of the team, the sprint backlog, and schedule, there can be multiple instances of *Modeling Mockups* and *Specify AC+* activities being carried out in parallel, one instance for each requirement of the sprint. The coding of each requirement starts as soon as the AC+ and related mockups are specified. There is no need to wait for the detailing of all sprint requirements.

### 3.2.1 Modeling Concepts

This practice aims to model the concepts (data entities) related to the requirements of a sprint. This activity is one of the differentials of RSD compared to other agile approaches which focus on behavioral modeling.

It is crucial that the modeling includes all requirements of a sprint. The requirements should not be analyzed in isolation. The joint analysis contributes to the conceptual model become more

structured to meet future changes. It is recommended that before meeting with the customer, the team reviews the PB and the notes to identify potential problems that need to be clarified with the client. Together with the customer, the analyst should sketch a model of the concepts. However, if the customer availability is limited, the team should at least make notes on the business rules so that the concepts modeling can be done later without the customer's presence. The modeling can be performed using any tool or may be drawn on paper together with the customers. Although this practice does not aim to identify AC+, if the stakeholders perceive the existence of any AC+, they must be registered in the RAC (Repository of Acceptance Criteria), even if it is not possible yet to identify to which requirement it is related to. Also, if new requirements or changes are identified, they must be registered in PB in order to be analyzed.

As opposed to what is laid down in traditional approaches, RSD approach recommends that the constraints of the technologies being used in the project such as DBMS (Database Management Systems), programming languages, and persistence frameworks should be considered in this stage. This minimizes rework in the development phase, thereby increasing agility.

The initial conceptual model is based on the requirements allocated to the first sprint and others that the team may, due to their background experience, have knowledge of or that can have an impact on the architecture. It is not mandatory to change the data entities in each sprint; it depends on the requirements allocated to the sprint.

### 3.2.2 Modeling Mockups

This practice aims to model the mockups of the requirements that require some interaction between a user and the application. Usually, each RSD has only one associated mockup, but some requirements may have no associated mockup. On the other hand, some requirements may have more than one associated mockup. In these situations, it is worth assessing whether it would be more appropriate sub-dividing the requirement.

The team should conduct this practice together with the customer. The mockups can be drawn using any tool or a piece of paper. In this case, photos can be incorporated into the RSD. Depending on the availability of the client, the team can sketch an initial version of the mockups, taking into consideration the conceptual model. If new requirements or changes are identified, they must be registered in PB to be analyzed during the most convenient time. If it is required to make changes in the conceptual model, they should perform the guidelines described previously.

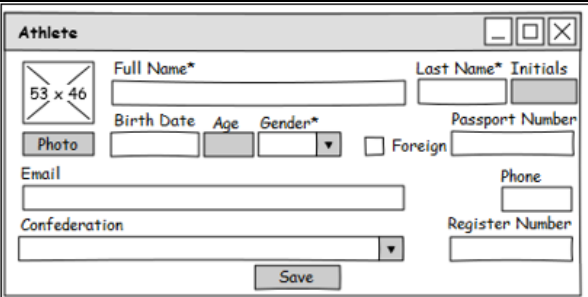
### 3.2.3 Specify the Acceptance Criteria+

This practice aims to identify or specifying the AC+ associated with a requirement. It should run in parallel with the *Modeling Mockups* activity. The team and the customer should specify the AC+ together. However, to optimize time, the team can specify some AC+ without the customer, taking into consideration the conceptual model and the knowledge gathered from other conversations with the customer. In fact, most of the AC+ provided by the customer has the business type. However, the team can extract other AC+ from the data entities. Many technical, interface, NFR, and validation rules can be reused from other similar requirements. The AC+s are stored in RAC to encourage the reuse during the specification and coding. Table 3 shows some of the AC+ related to a requirement of a doping control system [25] that is illustrated in Figure 2.

