JULIANA DANTAS RIBEIRO VIANA DE MEDEIROS

AN APPROACH TO SUPPORT THE REQUIREMENTS SPECIFICATION IN AGILE SOFTWARE DEVELOPMENT

RECIFE
2017

Juliana Dantas Ribeiro Viana de Medeiros

An Approach to Support the Requirements Specification in Agile Software Development

Thesis presented to the Post-Graduate Program in Computer Science of the Federal University of Pernambuco as partial fulfillment of the requirements for the PhD Degree in Computer Science.

Advisor: *Alexandre Marcos Lins de Vasconcelos*

Co-Advisor: *Carla Taciana Lima Lourenco Silva Schuenemann*

Co-Advisor: *Miguel Carlos Pacheco Afonso Goulão*

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

M488a Medeiros, Juliana Dantas Ribeiro Viana de
An approach to support the requirements specification in agile software development / Juliana Dantas Ribeiro Viana de Medeiros. – 2017.
180 f.:il., fig., tab.

Orientador: Alexandre Marcos Lins de Vasconcelos.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências e apêndices.

1. Engenharia de software. 2. Desenvolvimento de software. 3. Requisitos de software. I. Vasconcelos, Alexandre Marcos Lins de (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2017-149

Juliana Dantas Ribeiro Viana de Medeiros

An approach to support the Requirements Specification in Agile Software Development

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutora em Ciência da Computação

Aprovado em: 13/03/2017.

To GOD

Acknowledgments

I thank God for giving me health and disposition to carry out this work.

Special thanks to my husband, José Medeiros, who has provided the most appropriate environment for my professional growth.

To my children, Vitor, Gabriela, and Lucas, who had grown up living with my absence when I needed to work and study.

To my parents and brothers, who have always been the resting place in times of difficulty.

To Alexandre Vasconcelos and Carla Silva for the availability and guidance on how to best conduct this work.

To Miguel Goulão and the BE Mundus program, for the opportunity to develop part of the research at the New University of Lisbon, Faculty of Sciences and Technology (FCT) in Portugal. And, to the new friends I made at the FCT who welcomed me with great affection during the study period.

Thanks also for the availability of all participants of our research.

The Federal Institute of Education, Science and Technology of Paraíba (IFPB), the teachers and co-workers, who contributed to carrying out this research.

To my friends, who encouraged me throughout this project.

Abstract

Although Agile Software Development (ASD) has grown in recent years, research evidence points out several limitations concerning its requirements engineering activities. It was observed that an inadequate specification acts as a catalyst to others problems, such as low productivity of the team and difficulty in maintaining software. Improving the quality of Software Requirements Specifications (SRS) may help gaining a competitive advantage in the software industry. The goal of this study is to investigate the phenomenon of the requirements specification activity in ASD, discuss relevant findings of this phenomenon to industrial practice, and propose practices to write a SRS targeted to development team. First, a Systematic Mapping (SM) study was conducted to characterize the landscape of requirements engineering in ASD. The thematic synthesis method was used to code and synthesize the data collected from the primary studies selected. After that, some of the challenges pointed out in the SM were investigated in more depth in six industrial case studies. Data collected from documents, observations, and interviews with software engineers were triangulated, analyzed, and synthesized using techniques of grounded theory and meta-ethnography. The analysis and cross-synthesis of the case studies resulted in a model that defines the simplicity and objectivity as essential quality factors of SRS in ASD. The main factors that affect the quality are related to the customer-driven nature that tends to leave the prolix SRS, hindering the understanding of the software engineers, as they are, at the same time, insufficient to support coding, testing and maintenance tasks. One approach was proposed to provide a SRS closer to the development needs, addressing some of the quality factors of the model. Empirical studies that evaluated the approach show that the design practices used in the proposed approach have the potential to reduce the gap between the problem and the solution domains, producing an objective SRS that is team-driven and closer to that will be implemented.

Keywords: Agile Software Development. Software Requirements Specification. Empirical Software Engineering.

Resumo

Embora o Desenvolvimento Ágil de Software (DAS) tenha crescido nos últimos anos, estudos empíricos apontam vários problemas relacionados com as atividades de engenharia de requisitos. Observou-se que a especificação inadequada age como um catalizador para outros problemas, como por exemplo, baixa produtividade da equipe e dificuldades na manutenção do software. Melhorar a qualidade da Especificação de Requisitos de Software (ERS) pode ajudar a ganhar uma vantagem competitiva na indústria de software. O objetivo deste estudo é investigar o fenômeno da especificação de requisitos no DAS, discutir relevantes implicações desse fenômeno para a indústria, e propor práticas para escrever ERS voltadas para a equipe de desenvolvimento. Primeiro, um Mapeamento Sistemático (MS) foi realizado para caracterizar o panorama da engenharia de requisitos no DAS. O método de síntese temática foi utilizado para codificar e sintetizar os dados coletados a partir dos estudos primários selecionados. Em seguida, alguns dos desafios apontados no MS foram investigados com mais profundidade em seis estudos de caso industriais. Os dados coletados a partir de documentos, observações e entrevistas com engenheiros de software foram triangulados, analisados e sintetizados usando técnicas de teoria fundamentada e meta-etnografia. A análise e síntese cruzada dos estudos de caso resultaram em um modelo de qualidade que define a simplicidade e objetividade como fatores essenciais na ERS no DAS. Os principais fatores que afetam a qualidade estão relacionados à natureza orientada para o cliente que tende a deixar a ERS prolixa, dificultando a compreensão do engenheiro de software, ao mesmo tempo que é insuficiente para a codificação, testes e manutenção. Uma abordagem foi proposta para fornecer uma especificação de requisitos mais próxima das necessidades de desenvolvimento, atendendo alguns dos fatores de qualidade do modelo. Os estudos empíricos que avaliaram a abordagem demonstram que as práticas de design utilizadas pela abordagem tem o potencial de reduzir a distância entre o domínio do problema e o da solução, produzindo uma ERS objetiva, voltada para o desenvolvedor, e próxima do que vai ser implementado.

Palavras-chave: Desenvolvimento Ágil de Software. Especificação de Requisitos de Software. Engenharia de Software Empírica.

List of Figures

List of Tables

List of Acronyms

AC	Acceptance Criteria
AC+	Acceptance Criteria Plus (proposed by RSD approach)
API	Application Programming Interface
ASD	Agile Software Development
AT	Acceptance Testing
ATDD	Acceptance Test Driven Development
BDD	Behavior Driven Development
CR	Change Request
EFT	Electronic Funds Transfer
MDD	Model-Driven Development
NFR	Non-Functional Requirements
NC	Non-Conformities
PO	Product Owner
PS	Primary Studies
RAC	Repository of Acceptance Criteria
RE	Requirements Engineering
RQ	Research Question
RSD	Requirements Specification for Development
RUP	Rational Unified Process
SM	Systematic Mapping
SRS	Software Requirements Specification

SRQ	Specific Research Questions
TM	Traceability Matrix
TDD	Test-Driven Development
UFPE	Universidade Federal de Pernambuco
US	User Stories
XP	Extreme Programming

Contents

1 INTRODUCTION

This introductory chapter discusses some aspects that characterize and justify the research conducted in this thesis. Firstly, Section 1.1 presents a motivation for this research, highlighting some problems related to the Requirements Engineering (RE) in Agile Software Development (ASD). The statement of the goals, sub-goals and research questions are presented in Section 1.2. Section 1.3 describes the methodological classification of the research and the steps used to achieve the objectives of the thesis. A summary of the thesis structure is presented in Section 1.4.

1.1 Motivation

According to Thayer and Dorfman (1997), Requirements Engineering (RE) provides the appropriate mechanism to understand what the customer wants, analyzing the needs, verifying the feasibility, negotiating solutions, specifying and managing their changes.

The quality of the Software Requirements Specification (SRS) has been recognized as an important condition to gain a competitive advantage in the software industry (SAITO et al., 2013). SRS is a structured collection of software requirements (functions, performance, design constraints, and attributes) and its external interfaces (ISO-IEEE 830, 1998).

The adoption of agile methods to develop software has emerged as an alternative to traditional development. Agile methods treat the RE very differently from traditional methods (HEIKKILA et al., 2015). An Agile Software Development (ASD) begins with only a general overview of the problem without further details. Requirements understanding is done throughout the project, in an iterative and incremental way based on customer feedback. This suggests that the notion of quality of SRS in ASD is different from the notion of quality in the traditional development (HECK and ZAIDMAN, 2014). Despite the importance of RE in the success of software development, RE is seen in agile methods as bureaucratic, that makes the process less agile (PAETSCH et al., 2003). One of the values defined in the Agile Manifesto (2001) supports "*working software over comprehensive documentation*".