**Table 3. Acceptance Criteria+ Examples**

| ID   | Description   | Type |
|------|---|------|
| AC01 | The email address must be valid   | V    |
| AC03 | To save, it is necessary that all required fields (*) are filled  | V    |
| AC04 | Only active records must be displayed   | V    |
| AC07 | The age must be calculated from the date of birth   | V    |
| AC08 | The routine to save an athlete should also save the corresponding addresses   | T    |
| AC09 | The operation to read and write files in the file system should be done through relative address  | T    |
| AC12 | The sequential code to identify the record must be generated by the database  | T    |
| AC13 | The initials of the athlete must be extracted from the athlete's name, e.g., if the name is "Fabiana de Almeida Murer", initials must be "F.A.M". | T    |
| AC17 | All foreign athletes must have a passport number  | B    |
| AC20 | The drop-down list must only display the confederations that the user logged has access permission in your profile.                               | B    |
| AC21 | There cannot be two athletes with the same registration number in the same confederation  | B    |
| AC50 | The label must use the multilingual resource  | N    |
| AC90 | The widget is read-only   | I    |

The RSD structure (**Figure 2**) is divided into four parts. The first part (top) shows the mockups associated with the requirement, if applicable. Notice in the example that the use of the mockup allows the visualization of the data of the athletes and how they will be presented in the system, which facilitates the user validation while he/her is detailing the requirement with the development team. The second part (the left column in the table) presents widgets that are present on the mockup. The third part (center column) shows the data entities and attributes extracted from the conceptual model which relate to each widget. The widgets are also in the conceptual model related to the requirement. However, this information (left and center column) is targeted to the developer who will code the requirement.

| <b>Label:</b> Registration of athlete  | <b>Priority:</b> Critical | <b>Source:</b> Ana  | <b>Sprint:</b> 1 |
|--|---------------------------|---------------------|------------------|
| <b>Description:</b> The system should enable the inclusion and updating of data of national and foreign athletes of sports federations recognized by the International Olympic Committee |                           |                     |                  |
|   |                           |                     |                  |
| Widget   | Concepts                  | Acceptance Criteria |                  |
| Photo  | person.name               | AC09                |                  |
| Full Name*   | person.fullName           | -                   |                  |
| Last Name*   | person.lastName           | -                   |                  |
| Initials#  | athlete.initialsName      | AC13, AC90          |                  |
| Birth Date   | athlete.birthDate         | AC05, AC06          |                  |
| Age#   | -                         | AC07, AC90          |                  |
| Gender   | person.gender             | -                   |                  |
| Foreign  | person.isForeign          | AC17                |                  |
| Passport Number  | person.passportNumber     | AC17                |                  |
| Email  | person.email              | AC01                |                  |
| Phone  | person.phone              | -                   |                  |
| Confederation*   | confederation.name        | AC04, AC20          |                  |
| Register Number  | athlete.registerNumber    | AC21                |                  |
| Save   | -                         | AC03, AC08          |                  |
| -  | person.idPerson           | AC12                |                  |

**Figure 2. Example of an SRS using the RSD approach**

Finally, the fourth part (right column) shows the AC+ related to the widgets and the data entities. An AC+ may be reused for

different requirements. Reuse may also occur several times in the same SRS, for example, AC90. AC+ does not have to be associated with a data entity (e.g. AC7) nor to a widget (e.g. AC12). In general, acceptance criteria which describe NFR, web services or algorithms have no relation with widgets.

In this example, note that the AC+ could be used to detail business-related needs (AC17 and AC21), but also to describe information that is closer to what will be implemented, such as validation constraints (AC01), interface (AC90), technical (AC09) and NFR (AC50). RSD allows all these requirements to be represented in a single and integrated form which may facilitate the understanding of the developer. Although SRS are not intended to be used as a requirements validation mechanism, mockups allow validating the understanding of the needs during the identification of the AC+ with the client. AC20 is a business need requested by the client, but note that it is used the term "dropdown". RSD does not restrict the adoption of technical terms, given that the SRS is not used to validate requirements with the client. As said before, the validation in ASD methods, such as RSD, is done by frequent software deliveries. Besides, the AC+ can exemplify some rules to facilitate understanding (e.g. AC13).

If USs were used to describe this requirement, the language used would be customer-oriented and focused only on business requirements. As a complement to the US, the company could use mockups and the conceptual model, but in this case, the use of such practices would not be associated with the acceptance criteria. The integration of mockups, conceptual model and AC+ is only possible using the RSD approach. Besides, the AC+ considers other constraints beyond the business type. In RSD, the requirements are not identified and detailed by role, as happens when using US, but by business need, regardless the roles related to them. An AC+ can be related to more than one stakeholder and to more than one requirement, unlike the USs.

RSD aims to facilitate the understanding of the developer through the link made between the AC+, the mockup of widgets and the conceptual model. Besides, the adoption of AC+ allows that the internal tests performed by the team and the acceptance tests performed by the customer can be extracted directly from the RSD, without the need to prepare another artifact for this purpose.

## 4. EVALUATION

Case study (CS) is a research method that can be used when it is desired to know whether a theory applies to a specific real world setting [26]. We conducted a CS to assess how the RSD works in practice and gather insights to enhance it. We set out from the principle that finding out how to solve a problem cannot be separated from the human context. So, we took a constructivist stance, and a CS offers the opportunity to obtain a thorough understanding of how and why certain phenomena occur. We seek answers to the following Research Questions (RQ): RQ1: *What is the quality of the SRS produced using RSD?*; RQ2: *How the RSD approach affects the work of the team?*

This study was conducted over 8 months in the development of an information system for doping control [25] for a federation affiliated to the International Association of Athletics Federations.

### 4.1 Design and Procedures

We used the guidelines suggested by [27] composed of five steps: Planning; Preparation; Collecting evidence; Analyzing the data collected; and Synthesis. This study had an exploratory purpose, and its type is classified as being single-case and as embedded because there were two units of analysis: system analysts and



developers. Data were collected and analyzed simultaneously, in incremental and iterative steps. We used three data collection sources: observations, analysis of documents and interviews.

We participated as observers in the team activities, such as i) Discovery sessions with the customer to specify AC+, mockups, and the conceptual model; ii) Acceptance testing and iii) Meetings to analyze the impact of changes in requirements. Regarding the analysis of documents, the development team analyzed 39 RSD and 257 AC+. Some RSDs were related to more than one requirement. We also analyzed 157 non-conformities (NC) identified in the software by the team during acceptance tests, and 71 changes made in RSD (volatility). The interviews were conducted individually with 10 team members. A questionnaire with 38 questions was used to guide the interviews. It is available at <https://sites.google.com/site/rsdapproach/evaluation/interview>. Table 4 shows a sample of the questions. The answer of each interview was reviewed to check if it was correctly noted and to capture complementary information, if needed.

**Table 4. Excerpt of the Interview Guide**

|      |  |
|------|--|
| Q8.  | What helps or hinders you from using the approach?                                 |
| Q9.  | How do you assess the structure of RSD?  |
| Q19. | How do you assess the effort required to implement the requirements from the RSD?  |
| Q20. | How do you assess the effort required to create the RSD?                           |
| Q22. | Was the quality of the RSD different from what you expected?                       |
| Q28. | How do you assess the effort required to use the RSD compared to other approaches? |
| Q38. | How do you assess the RSD in relation to ISO\IEEE 830?                             |

The team assessed the quality of SRS from 2 aspects: i) Structure; and ii) Content. The effort required and the challenges faced when using RSD were also assessed. The quality of RSDs was evaluated by using attributes from ISO-IEEE 830 [28]: *Correct*, *Unambiguous*, *Complete*, *Consistent* and *Ranked for Importance*. For each attribute, RSD was assessed as in Compliance (1) or Non-compliance (0). We analyzed the correlation between volatility, NC and the quality of RSD by using the Spearman's correlation coefficient, also called  $\rho$  (rho). We analyzed the data using IBM SPSS® Statistics Package Software. Correlation analysis was conducted by checking the significance of the coefficients. The findings from the triangulation of the collected data are presented in the next section.

## 4.2 Results and Discussions

Scrum and several agile practices were well established in the project, such as backlog, frequent prioritizing requirements, iterations, frequent releases, continuous integration, automated build, and retrospective. Each sprint lasted for a month, but partial versions of the product were released every week. The development team had experience with ASD. The RSD has been prepared for two systems analysts with customer collaboration to specify concepts, mockups, and AC+ of the business type. The AC+ of the other types were usually specified by the analyst. Eight developers evaluated the quality of the RSD that they used during the project. Each analyst assessed the quality of the RSD produced by other analysts. The most experienced developer played the roles of architect and configuration engineer.

### 4.2.1 RQ1: What is the quality of the RSD?

To evaluate the RSD structure (Q9), the team used a scale from 1 (*Inadequate*) to 5 (*Very Adequate*). The structure is *Very Adequate* in the opinion of most respondents (80%). The remaining evaluated it as *Adequate* (4). Most interviewees (60%) pointed out that the RSD structure is more appropriate than other approaches'

structure. USs and use cases were cited by more than 70% of respondents as approaches that they had used previously. In the opinion of the team (Q22), the approach provides a SRS that met the expectations of the developers, as follows:

*“The description of the functional and system requirements through acceptance criteria leads to a developer-oriented SRS containing descriptions on how the requirement should be implemented. This helps us in the coding activity.” (D#3)*

In the evaluation in accordance with the ISO attributes (Q38), the developers considered that the RSD is considered *Modifiable*, *Traceable*, and *Correct*. And partially *Verifiable*, *Complete*, and *Unambiguous*. One respondent (analyst) considered that the RSD did not comply with the attributes: *Consistent* and *Ranked*.