A survey conducted by VersionOne (2015), involving about 4000 people, shows that 45% of respondents use agile methods in most projects. However, empirical studies conducted in the industry point out several problems related to requirements engineering in ASD (HEIKKILA et al., 2015; DANEVA et al., 2013; READ and BRIGGS, 2012), such as low availability of the customer, poor quality of SRS, inadequate management and prioritization of the requirements, among others, as shown in Figure 1.1.

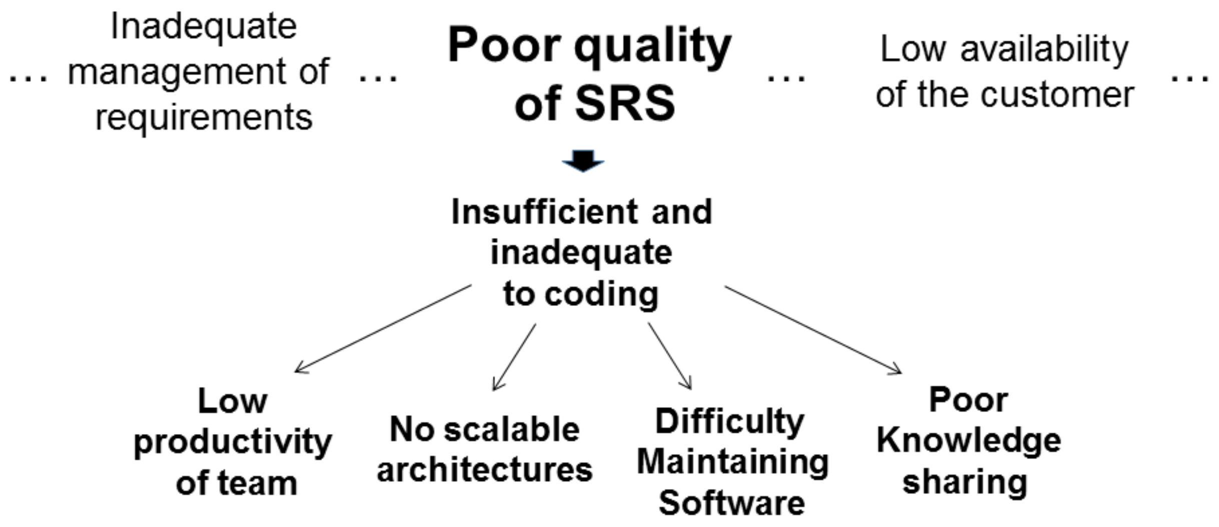

... Inadequate management of requirements ...
Poor quality of SRS
... Low availability of the customer ...
Insufficient and inadequate to coding
Low productivity of team
No scalable architectures
Difficulty Maintaining Software
Poor Knowledge sharing

Figure 1.1 Some problems related to the requirements engineering in ASD

The scope of this research focus on investigating and proposing contributions to the problems related to the quality of SRS in the context of the ASD. Although the Agile Manifesto (2001) recommends validating requirements through frequent software releases, in current agile approaches, the SRS continues to be written in a language intended for the customer, instead of being directed to the software engineers. Besides compromising team productivity, an inadequate requirements specification acts as a catalyst for others problems, often overshadowed, but that have a very large impact on the failure of projects, as shown below:

- **No scalable architecture**: in ASD the understanding of customer needs is obtained incrementally, from frequent deliveries and validations with the user. However, the design activity is neglected in the requirements specification (HECK and ZAIDMAN, 2016), producing a complex and fragmented architecture that hinders its scalability (RUDOFER et al., 2012; BJARNASON et al., 2012);

- **Difficulty in maintaining software**: one of the agile values is that teams respond quickly to changes (AGILE MANIFESTO, 2001), however some studies point challenges related to the software maintenance in ASD (BATOOL et al., 2013; HAUGSET and STALHANE, 2012). In fact, some changes are easily incorporated, such as changing the position of a field on the screen, removing a validation rule or creating a new report. However, there are client requests that imply structural changes. In such cases, the response to requests is not so fast, especially if the documentation is inadequate. Sometimes, it is not worth to refactor the existing code. It is better to implement all the code again;
- **Difficulty in knowledge sharing when turnover**: software development companies have one of the largest professional turnover rates (TECHREPUBLIC, 2015). The departure of some staff person can lead to loose important knowledge if information such as business rules, data model, and architecture are only in the minds of these people. The entry of new people in the team requires that some people interrupt their activities to pass on knowledge to the new members.

Recent studies show that the SRS in ASD is considered superficial, insufficient and inadequate to be implemented (HEIKKILA et al., 2015; HECK and ZAIDMAN, 2016). User stories (US) are a widely adopted requirements notation in agile software development. Yet, user stories are too often poorly written in practice and exhibit inherent quality defects (LUCASSEN et al., 2016). Developers consider that the SRS based on user stories are brief, vague, ambiguous and insufficient for capturing the complexities of the up-front design (ABDULLAH et al., 2011; READ and BRIGGS, 2012). 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. User stories lack the power of expression to describe design requirements (HEIKKILA et al., 2015).

The existing gap between the specifications of the customer's needs and the details required to produce the solution is a challenge that jeopardizes the development process in agile projects (RUDORFER et al., 2012). Agile projects are those guided by the values and principles of the agile manifesto (AGILE MANIFESTO, 2001).

Only simple, customer visible functional requirements can be described as basic user stories. When the software system is large and complex or hardware-dependent, user stories do not convey enough information for software design, and separate system and subsystem requirements are required (HEIKKILA et al., 2015). The focus on functional requirements often leads to overlooking design information such as technical and validation constraints, making their development harder at later stage (HAUGSET and STALHANE, 2014). 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.

Story cards are extensions of user stories and it is also being used to specify requirements in agile projects. A story card is like an iceberg: what you see is only a small part of what you will get (NAWROCKI et al., 2002). Sharp et al. (2009) consider that story cards have an incomplete and partial view of the requirements, which hampers the development from them. Use cases and scenarios (JACOBSON et al., 1992) are used less frequently in agile projects. However, they also have limitations to describe technical aspects and are difficult to interpret (THOMSON et al., 2008). Use cases permit a complete cataloguing of user tasks for different classes of users but say little or nothing about how these tasks are presented to the user or how they should be prioritized in the interface (LOSADA et al., 2013).

The research about the requirements specification activity has been developed for more than 20 years and has produced important knowledge. However, the vast majority of the previous studies were not conducted in the context of ASD. Lucia and Abdallah (2010) discusses some problems related to the requirements activities in ASD. However, the assumptions are not based on empirical studies. Heck and Zaidman (2014) and Lucassen et al. (2016) describe some criteria that should be considered in the SRS in ASD, but their goal was only to evaluate the final quality of SRS. Results of these studies lack explanatory power because they do not analyze the factors that affect the quality of SRS and neither the relationship between them.

1.2 Goals and Research Questions

Considering the challenges related to the phenomenon of the requirements specification activity in ASD, the goal of this thesis is

to propose practices to provide a SRS with information targeted to the development team supporting the agile software development.

The sub-goals are as follows:

- Investigating in the literature how the requirements engineering has been conducted in projects that adopt agile methods;
- Investigating the phenomenon in different agile contexts in the industry;
- Building an explanatory model from the industry investigation providing a deeper description about the factors that should be considered to write more useful SRS in agile projects;
- From the results of the investigations, proposing practices to provide a SRS containing design information targeted to development team in ASD;
- Evaluating how the proposed practices work in real projects.

The Research Questions (RQ) that guide this study are the following:

- *RQ1: How the requirements engineering has been conducted in projects that adopt agile methods?*
- *RQ2: What factors affect the quality of SRS in ASD and how these factors affect the work of the software engineers?*
- *RQ3: How to produce an SRS with a single integrated view of the requirements and design information directed to the development team in ASD?*

1.3 Research Methodology

The general purpose of this research is to pursue answers for questions that are, in its essence, exploratory. Exploratory questions are designed to gain deeper knowledge about some phenomenon, and discuss useful issues that help to clarify the understanding of that phenomenon (EASTERBROOK et al., 2008). The phenomenon in question is the requirements specification activity in ASD.

The philosophical stance chosen for the study affects the methods that should be used to answer the research questions and what can be accepted as truth. In this research, the constructivist stance was chosen which *"concentrates less on verifying theories, and more on understanding how different people make sense of the world, and how they assign meaning to actions"* (EASTERBROOK et al., 2008). In such studies, scientific knowledge is attached to the context from where it was created. Constructivists prefer methods that collect rich qualitative data about human activities.

A mixed method research strategy was used in order to investigate the phenomenon in the literature and industrial practice and build a rich description about it. First, a systematic mapping (SM) was conducted, and then a cross-case analysis was performed from data collected from six software organizations. Case studies were chosen as the research method because they offer the opportunity to obtain a thorough understanding of how and why the phenomena occur in practice. We set out from the principle that finding out how to solve a problem cannot be separated from the human context. Accordingly, we use a qualitative approach.