All interviewees (10) also evaluated the quality of RSD compared to approaches used in other projects. Most interviewees considered that the RSD quality is better in *Correct* and *Complete* attributes. Regarding the *Complete* attribute, nine developers assessed as better, just one considered as worse. Regarding *Unambiguous*, *Modifiable* and *Consistent* attributes, half of the respondents considered that the RSD is better than other approaches used by them. The other half considered that RSD has the same quality. 8 out of 10 respondents did not consider that the RSD quality is worse than the other approaches. Only two (one analyst and one developer) considered the approach worse than other approaches in relation to the *Ranked* attribute. The quality evaluation of the RSD is summarized in Table 5.

**Table 5. Summary of the evaluation by quality attribute**

|                    | Quality of SRS |     | Non-Conformities in Software* |      | Volatility in SRS* |     |
|--------------------|----------------|-----|-------------------------------|------|--------------------|-----|
|                    | C              | NC  | C                             | NC   | C                  | NC  |
| <b>Correct</b>     | 85%            | 15% | 3,1                           | 9,3  | 1,3                | 4,5 |
| <b>Unambiguous</b> | 95%            | 5%  | 3,6                           | 12,0 | 1,8                | 3,0 |
| <b>Complete</b>    | 64%            | 36% | 4,3                           | 3,6  | 1,2                | 3,0 |
| <b>Consistent</b>  | 64%            | 36% | 3,5                           | 5,0  | 1,4                | 2,6 |
| <b>Ranked</b>      | 59%            | 41% | 3,9                           | 4,3  | 1,7                | 1,9 |

C-Compliance; NC-Non-Compliance; \*average

Table 5 shows the percentage of compliance of SRS with the quality attributes (1<sup>st</sup> column), the average quantity of NC found in the software (2<sup>nd</sup> column) and average changes made in the SRS (3<sup>rd</sup> column). For each attribute, we count the SRS evaluated as in compliance with it (X). Then, we summed all non-conformities related to the SRS (Y). The average was calculated as X/Y. For example, 33 SRS were in compliance with the *Correct* attribute (Y). The tests reported 101 non-conformities regarding the requirements related (X). Therefore, the average was  $101/33 = 3.06$  (rounded to 3.1). The same procedure was applied to calculate the average of volatility. The *Ranked* attribute had the lowest percentage of compliance (59%). The *Unambiguous* attribute had the highest percentage of compliance (95%).

The results of the statistical analysis (Table 6) showed that there are significant correlations between some quality attributes and the number of NC in the software. We investigated 21 relations between a) the quality of RSD, b) NC found in the tests and c) the changes made in RSD (volatility), as shown in Table 6.

Only five correlations were statistically significant. R12, R16, and R20 were considered significant with a 0.01 margin of error. R11 and R17 were significant with a 0.05 margin of error. Initially, we analyzed the relation between quality attributes (R1..R10), but no significant correlation was found in these attributes.

Then, we investigated the relations between the quality attributes and the NC in the software (R11..R15). During the acceptance tests, the team reported 157 NC in the software. The results showed that the *correct* SRS produced three times less NC, as shown in Table 5. R11 presented a significant correlation ( $p = -0.411$ ) between *correct* SRS and the amount of NC identified, i.e., the correct RSD had less nonconformity. The results indicated that *unambiguous* SRS produced less NC (R12), with  $p = -0.317$ . Many NC were identified in the attributes *Consistency*, *Complete* and *Ranked for Importance*, but the results showed no significant correlation between these attributes and NC.