Qualitative research methods, such as case studies, offer rich data, and tend to be well-suited for our purposes. They help researchers building tentative theories (EASTERBROOK et al., 2008; YIN, 2009; MERRIAM, 2009). According to Marconi and Lakatos (2008), qualitative research is concerned to analyze and interpret the deeper aspects, describing the complexity of human behavior by providing more detailed analysis of the investigations, habits, attitudes and behavioral tendencies. Qualitative researchers study phenomena in their natural settings, attempting to make sense of, or interpret, phenomena regarding the meanings people bring to them (MERRIAN, 2009).

To carry out the research, we opted to an inductive approach. An inductive approach is a mental process that departs from particular data, sufficiently observed, and infers a general truth or universal, not contained in the examined parts (MARCONI and LAKATOS, 2003). In the inductive process, the researchers gather data to build concepts, hypotheses, or theories rather than deductively testing hypotheses as in positivist stance. The induction is performed in three steps (MARCONI and LAKATOS, 2003):

- **Observation of the phenomenon** - observe and analyze, in order to discover the causes of their manifestation;
- **Discovery of the relationship between facts** - compare and collate the facts to discover the constant relationship between them;
- **Generalization of the relationship** - generalizing the relationship found between similar facts and phenomena.

This research is broadly based on empirical software engineering methods and guidelines. Empirical Software Engineering is a research paradigm that makes use of well-proven research methods to plan and carry out investigations, enhancing their scientific nature. Empirical research explores, describes, predicts, and explains natural, social, or cognitive phenomena by using scientific methods and evidence-based experience (SJOBERG et al., 2007). Evidence is any observable event that tends to establish or disprove a fact (KITCHENHAM et al., 2005). This research was conducted following the steps as shown in Figure 1.2

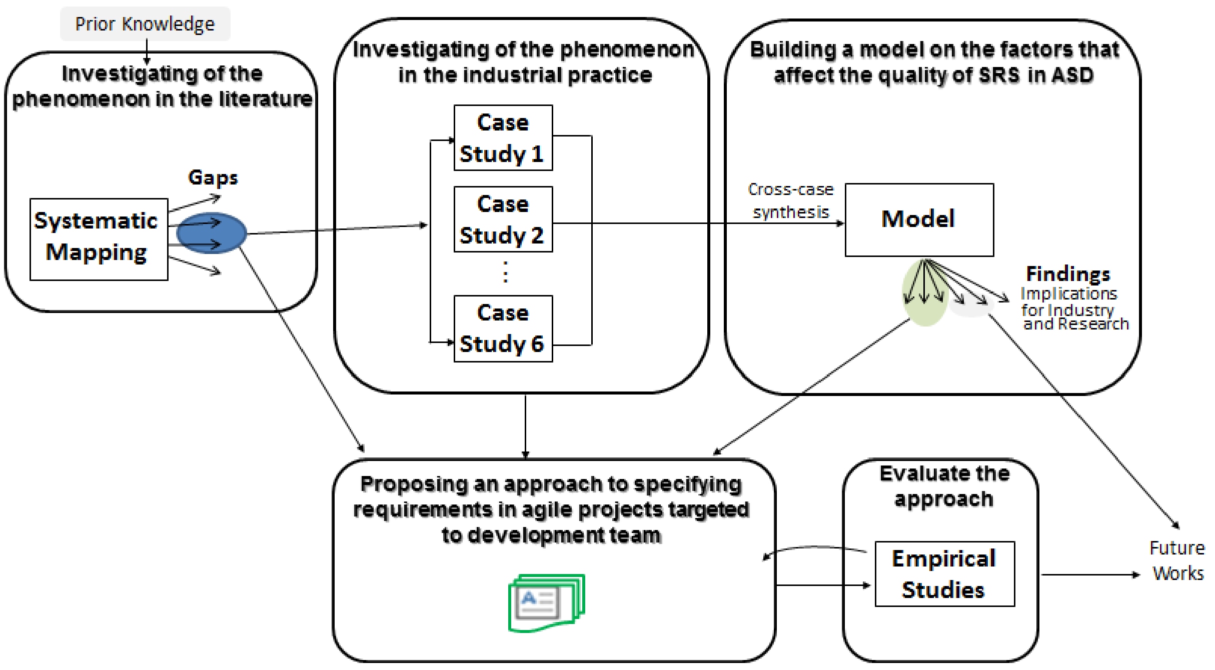

Prior Knowledge
Investigating of the phenomenon in the literature
Gaps
Systematic Mapping
Investigating of the phenomenon in the industrial practice
Case Study 1
Case Study 2
Case Study 6
Building a model on the factors that affect the quality of SRS in ASD
Cross-case synthesis
Model
Findings
Implications for Industry and Research
Proposing an approach to specifying requirements in agile projects targeted to development team
Evaluate the approach
Empirical Studies
Future Works

Figure 1.2 Research Steps

- **Step 1- Investigating of the phenomenon in the literature**: first, a Systematic Mapping (SM) was conducted to investigate the requirements engineering in ASD. Primary studies were selected, analyzed and synthesized to provide a comprehensive and updated view of the state of the art on this subject, as well as to identify research gaps;

- **Step 2- Investigating of the phenomenon in the industrial practice**: the results of the SM pointed out gaps related to the management, verification, validation and specification of the requirements. This step aimed at investigating in practice the gaps related to the requirements specification activity in ASD. So, six industrial case studies were conducted to understand and explain the phenomenon;
- **Step 3- Building a model:** A model was built from the cross-case synthesis of the six companies, providing a deeper understanding of the quality of SRS in ASD, and pointing out some implications for the industry and research;
- **Step 4 - Proposing an approach**: based on the findings of investigations carried out in the literature and industry, we developed an approach that is a set of practices to specify requirements in agile projects targeted to development team. The approach was named as RSD (Requirements Specification for Developer) and it is the main contribution of this research;
- **Step 5: Evaluation**: we conducted two empirical studies to evaluate in practice the use of the proposed approach. Two case studies investigated the perception of the software engineers concerning the content and structure of RSD, the effort required to specify and implement using the RSD approach and the difficulties found.

Table 1.1 shows in which steps the research questions (defined in Section 1.1) will be addressed.

Table 1.1 – Research Questions x Research Steps

	Steps				
Research Question	1	2	3	4	5
RQ1	X				
RQ2	X	X	X		
RQ3				X	X

The thematic synthesis (CRUZES and DYBA, 2011) method was used to analyze and synthesize the data collected from the primary studies in the systematic mapping. For the investigation in the industry, the interviews were transcribed, and then techniques of ground theory (SJØBERG et al., 2008) proposed by the constant comparison method (SEAMAN, 2008) was used to code, categorize and synthesize the data in each study. Meta-ethnography procedures were used to translate concepts and propositions during the cross-case synthesis (NOBLIT and HARE, 1988). Each step is described in a specific chapter of this thesis. The methods and procedures used in each step are detailed in the respective chapters. Table 1.2 shows the methodological classification of this research.

Table 1.2 – Methodological Classification of the Research

Questions Type	Exploratory
Philosophical Stance	Constructivist
Data analysis	Qualitative
Approach	Inductive
Research Method	Systematic Mapping and Case Studies
Synthesis Method	Thematic Synthesis, Grounded Theory, Meta-ethnography

1.4 The Structure of the Thesis

We organize the remainder of this work as follows:

- **Chapter 2** reviews essential concepts used throughout this research;
- **Chapter 3** presents the summary of the protocol, procedures and results of the systematic mapping that investigated in the literature how the requirements engineering has been conducted in ASD;
- **Chapter 4** carefully explains the six industrial case studies performed in order to investigate in practice the phenomenon of the requirements specification activity in ASD. The research method, data collection and analysis procedures are described, as well as the context and the findings of each case study;
- **Chapter 5** reports the steps to conduct the cross-case analysis and synthesis, as well as the resulting model from this analysis. The model is compared against similar and conflicting evidence from the studies available in the literature, enfolding the experience provided by them;
- **Chapter 6** carefully describes the metamodel and practices of the approach proposed to specify requirements in ASD. Also, compares the approach with some related works;
- **Chapter 7** describes the research design, procedures, results and threats to validity of two empirical studies performed to evaluate the proposed approach;
- **Chapter 8** presents the concluding remarks, reviews the contributions of this research, details future works and enumerates suggestions for new researches.

2
THEORETICAL BACKGROUND

This chapter presents the theoretical background on which this research is based. The quality of Software Requirements Specification (SRS) is discussed in Section 2.1. Section 2.2 presents the essential concepts about Agile Software Development (ASD). Finally, Section 2.3 discusses some design practices.