**Table 6. Results of Spearman's correlation statistical tests**

|                                       |                 | Relation                   | p (rho)         |
|---------------------------------------|-----------------|----------------------------|-----------------|
| Between Quality Attributes            | R <sub>01</sub> | Unambiguous/Complete       | -0,174          |
|                                       | R <sub>02</sub> | Correct/Unambiguous        | 0,223           |
|                                       | R <sub>03</sub> | Consistent/Unambiguous     | 0,068           |
|                                       | R <sub>04</sub> | Consistent/Ranked          | 0,137           |
|                                       | R <sub>05</sub> | Consistent/Complete        | -0,003          |
|                                       | R <sub>06</sub> | Complete/Correct           | -0,023          |
|                                       | R <sub>07</sub> | Consistent/Correct         | 0,125           |
|                                       | R <sub>08</sub> | Correct/Ranked             | -0,211          |
|                                       | R <sub>09</sub> | Ranked/Unambiguous         | -0,194          |
|                                       | R <sub>10</sub> | Complete/Ranked            | -0,081          |
| Quality Attributes x Non-Conformities | R <sub>11</sub> | Correct/Non-Conformity     | <b>-0,411**</b> |
|                                       | R <sub>12</sub> | Unambiguous/Non-Conformity | <b>-0,317*</b>  |
|                                       | R <sub>13</sub> | Ranked/Non-Conformity      | -0,104          |
|                                       | R <sub>14</sub> | Complete/Non-Conformity    | -0,01           |
|                                       | R <sub>15</sub> | Consistent/Non-Conformity  | -0,092          |
| Volatility x Non-Conformities         | R <sub>16</sub> | Volatility/Non-Conformity  | <b>0,405*</b>   |
| Volatility x Quality Attributes       | R <sub>17</sub> | Correct/Volatility         | <b>-0,51**</b>  |
|                                       | R <sub>18</sub> | Unambiguous/Volatility     | -0,037          |
|                                       | R <sub>19</sub> | Ranked/Volatility          | -0,005          |
|                                       | R <sub>20</sub> | Complete/Volatility        | <b>-0,357*</b>  |
|                                       | R <sub>21</sub> | Consistent/Volatility      | -0,253          |

\* Correlation is significant at the 0.05 level (2-tailed)

\*\* Correlation is significant at the 0.01 level (2-tailed) p (rho)- Correlation Coefficient

We also investigated the correlation between NC and volatility (R16). The team made 71 changes in SRS. Some SRS were changed more than once. R16 ( $p = 0.405$ ) indicated that the SRS changed many times produced more NC in the software. Finally, we investigated the relation between the attributes and the volatility of SRS (R17..R21). R17 presented the highest correlation coefficient ( $p = -0.510$ ). The negative sign indicates an inverse correlation between the amount of changes (volatility) and correctness of the RSD. The correct SRSs had few changes. The incomplete SRS had many changes (R20,  $p = -0.357$ ). In such requirements, it is possible that the team did not fully understand the customer's needs. Although many changes had been made to these SRS, they were not considered *complete* and *correct*.

A weak correlation (not significant but  $p > 2$ ) was identified between the *correct* and *unambiguous* attributes. Volatility had a weak correlation negative with the consistency of the SRS. Also, between *correct* and *Ranked for Importance* attribute. The RSD structure did not represent the AC+ priority. Thus, if an AC+ had a different priority (i.e., if an AC+ had to be implemented in another sprint), the analyst registered this information as a note in Redmine, a project management tool. This was the main reason why the *Ranked* attribute which had only 59% of compliance (Table 5). Below is the statement of a developer (Q8) on the issue:

*"The priority of acceptance criteria should be described in the RSD instead of Redmine. If the developer has no attention to look at the written note in Redmine, he loses time coding AC+ allocated to another sprint."* (D#7)

It is worth to notice that the limitation stated by D#7 do not affect any other quality attribute, nor the software conformity, nor the SRS volatility because there are no correlations between them and the *Ranked for Importance* attribute. Other limitations include some inconsistencies identified between the conceptual model and mockups regarding the required fields, and SRSs considered incomplete because they were missing AC+.

#### 4.2.2 RQ2: How the RSD approach affects the work of the team?

RSD introduces new practices to improve SRS in ASD. Thus, we formulated the following hypothesis: *The effort required to specify requirements using RSD is higher than using other approaches, but the effort to implement using RSD is lower than when implementing using other approaches.* However, in the opinion of the interviewees (Q19, Q20, Q28) the effort required for using the RSD is not higher than using other approaches. One analyst considered that less effort was required. The other analyst believes that it is the same. All developers considered that RSD required either less effort (5) than other approaches or the same amount (5). According to most of the developers (7), implementing RSD requires a reasonable effort, and the other developers (3) consider that implementing from RSD requires little effort.