2.1 Software Requirements Specification Quality

One of the most cited reasons for failure in software projects is poor requirements gathering techniques (KASSAB, 2015). According to SAITO (2013), the successful development of software depends on the quality of SRS. The SRS may contribute to the sharing of knowledge among stakeholders and in distributed projects. Also, it can reduce the loss of knowledge when a stakeholder becomes unavailable, thereby facilitating software maintenance in the future. In large teams, it might be better to have an appropriate SRS instead of explaining the same thing many times to different people (PAETSCH, 2003).

The quality of SRS is a widely discussed concept in the literature. An exploratory study of various models for SRS quality evaluation is presented in (SAAVEDRA, 2013), which analyzes in detail the nine most referenced models in the literature. All these nine models are based on the standard ISO-IEEE 830 (1998) that is the standard body of work on this subject, defining a set of eight quality characteristics for a good SRS:

- **Correct**: A SRS is correct if, and only if, every requirement stated therein is one that the software shall meet;

- **Unambiguous**: A SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation;
- **Complete**: The SRS needs no further amplification because it is measurable and sufficiently describes the capability and characteristics to meet the stakeholder’s need;
- **Consistent**: It refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct;
- **Ranked for importance and stability**: A SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement;
- **Verifiable**: A SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement;
- **Modifiable**: A SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style;
- **Traceable**: A SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.

In 2011, the ISO-IEEE 830 (1998) was replaced by the standard ISO-IEEE 29148 (2011), which introduces feasibility, necessity, free of implementation, and singularity as new characteristics for requirements while removing *prioritization (ranked for importance)*, *correctness*, and *modifiability*:

- **Feasible**: The requirement is technically achievable, does not require major technology advances, and fits within system constraints (e.g., cost, schedule, technical, legal, regulatory) with acceptable risk;
- **Necessary**: The requirement defines an essential capability, characteristic, constraint, and/or quality factor. If it is removed or deleted, a deficiency will exist, which cannot be fulfilled by other capabilities of the product or process;

- **Implementation Free**: Avoid placing unnecessary constraints on the architectural design. The objective is to be implementation-independent. The SRS states what is required, not how the requirement should be met;
- **Singular**: The requirement statement includes only one requirement with no use of conjunctions.

Furthermore, the new standard distinguishes between *individual* and a *set of requirements*. A set of requirements shall be *complete*, *consistent*, *affordable*, and *bounded*, as follows. These can be fulfilled by satisfying the individual ones.

- **Complete**: The set of requirements needs no further amplification because it contains everything pertinent to the definition of the system or system element being specified;
- **Consistent**: The set of requirements does not have individual requirements which are contradictory. Requirements are not duplicated. The same term is used for the same item in all requirements;
- **Affordable**: The complete set of requirements can be satisfied by a solution that is obtainable/feasible within life cycle constraints (e.g., cost, schedule, technical, legal, regulatory);
- **Bounded**: The set of requirements maintains the identified scope for the intended solution without increasing beyond what is needed to satisfy user needs.

According to Lucassen (2016), unfortunately, most requirements specifications are unable to adhere to ISO-IEEE 29148 (2011) in practice, although evidence shows a correlation between high-quality requirements and project success. These quality attributes of ISO-IEEE are widely used in the traditional development. The novelty of our study is to investigate how these attributes work in ASD and identify what other factors affect the quality of SRS in ASD.

The characteristics whose absence or presence denote quality are completely dependent upon the situation surrounding each product (BROWN, 1997). In essence, quality is relative. Our research investigated the quality concept of SRS in the context of the ASD.

2.2 Agile Software Development

In 2001, the publication of the Agile Manifesto (2001) was a milestone for Agile Software Development (ASD). A group of 17 experts proposed practices to improve software development by defining values and principles to guide the agile methods. They considered that traditional approaches were heavy and made the development process very bureaucratic. In this context, the adoption of agile methods has emerged as an alternative to traditional models (VERSIONE, 2016). The Agile Manifesto establishes four values (AGILE MANIFESTO, 2001):

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

The Agile Manifesto proposes the adoption of flexible and adaptive processes to accept the changes as an inseparable part of its development process. A set of 12 agile principles were defined and should be followed to minimize the risks caused by the frequent changes in requirements (AGILE MANIFESTO, 2001):

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software;
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage;
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale;
- Business people and developers must work together daily throughout the project;
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done;
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation;
- Working software is the primary measure of progress;
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely;
- Continuous attention to technical excellence and good design enhances agility;
- Simplicity is the art of maximizing the amount of work not done;
- The best architectures, requirements, and designs emerge from self-organizing teams;

- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Based on the principles and values of the Agile Manifesto, several methods have been proposed, such as Adaptive Software Development (ASD), Crystal, Dynamic Systems Method (DSDM[1]), eXtreme Programming-XP (BECK, 1999), Feature Driven Development (FDD[2]), Lean Development (LD) and Scrum (SCHWABER, 2001). Several practices have been proposed for use in ASD as shown in Figure 2.1.

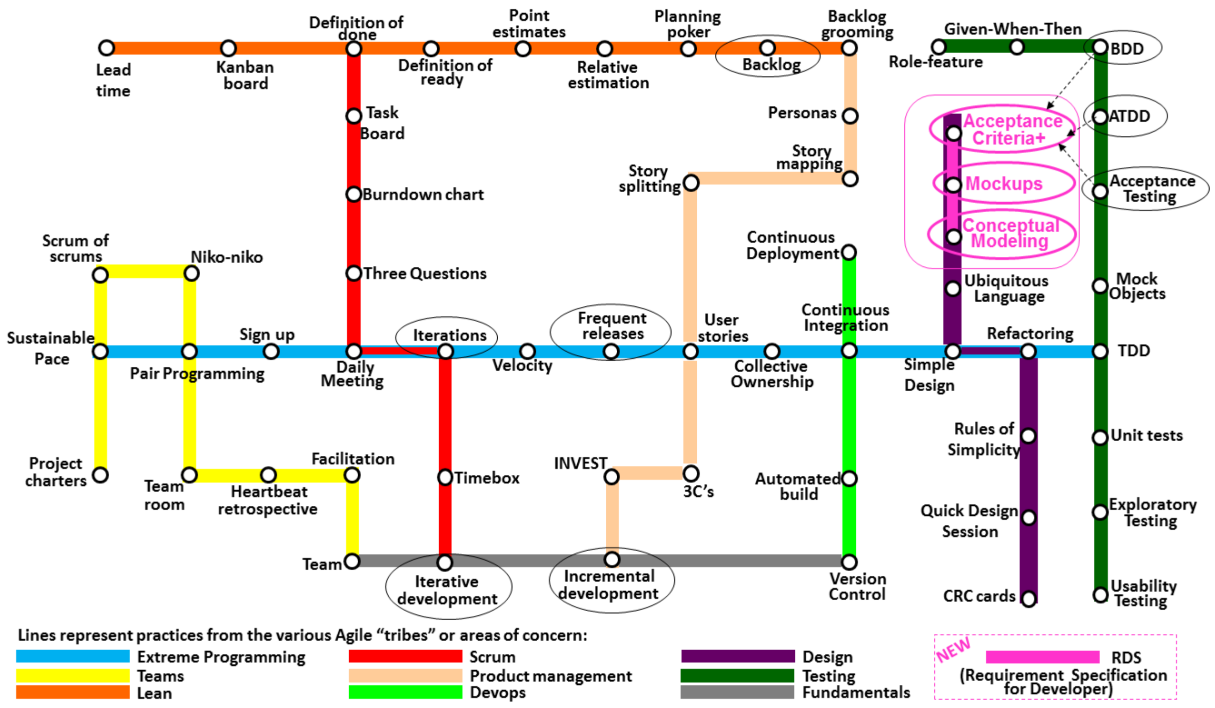


Figure 2.1 "Subway map" Agile Practices adapted from (Agile Alliance, 2016)

The approach proposed (RSD) in this thesis follows some agile practices, such as the fundamental practices (gray line): *Iterative Development* and *Incremental Development*. RSD approach (described in Chapter 6) uses two eXtreme Programming practices (blue line): *Iterations* and *Frequent Releases*, and also uses the *Backlog* concept of Lean (salmon line). Besides, RSD approach introduces three design practices (purple line): Conceptual Modeling, Mockups, and Acceptance Criteria. This latter adds the definition of technical aspects together with some of the concepts of the testing practices like Behavior Driven Development (BDD), Acceptance Test Driven Development (ATDD) and Acceptance Testing (AT) (AGILE ALLIANCE,

[1] https://www.dsdm.org/

[2] http://agilemodeling.com/essays/fdd.htm

2016). In next section, we present a brief description of the design practices used by the RSD approach.

2.3 Design Practices

In the scope of this work, design information is related to the solution domain and therefore focuses on describing how functional, non-functional and system requirements should be implemented.

An SRS closer to what is implemented may reduce the effort required to coding, testing and maintaining (BJARNASON et al., 2011). The existing gap between the specifications of the customer's needs and the details required to produce the solution is a challenge that compromises the ASD (RUDORFER et al., 2012). According to Heikkila et al. (2015), the design information is neglected in SRS in ASD. Agile methods tend to focus on functional requirements. Technical aspects are overlooked in the early phases and have no detailed design. This makes it harder to implement it later in the process (HAUGSET and STALHANE, 2012). The design practices help to bridge the gap between the problem and the solution domains, providing a better understanding for the developer who will implement a feature.

2.3.1 *Acceptance Criteria+*

The Test-Driven Development (TDD) is an agile practice that has been growing in companies, and it refers to a style of programming in which 3 activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring) (ALLIANCE, 2016). According to Beck (2002), test-first coding is not new. It is nearly as old as programming. It is an analysis and design technique.

RSD approach adopts the Acceptance Criteria (AC) used in User Stories (US). AC is based on the concept of the Acceptance Tests (AT) introduced in XP (BECK, 1999), which defines constraints for the US. 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 2.1.

Table 2.1 – Differences between AC and AC+

	AC	AC+
Link	Specific to a single user story (WHICHARD, 2016);	Can be reused by several requirements;
Scope	Focus on constraints related to the business rules (BECK, 1999);	Can be a business rule, interface, validation, technical or any other type of constraint;
Oriented to	Directed to the customer and described at high level, without much detail to developer (MAMOLI, 2016);	Directed to developer and technical jargon can be used;
Writer	Should be written by customers (BECK, 1999);	Any stakeholder;
Domain	Problem (POVILAITIS, 2016).	Problem and solution.

Like TDD, Acceptance Test Driven Development (ATDD) also involves creating tests before coding, and these tests represent expected behavior of the software. In ATDD, the team creates one or more acceptance-level tests for a feature before implementing it. Typically these tests are discussed and captured when the team collaborates with the business stakeholder(s) to understand a story in the backlog. 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 (AGILE ALLIANCE, 2016). BDD focuses on the behavioral aspect while the TDD focuses on the implementation aspect. BDD is usually done in the language of the domain experts to facilitate understanding rather than exposing the code level tests. It's defined in a GIVEN, WHEN & THEN format in order to structure the test definition.

AC+ uses concrete examples as a strategy to clarify the understanding of complex rules, like ATDD and BDD. However, AC+ uses a language aimed at the developer and also includes NFR and technical aspects, and not only functional requirements as ATDD and BDD. Table 2.2 illustrates the relationship among the AC+ and others agile practices.

Table 2.2 – Relationship among some agile practices

Agile Practices / Scope	TDD	AT	ATDD	GIVEN THAN THAT	BDD	AC +
Coding of Acceptance Tests (Programming Language)	X		X			
Acceptance testes for Functional Requirements (Customer-oriented)		X	X		X	X
Concrete Examples			X		X	X
Structuration of Acceptance Tests				X	X	
Acceptance testes for NFR and technical aspects (team-oriented)						X

2.3.2 *Conceptual Modeling*

Traditionally an information system is defined regarding two perspectives: one related to its function and the other to its structure (HIRSCHHEIM, 1995). The functional perspective of a system results in a high-level description of the system's functionality from a user's point of view. From a structural perspective, a system is depicted regarding entities, static relationships, and constraints. Conceptual modeling is the systematic activity of describing some aspects of the physical and social world around for purposes of understanding and communication (LOUCOPOULOS, 1992). According to Olivé (2007), conceptual models are needed to achieve a common understanding of the system domain among all stakeholders.

Although the Conceptual Modeling is an established practice in traditional approaches, it is not systematically used in popular agile methods, such as scrum (SCHWABER, 2001) and XP (BECK, 1999). The requirements approaches employed in ASD, such as US, focus on functional modeling. There is no concern in defining the conceptual model in a systematic way together with US. 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 Cohn (2004). However, it does not treat other concepts involved in the business domain and not address the relationship between these actors, it treats only actors.

The conceptual modeling is not part of the agile methods, an exception can be found in FDD which is an agile method where the functional requirements are described by features, sequence diagrams, and the conceptual modeling is used through an object model that evolves into a class model with attributes and operations. However, FDD treats only functional requirements. NFR and other system requirements are not addressed.

The approach proposed in this thesis links the functional and structural perspectives in order to have a single view. The acceptance criteria (AC+) define functional requirements, non-functional and technical aspects under the developer's point of view. The structural perspective is obtained from the representation of concepts (entities) and their relationships without defining operations. The prototype of the user interface, where applicable, interconnects the other two perspectives.

2.3.3 *Prototyping the User Interface*

User interface prototypes (also known as mockups or wireframes) are drawings that show how the user interface of the software is supposed to look during the interaction between that software and the end-user. Mockups may be very simple, just to help the presentation of the user-system interactions, or more detailed with rich graphics, whenever specific constraints on the graphical user interface are highlighted, e.g., specific logos or branding related colors (RICCA, 2014). Although many tools[3] exist for drawing screen mockups, several professionals prefer to sketch screen mockups on paper.

Mockups have proven an efficient practice to capture and defining functional requirements (INAYAT, 2015; RICCA, 2014). Mockups improve requirements gathering, without implying an additional effort in the process. One of their advantages is that they are technically valuable for developers and, at the same time, fully understandable by end-users (RIVERO, 2014).

Many agile projects require user interaction design, but the integration of mockups into ASD is not well understood (NEBE, 2016). The popular agile methods do not use mockups as part of their process, an exception can be found in DSDM that is also one of the agile methods whose authors contributed to the definition of the Agile Manifesto (2001). DSDM is considered the heaviest agile method because it uses thirteen roles and establishes 14 artifacts to be produced during the development process. Hence its adoption in the industry has been very low compared to other methods such as Scrum and XP (VERSIONE, 2016). So, the companies need to adapt their processes to integrate the mockups in ASD.

The technique proposed in this thesis uses mockups in order to interconnect the acceptance criteria (AC+) and conceptual models, and thus facilitate the understanding of the developers.

[3] http://www.creativebloq.com/wireframes/top-wireframing-tools-11121302

2.4 Summary

In this chapter, essential concepts about the quality of SRS and some practices used in ASD were reviewed, such as AT, TDD, ATDD and BDD. The concept of AC+ based in these practices was introduced.

The benefits of design practices consolidated in traditional development, such as conceptual modeling and mockups, were discussed in the context of ASD. Next chapter presents the study conducted in the literature to investigate these concepts and the phenomenon of the requirements specification activity in ASD in order to identify the initial theoretical background of this thesis.

3 LITERATURE REVIEW

This chapter describes the Systematic Mapping (SM) that investigated in the literature how the requirements engineering has been conducted in agile projects. First, in Section 3.1, the study design is described. Section 3.2 presents a summary of the research protocol describing the procedures used to select, collect, and analyze data. Section 3.3 presents and discusses a summary of the results. Finally, Section 3.4 presents the threats do validity of the SM.

3.1 Study Design

The SM was conducted to investigate how RE is used in projects that adopt agile methods. This exploratory study aimed identifies the RE approaches that were being running in ASD, as well as the problems and challenges related. The scope of the study focused on primary research papers describing work validated empirically. The relevant studies identified were analyzed and interpreted to answer the research questions defined in Section 3.2.

This study followed the guidelines suggested by Kitchenham and Charters (2007), and Travassos and Biolchini (2007) to leave the results more reliable, auditable and able to be replicated by other researchers. The procedure method defined for this study was the thematic synthesis (CRUZES and Dyba, 2011), which aims to identify the themes or recurrent matters in several studies. Moreover, the method defines procedures to interpret, explain and draw conclusions from review results as described in Appendix A (Section 5.6).

According to Marconi and Lakatos (2008), the variables of a study can be considered independent or dependent. The independent determine the condition or cause for a result, affecting or determining the dependent variables. In this study, the independent variables are requirements engineering and agile methods. The dependent variables are approaches for the elicitation and specification of the requirements and the challenges of the requirements engineering in ASD. The methodological framework of the SM is shown in Table 3.1.

Table 3.1 – Methodological Framework of the Systematic Mapping

Approach Method	Inductive
Analysis	Qualitative
Procedure Method	Synthesis thematic
Independent variables (X)	Requirements engineering and Agile methods
Dependent variables (Y)	Approaches for the elicitation and specification of the requirements and the challenges of the requirements engineering in ASD

3.2 Procedures

The SM was conducted in seven phases as shown in Figure 3.1, also had the participation of two other researchers (master students) during all the process. A summary of the procedures executed is described in this section. The search string and the complete research protocol can be found in Appendix A. The main results of the analysis and synthesis are detailed in Section 3.3. All procedures executed and the complete results are described in Medeiros et al. (2015) and Alves (2015).