According to the analysts, much effort is required to keep the Traceability Matrix (TM) up-to-date. The TM presents the relationship between requirements and AC+, so it helped in the impact analysis when a request for changing requirements is received. The TM was not properly used in the project due to the high effort needed to keep it up-to-date. In parallel to drawing up the SRS, the TM had to be updated manually. The same happened when the team needed to change an AC+ and/or a requirement. The TM was operationalized by using a spreadsheet shared among the stakeholders. Developers reported that due to the TM being outdated, they did not use it to identify AC+ that could be reused in the source-code. Thus, they wasted time searching for reuse opportunities because they had to analyze the source code, instead of obtaining this information from the TM. However, since the matrix was always out-of-date, it didn't help the team. Below is the statement of a system analyst on the issue:

*"The traceability matrix is not being properly used in the project because the manual update requires much effort. I do not think it's worth wasting time on this. The matrix should be extracted automatically from the documentation."* (A#1)

Although not a practice initially defined in RSD, the AC+ were categorized by the system analysts in *General* (potential to be reused in other requirements) and *Specific*. Analysts considered that this categorization facilitated the search for it in the RAC and improved the reuse. At the end of the study, the TM was updated to examine the reuse rate of the AC+. Many AC+ were reused in several requirements, especially the AC+ of the validation, technical, and interface types, resulting in a reuse rate greater than 60% which improved the team productivity, as follows:

*"The definition of requirements through the AC+ has contributed to the reuse of rules used in other requirements, which improves our productivity."* (A#2)

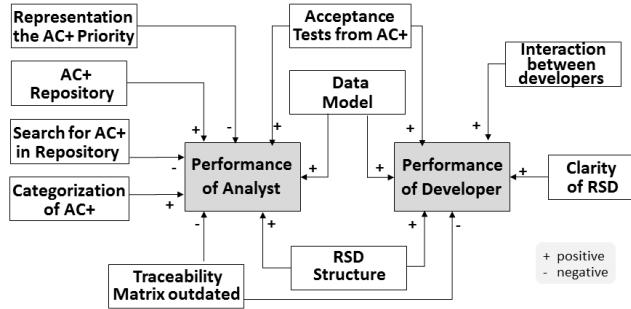
However, reuse depended largely on knowledge about the existence of AC+. We identified that there were high reuse rates of the AC+ specified by the same analyst. The reuse rate of the AC+ specified by different analysts was much lower. Analysts have reported difficulties to find AC+ that can be reused because

it was stored as a text document. A more efficient mechanism is required to share and find AC+. The support of a tool can increase reuse rates. Besides storing the AC+ in a database, the tool could provide features to locate them more efficiently.

Analysts and developers pointed out the structure of the RSD had a positive influence on team performance, as well as the objectivity and clarity of the RSD, as follows:

*“The structure and clarity of documentation were one of the aspects that most contributed to improving the productivity in the execution of my activities as a developer.” (D#5)*

**Figure 3** summarizes the good practices (+) and limitations (-) that affect the team's performance using RSD.



**Figure 3.** Factors that affect the performance using RSD

The team pointed out that the conceptual modeling was a positive factor that contributed to the construction of a more stable data model. Despite the incremental development and the changes in requirements, little changes were required in the structure of the data model during sprints, which reduced the rework.

The acceptance tests were conducted using the RSD as a reference. This was pointed out as a good practice because it was not necessary to create a specific artifact for testing (Q8). The tester validates whether the software complies with each AC+. We identified that the acceptance test could be optimized by tool to generate a roadmap (checklist) automatically from AC+. Therefore, the tester needs only to check or uncheck the checklist, according to the test result. In addition to making the registration of NC associating the AC+ related, the tool can send an email notification to the responsible to make the necessary corrections. Software will optimize the time of the team. Furthermore, the use of the checklist prevents the tester to forget to validate an AC+, thus contributing to improving the quality of testing.

### 4.3 Threats to Validity

The quality of the data extraction was a potential threat to the validity. To mitigate it, data were triangulated (interviews, observations, and analysis of documents) and two researchers (first author and project coordinator) checked the results. This was the main strategy for increasing credibility.

Another threat was the software engineers give answers that they thought the researchers wanted to hear, rather than responding to real opinion of them. To minimize this threat, the interviewees were encouraged to critique and point out the difficulties and limitations of the approach in order to improve it and the work of the team. The researchers had no personal or professional relationship with the software engineers. The confidentiality of the answers was guaranteed, and it was reinforced with the interviewee the importance to detail the answers as much as

possible. Also, leading questions were avoided, and probes were defined with the objective of deepening the respondent's answers.

The subjectivity inherent in categorizing and classifying the factors that affected the quality of SRS was tackled by undertaking analysis based on the team's perception. The concepts of quality defined in ISO-IEEE 830 were used to minimize the impact of subjectivity.