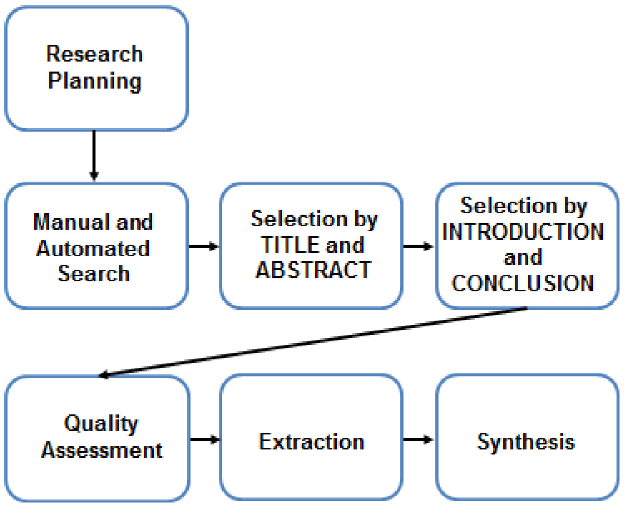

Research Planning
Manual and Automated Search
Selection by TITLE and ABSTRACT
Selection by INTRODUCTION and CONCLUSION
Quality Assessment
Extraction
Synthesis

Figure 3.1 Systematic Mapping Steps

At the planning stage, the research protocol was developed describing the research questions, procedures, and methods used to perform the study. The SM seeks answers to RQ1 defined in Section 1.2: *How the requirements engineering has been conducted in projects that adopt agile methods?*. Specific Research Questions (SRQ) were defined to guide the extraction, analysis and synthesis:

- **SRQ1:** What approaches are being used to elicit requirements in projects that adopt agile methods?
- **SRQ2**: What approaches are being used to specify requirements in projects that adopt agile methods?
- **SRQ3**: What are the challenges and limitations regarding the requirements engineering activities in agile projects?
- **SRQ4**: What are the implications for the software industry and academy arising from requirements practices currently used in agile projects?

The automated search was conducted through the Reviewer[4] tool that executed the string search simultaneously in all the sources (IEEE Xplore Library, ACM Library, Science Direct, Springer Link and Scopus). Before starting the SM, some tests were performed, running the search string in order to validate and adjust the mapping protocol. The manual search was conducted in the proceedings of the following conferences on requirements engineering and agile methods: Requirements Engineering Conference and Agile Development Conference. In the third phase, the criteria for inclusion and exclusion (Appendix A, Table A.2) were applied by reading the titles and abstracts. One of the criteria used to select the papers was that the study should have some empirical validation.

In the fourth phase, the same criteria were applied by reading the introduction and conclusion of the resulting studies of the third phase. In the fifth phase, all sections of the papers were read to assess the quality of the papers from the previous stage. A questionnaire adapted from Dyba and Dingsoyr (2008) was used, which had ten questions to guide the evaluation (Appendix A, Table A.3). One of them assessed if the papers were the result of research, or were merely a consolidation of lessons learned based on the opinion of an expert. Another question assessed whether the paper had a discussion about the contribution to practice or literature. A three-point scale of Likert was used to evaluate the papers: 0 (Nothing meets the criteria evaluated); 0.5 (The paper does not make clear whether or not meet the criteria) and 1 (Paper meets the criteria evaluated). Once calculate the sum of the scores of all the questions, the paper was classified into one of the four groups: low, medium, high or very high quality. Studies with low quality (the sum less than three) were excluded.

The next stages were the extraction and synthesis of the papers selected in the previous phases. The papers were divided among the three researchers to per-

[4]Reviewer (https://github.com/bfsc/reviewer)

form the extraction using a standard form (Appendix A, Table A.4). The synthesis was conducted by a researcher and then reviewed by another. The thematic synthesis technique was used following the guidelines suggested by Cruzes and Dyba (2011). The coding procedure was done from the reading of the forms containing the extracted data. For SRQ3 synthesis, the challenges were identified with a code. Then, the codes were grouped into themes. A review of the codes was performed trying to identify similarities, duplicate and undue codes. The next step was the grouping of subjects into categories (or high-level themes). The codes, themes, and categories were successively revised until you get the results presented in Section 3.3. The results were analyzed with MS Excel® software support, which was also used to generate the graphs and tables.

3.3 Results

We retrieved a total of 2852 papers from the automated (2501) and manual (351) search. After the fifth phase, 24 Primary Studies (PS) remained for a deeper analysis, as show in Figure 3.2. The SM collected and analyzed evidence from 68 companies, involving a total of 270 people from the 24 PS. Appendix B presents the list of the 24 PS.

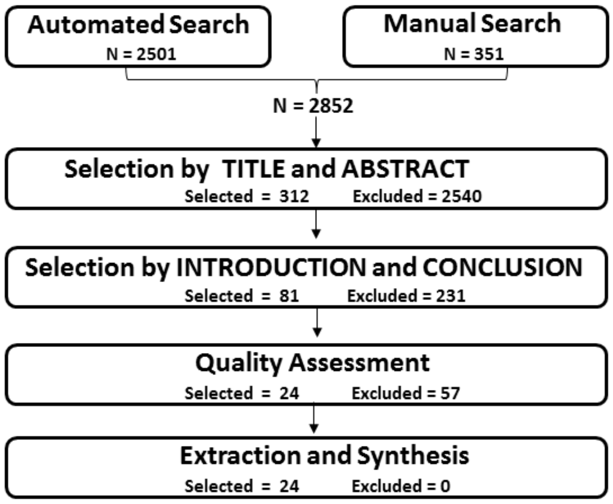


Figure 3.2 Primary studies selected by phase

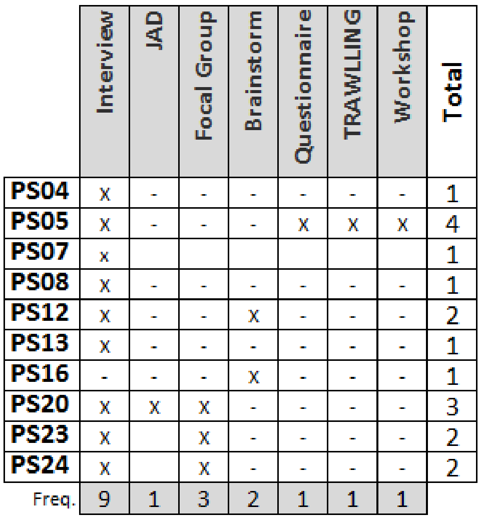
A summary of the results is presented as follows by research question.

SRQ1: What approaches are being used to elicit requirements in projects that adopt agile methods?

Requirements Elicitation is a process through which the acquirer and the suppliers of a system discover, review, articulate, understand, and document the requirements of the system and the life cycle processes (ISO-IEEE 830, 1998).

The synthesis of the PS pointed out that seven different approaches are being used to requirements elicitation in agile projects, as shown in Table 3.2. Of the 24 papers analyzed, only ten papers reported the approach used to elicit requirements. Interview with the client is the most used approach. JAD (Joint Application Development), Focus Group, Brainstorm, Questionnaire, Trawling[5] and Workshop also were cited as approaches for gathering requirements. Some primary studies reported the use of more than one approach, for example, **PS05** (Primary Study #5) reported the use of 4 approaches: Interviews, Questionnaires, Trawling, and Workshops.

Table 3.2 – Primary Studies versus approach for requirements elicitation



	Interview	JAD	Focal Group	Brainstorm	Questionnaire	TRAWLLING	Workshop	Total
PS04	x	-	-	-	-	-	-	1
PS05	x	-	-	-	x	x	x	4
PS07	x							1
PS08	x	-	-	-	-	-	-	1
PS12	x	-	-	x	-	-	-	2
PS13	x	-	-	-	-	-	-	1
PS16	-	-	-	x	-	-	-	1
PS20	x	x	x	-	-	-	-	3
PS23	x		x	-	-	-	-	2
PS24	x		x	-	-	-	-	2
Freq.	9	1	3	2	1	1	1	

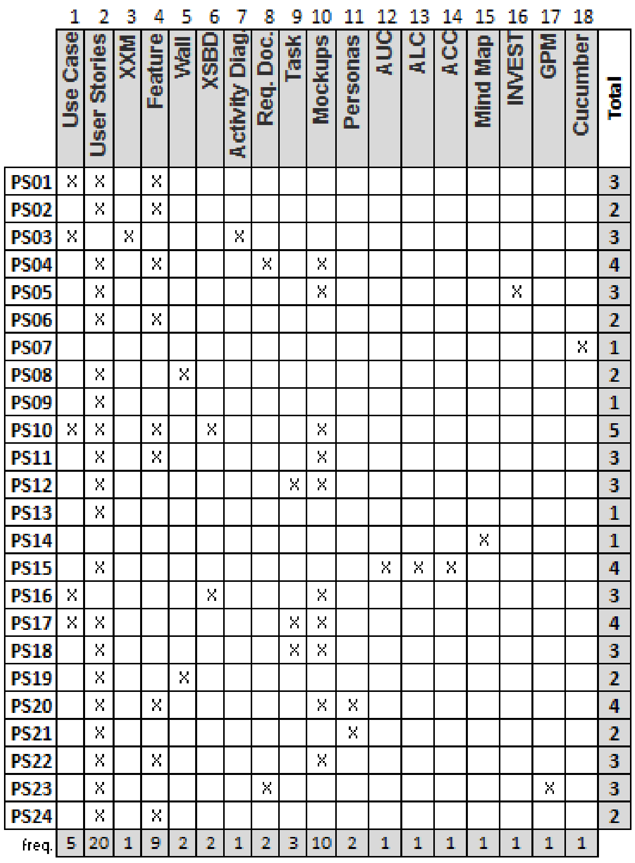
SRQ2: What approaches are being used to specify requirements in projects that adopt agile methods?

Eighteen different approaches to specify requirements have been reported in primary studies analyzed, as shown in Table 3.3. The most widely used approach is User Story. It was reported by more than 80% of the papers (20). Mockup is the second most used approach (10). Five papers reported using more traditional approaches such as Use Case\Scenarios (**PS1**, **PS3**, **PS10**, **PS16** and **PS17**). One paper (**PS10**) reported the use of five approaches. Although nine approaches have been reported in only one paper, they were considered in this study, given that the

[5]Trawling (from Mike Cohn's book "Agile planning and estimating")

paper presented evidence that they have been validated empirically in the industry. These approaches are: XXM (eXtreme X-Machine), AUC (Agile Use Case), ALC (Agile Loose Case), ACC (Agile Choose Case), INVEST (Independent, Negotiable, Valuable, Estimable, Small and Testable), Activity Diagram, Mind Map, GPM (Goal Preference Model) and Cucumber[6]. In fact, AUC, ALC, ACC are extensions of Uses Cases, and INVEST are principles for writing good User Stories.

Table 3.3 – Primary Studies versus approach for requirements specification



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
	Use Case	User Stories	XXM	Feature	Wall	XSBD	Activity Diag.	Req. Doc.	Task	Mockups	Personas	AUC	ALC	ACC	Mind Map	INVEST	GPM	Cucumber	Total
PS01	X	X		X															3
PS02		X		X															2
PS03	X		X				X												3
PS04		X		X				X		X									4
PS05		X								X						X			3
PS06		X		X															2
PS07																		X	1
PS08		X			X														2
PS09		X																	1
PS10	X	X		X		X				X									5
PS11		X		X						X									3
PS12		X							X	X									3
PS13		X																	1
PS14															X				1
PS15		X										X	X	X					4
PS16	X					X				X									3
PS17	X	X							X	X									4
PS18		X							X	X									3
PS19		X			X														2
PS20		X		X						X	X								4
PS21		X									X								2
PS22		X		X						X									3
PS23		X						X									X		3
PS24		X		X															2
freq.	5	20	1	9	2	2	1	2	3	10	2	1	1	1	1	1	1	1	

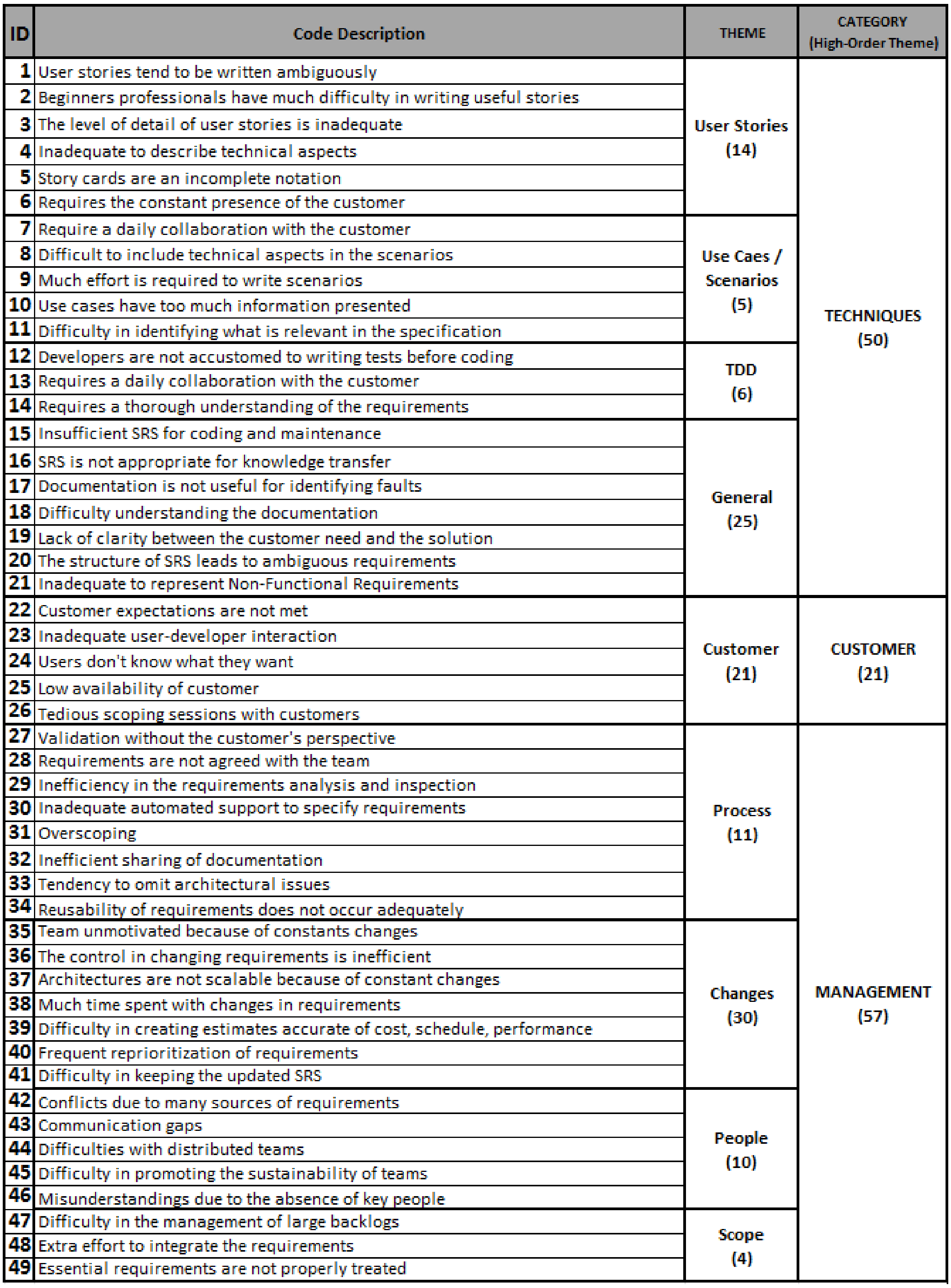
SRQ3: What are the challenges and limitations regarding the requirements engineering activities in agile projects?

The recommendations of Cruzes and Dyba (2011) were followed to map out the challenges of requirements activities in agile projects. Initially, 115 codes (challenges), 15 themes and 7 categories were identified. After revisions and successive refinements, eliminating duplication and grouping similarities, we reached a total of 49 codes, 9 themes, and 3 categories. Table 3.4 shows the challenges (factors) that

[6]https://cukes.info/

have been mapped by category and theme showing the number of occurrences within the parentheses. Each code received an identifier to facilitate its identification.

Table 3.4 – Requirements engineering challenges in agile projects



ID	Code Description	THEME	CATEGORY (High-Order Theme)
1	User stories tend to be written ambiguously	User Stories (14)	TECHNIQUES (50)
2	Beginners professionals have much difficulty in writing useful stories		
3	The level of detail of user stories is inadequate		
4	Inadequate to describe technical aspects		
5	Story cards are an incomplete notation		
6	Requires the constant presence of the customer		
7	Require a daily collaboration with the customer	Use Caes / Scenarios (5)	
8	Difficult to include technical aspects in the scenarios		
9	Much effort is required to write scenarios		
10	Use cases have too much information presented		
11	Difficulty in identifying what is relevant in the specification		
12	Developers are not accustomed to writing tests before coding	TDD (6)	
13	Requires a daily collaboration with the customer		
14	Requires a thorough understanding of the requirements		
15	Insufficient SRS for coding and maintenance	General (25)	
16	SRS is not appropriate for knowledge transfer		
17	Documentation is not useful for identifying faults		
18	Difficulty understanding the documentation		
19	Lack of clarity between the customer need and the solution		
20	The structure of SRS leads to ambiguous requirements		
21	Inadequate to represent Non-Functional Requirements		
22	Customer expectations are not met	Customer (21)	CUSTOMER (21)
23	Inadequate user-developer interaction		
24	Users don't know what they want		
25	Low availability of customer		
26	Tedious scoping sessions with customers		
27	Validation without the customer's perspective	Process (11)	MANAGEMENT (57)
28	Requirements are not agreed with the team		
29	Inefficiency in the requirements analysis and inspection		
30	Inadequate automated support to specify requirements		
31	Overscoping		
32	Inefficient sharing of documentation		
33	Tendency to omit architectural issues		
34	Reusability of requirements does not occur adequately		
35	Team unmotivated because of constants changes	Changes (30)	
36	The control in changing requirements is inefficient		
37	Architectures are not scalable because of constant changes		
38	Much time spent with changes in requirements		
39	Difficulty in creating estimates accurate of cost, schedule, performance		
40	Frequent reprioritization of requirements		
41	Difficulty in keeping the updated SRS		
42	Conflicts due to many sources of requirements	People (10)	
43	Communication gaps		
44	Difficulties with distributed teams		
45	Difficulty in promoting the sustainability of teams		
46	Misunderstandings due to the absence of key people		
47	Difficulty in the management of large backlogs	Scope (4)	
48	Extra effort to integrate the requirements		
49	Essential requirements are not properly treated		

It is important to notice that the number of occurrences do not reflect the importance of the theme\category, but only the sum of papers that the challenges were cited. Although 49 challenges have been mapped, some of them were reported in several papers, as shown in Table 3.5. For example, challenge #15 (Insufficient SRS for coding and maintenance) was reported in nine primary studies. Thus, a total of 128 occurrences of challenges were counted.