The approach could have been evaluated by other researchers or by using another research method. These may have reduced any bias in the results that may have resulted from the authors conducting the evaluation. The objective of the researchers, who themselves conducted the case study, was to identify in loco the limitations and the difficulties that the team had in using the approach, and also to identify best practices and strengths. Certainly, the level of detail captured might be different from an evaluation carried out by others.

## 5. RELATED WORK

Some studies were identified that propose practices to improve SRS in agile projects. Batool [29] proposed a conceptual scrum framework based on user stories to describe functional requirements. Class diagrams and mockups are used but they are not integrated with user stories. Also, the SRS lacks NFR, technical aspects and traceability of the requirements. The framework also adopts other artifacts such as story cards, index cards, and vision document which affect the agility of the process.

Gebhart [30] argues that scenarios are an appropriate way to describe a system from the user's point of view and presents an enhancement based it. The methodology establishes activities for the identification of stakeholders and goals. The goals are realized through scenarios that can be reused, are customer facing, free of implementation and do not contain architectural decisions.

SnapMind provides a language based on mind maps to represent both US and domain models for ASD [31]. It aims to make the requirements modeling process more user-centered, but it does not support technical constraints, NFR, mockups and AC.

## 6. CONCLUSIONS

Current techniques used to specify requirement in ASD are customer-oriented and have been proven to be insufficient to developers. We presented the RSD approach in order to overcome this issue by the inclusion of design practices like conceptual modeling, mockups and AC+, in an integrated manner. A case study was conducted to assess how RSD works in practice, and to gather the difficulties faced by the team when using it.

The results showed that RSD met the developer's expectations and proved to be a very objective SRS, suitable for coding activities. The practices introduced did not adversely affect the process agility. The results support that RSD has the potential to reduce the gap between the problem and solution domains, thereby enabling the developer to acquire a better understanding of the feature to be implemented. Also, RSD allows technical aspects to be represented and produces an SRS that is closer to what will be implemented. The feedback collected through interviews suggests that RSD does not add extra effort (as suggested by 50% of the respondents) or may even help reducing the effort involved in coding, testing and maintenance (as suggested by the remaining 50% of the respondents).

To facilitate the adoption of RSD in practice, we are working on the development of a tool that does the automatic extraction of widgets from mockups, and the entities and fields from the

conceptual model. The tool also provides mechanisms to describe the AC+ considering the priority and to search the AC+ in the repository, encouraging reuse. In the future, the tool will also generate the traceability of the requirements and AC+, and the skeleton (source-code) of the business classes from the conceptual model. In addition, the tool will support the tests from the AC+ and the quality evaluation of AC+ by developers.

Also, we intend to conduct assessments in the context of other agile projects to identify the points of convergence and divergence regarding this empirical study and enhancing the RSD approach. The objective of the case study was evaluating the use of the RSD in practice and identifying its strengths and limitations. We did not have the intention of making a comparative assessment of the RSD with other approaches, although a few questions were asked about it. An experiment can be conducted to evaluate the RSD quantitatively in comparison to other approaches.

## 7. REFERENCES

- [1] Agile Manifesto. *Manifesto for Agile Software Development*. Available: <http://www.agilemanifesto.org/>. 2001.
- [2] VersionOne. 9TH Annual State of Agile Survey. Available: <http://info.versionone.com/state-of-agile-development-survey-ninth.html>. 2015.
- [3] Heikkilä, V. T., Damian, D., Lassenius, C., and Paasivaara, M. *A Mapping Study on Requirements Engineering in Agile Software Development*, 41st Euromicro, Funchal. 2015.
- [4] Read, A. and Briggs, R.O. *The Many Lives of an Agile Story: Design Processes, Design Products, and Understandings in a Large-Scale Agile Development Project*. 45th Hawaii International Conference, pp.5319,5328. 2012.
- [5] Daneva, M., Van Der Veen, E., Amrit, C., Ghaisas, S., Sikkil, K., Kumar, R., Ajmeri, N., Wieringa, R. *Agile requirements prioritization in large-scale outsourced system projects: An empirical study*. J. Syst. Soft. 86, 5. 2013.
- [6] Medeiros, J., Goulão, M., Vasconcelos, A., and Silva, C. *Towards a model about quality of software requirements specification in agile projects*. 10th QUATIC. Lisbon, Portugal. 2016.
- [7] Povilaitis, S. Acceptance Criteria. Available: <http://www.leadingagile.com/2014/09/acceptance-criteria/>. 2014.
- [8] Heck, P. and Zaidman, A. A quality framework for agile requirements: a practitioner's perspective. <http://arxiv.org/abs/1406.4692>. 2014.
- [9] Bjarnason, E., Wnuk, K. and Regnell, B. *A case study on benefits and side-effects of agile practices in large-scale requirements engineering*. 1st AREW. ACM, USA. 2011.
- [10] Lee, J.C., Judge, T.K., McCrickard, D.S. *Evaluating eXtreme scenario-based design in a distributed agile team*. CHI EA. New York, USA, 863-877. 2011.
- [11] Haugset, B., Stalhane, T. *Automated Acceptance Testing as an Agile Requirements Engineering Practice*. 45th HICSS, Maui, HI, pp. 5289-5298. 2012.
- [12] Hirschheim, R., Klein, K., and Lyytinen, Kalle. *Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations*. Cambridge Univ., USA. 1995.
- [13] Loucopoulos, P., Zicari, R. *Conceptual Modeling, Databases and CASE: An Integrated View of Information System Development*. John Wiley & Sons, New York, USA. 1992.
- [14] Olivé, A. *Conceptual Modeling of Information Systems*, Springer Verlag, ISBN 978-3-540-39389-7. 2007.
- [15] Schwaber, K. and Beedle, M. *Agile Software Development with Scrum*. Prentice Hall PTR, NJ, USA. 2001.
- [16] Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston, MA, USA. 1999.
- [17] Agile Alliance. *Practices Map*. Available at: <http://guide.agilealliance.org/subway.html>. 2016.
- [18] Cohn, M. *User Stories Applied: For Agile Software Development*. Addison Wesley, Redwood, CA, USA. 2004.
- [19] Ricca, F., Scanniello, G., Torchiano, M., Reggio, G., and Astesiano, E. *Assessing the Effect of Screen Mockups on the Comprehension of Functional Requirements*. ACM Trans. Softw. Eng. Methodol. 24, 1. 2014.
- [20] Rivero, J.M., Grigera, J., Rossi, G., Luna, E. R., Montero, F., Gaedke, M. *Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering*, IST, 56,6. 2014.
- [21] Ferreira, J., Noble, J. and Biddle, R. *Agile Development Iterations and UI Design*, Agile Conference (AGILE), Washington, DC, pp. 50-58. 2007.
- [22] Whichard, G. Definition of Done vs. Acceptance Criteria. <http://www.governmentciomagazine.com/2014/08/definition-done-vs-acceptance-criteria>. 2014.
- [23] Mamoli, S. On Acceptance Criteria for User Stories. Available: [http://nomad8.com/acceptance\\_criteria/](http://nomad8.com/acceptance_criteria/). 2010.
- [24] Medeiros, J., Alves, D., Wanderley, E., Vasconcelos, A. and Silva, C. *Requirements Engineering in Agile Projects: A Systematic Mapping based in Evidences of Industry*. ESELAW, CIBSE. Peru. pp.460-476. 2015.
- [25] Medeiros, F., Medeiros, J., Ayres, F., Viana, C., Rocha, J., Viegas, V., Mendes, E., Santos, A. *An Information System to Support the Anti-doping Process*. Information Science and Applications (ICISA). Springer Singapore. 2016.
- [26] Wohlin, C., Höst, M., and Henningsson, K. *Empirical Research Methods in Software Engineering*. Lecture notes in Computer Science, 2765 7-23. 2003.
- [27] Runeson, P. and Martin, H. *Guidelines for conducting and reporting case study research in software engineering*. Empirical Software. Eng. 14, 2, 131-164. 2009.
- [28] ISO/IEEE 830-1998. Recommended Practice for Software Requirements Specifications, IEEE. 1998.
- [29] Batool, A., Hafeez, Y., Asghar, S., Abbas, M.A., Hassan, M.S.. *A Scrum Framework for Requirement Engineering Practices*. ISSN: 2306-1448 (online). 2013.
- [30] Gebhart, M., Giessler, P., Burkhardt, P., Abeck, S. *Quality-Oriented Requirements Engineering for Agile Development of RESTful Participation Service*. ICSEA. 2014.
- [31] F. Wanderley, A. Silva, J. Araujo and D. S. Silveira. *SnapMind: A framework to support consistency and validation of model-based requirements in agile development*, IEEE 4th MoDRE, Karlskrona, Sweden. 2014.