Table 3.5 – Primary studies versus challenges of RE activities in ASD

		Challenges																																																	
	ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	sum
Primary Study (PS)	1																			×							×	×		×					×			×					×								7
	2															×				×	×		×									×				×	×		×	×	×	×		×							12
	3										×	×				×		×	×							×																									6
	4												×	×	×									×		×												×		×	×			×					×		10
	5																																														×				1
	6															×				×	×		×						×			×		×		×	×	×	×	×	×		×	×							15
	7													×	×	×						×		×		×				×							×						×							×	10
	8	×				×																																													2
	9																																				×														1
	10																																×									×		×	×						4
	11																				×				×												×														3
	12		×	×												×																					×			×								×			6
	13			×																																															1
	14													×																																					1
	15																					×																													1
	16																						×							×																					2
	17							×	×	×														×		×											×	×													7
	18	×		×												×	×						×																	×								×			7
	19					×	×									×	×																																		4
	20		×	×												×	×							×	×	×	×										×				×					×					11
	21	×		×	×																×																	×													5
	22																														×						×														2
	23																			×				×																											2
	24															×				×			×	×		×											×			×	×										8
	sum	3	2	5	1	2	1	1	1	1	1	1	1	3	2	9	3	1	1	5	4	2	5	6	2	6	2	1	1	3	1	2	1	1	1	2	10	5	2	6	5	2	3	4	1	1	1	2	1	1	128

The themes **Changes** (30) and **Customer** (21) have the highest occurrences of problems – 30 and 21, respectively. It is important to highlight the code #25 (Low availability of the customer) with 6 occurrences in the Customer category and #36 (Control of changes in requirements is insufficient), with 10 occurrences in the category changes. It signals that the agile principles "*Teams adapt quickly to changes*" and "*Continuous interaction with the customer*" are not practices consolidated in the companies investigated in the primary studies.

Looking at the categories it can also be noticed a lot of occurrences of issues in **techniques** (50) used for requirements specification, mainly User Stories/Cards (14) and Scenarios/Use Cases (5). The General category (25) groups the challenges that are not specific to a particular technique, i.e., apply to all techniques used to specify requirements in ASD. The studies **PS08** (ABDULLAH, 2011), **PS18** (READ, 2012) and **PS21** (SAVOLAINEN, 2010) identified that User Stories are customer-oriented and therefore tend to be described in a prolix and ambiguous way (#1). Other studies, **PS12** (GREGORIO 2012), **PS13** (LORBER 2013) and **PS09** (BATOOL 2013) reported that the level of detail of the User Stories is inappropriate, (#3). The

PS19 (SHARP, 2009) considers that Story Card is an incomplete notation (#5). According to **PS03** (THOMSON, 2008), Use cases have too much information presented, a lot of irrelevant information (#10) and the **PS17** (OBENDORF, 2008) reported difficulties to include technical aspects in the Scenarios (#8).

Only one paper **PS22** (BATOOL, 2013) reported problems regarding the automated support to specify requirements in ASD (#30). Thirteen primary studies reported the use of some practice for requirements validation (TDD, Acceptance Tests or Test Cases). In these studies, some problems were reported about requirements validation (#27, #28 and #29).

A more detailed discussion of these challenges is presented in Alves (2015) and Medeiros (2015) who used a different numbering for codes as well as a different grouping of categories and themes.

SRQ4: What are the implications for the software industry and academy arising from requirements practices currently used in agile projects?

The findings raise some gaps that require further research in this area, such as:

- What factors affect the quality of SRS in ASD and how those factors affect the work of the software engineers?
- What adjustments can to be made in the current approaches used to specify requirements in agile projects?
- Is the productivity of teams compromised by the adoption of RE in agile projects?
- Is it worse the quality of SRS in agile projects when compared to SRS in traditional projects?

Another question that deserves attention from the academic community is the low quality of the papers. Initially, 31 papers would be used for data extraction. However, during the quality analysis stage, 23% of papers (7) were excluded due to low quality. Of the 24 papers selected, only 3 papers had a very high quality (**PS03**, **PS04**, **PS06**). Only one paper appropriately considered the relationship between the researcher and the other people involved in the research. This point requires closer attention from researchers in the conduction of their studies for the results to be effectively used. An interesting fact is that the vast majority, 20 papers, are academic studies, but with validations in real projects in the industry.

Based on the results obtained, it is observed that as in the traditional development, the ASD also presents a significant number of issues, mainly related to the requirements management and techniques used. Of a total of 128 occurrences of problems, 57 are related to the requirements management: Process, Changes, People, and Scope. This signals the need for more in-depth investigations into companies to analyze the development practices that entail such bottlenecks and compromise the productivity of the teams, the quality of SRS, the motivation of teams and appropriate collaboration with the customer.

3.4 Threats to Validity

A common threat of systematic mappings is the difficulty of finding all the relevant existing papers. To minimize this problem, we used multiple sources of data, automatic and manual. Among the automatic, are the four major search engines of software engineering, cited by Kitchenham and Charters (2007) and Dybå and Dingsoyr (2008).

Pilot tests were performed before starting the stages of research. The selection of the studies was conducted by more than one researcher in order to avoid bias. When there were differences of opinion among researchers, they were confronted and resolved with the participation of another researcher.

3.5 Lessons Learned

The use of the Reviewer tool for automated search facilitated the initial analysis from the title and abstract. The tool generated a spreadsheet with this information so that there was no need to download the papers.

Initial planning predicted the participation of four undergraduate students to work in the first phase of the mapping. However, the evaluation carried out by undergraduate students presented many divergences between them. In this way, one of the three researchers needed to mediate the conflict which required a lot of effort. Thus, the participation of the undergraduate students was canceled given that it was not worth it.

3.6 Summary

In this chapter, we presented a summary of the protocol, procedures, and results of the systematic mapping that investigated in the literature how the requirements engineering has been conducted in software projects that adopt agile methods. The SM was also published in an international conference (MEDEIROS et al., 2015) and in a Brazilian journal (MEDEIROS et al., 2015c).

We mapped the approaches used to elicit and specify requirements and 49 challenges that affect the requirements engineering activities in ASD. According to the data collected, the most used approach to elicit requirements is the interview. User Stories was reported by more than 80% of the papers, as the most used approach to specifying requirements. The studies indicate several problems due to Frequent Changes in Requirements (*Changes* Theme), Low Involvement of the Customer (*Customer* Theme) and regarding to the approaches used to requirements specification (*Techniques* category).

The findings of the systematic mapping were used as input to the investigation of the phenomenon in the industry, as shown in next chapter. In case studies, we investigated whether the challenges of the *Techniques* category affect the quality of SRS in ASD, how these challenges are related and how they affect the work of the software engineers. Also, we attempted to identify new factors that compromise the requirements specification activity in ASD.

4 INVESTIGATION IN THE INDUSTRIAL PRACTICE

This chapter carefully explains the empirical studies conducted to investigate the requirements specification activity in the context of ASD at six companies. Section 4.1 describes the research design. Data collection and analysis procedures are described in Section 4.2. The results of each case study are presented in section 4.3.

4.1 Design

The Systematic Mapping (SM) presented some challenges related to RE in ASD and some gaps that demand more investigations. Moreover, the literature review conducted by Inayat et al. (2014) shows that there is a need for more empirical studies using agile methods. So, this study aims to investigate the phenomenon of the requirements specification activity in different agile contexts and then build an explanatory model providing a deeper description to point out the factors that should be considered to write more useful SRS in ASD. More specifically, our goal is to find answers to the Research Question 2 (RQ2) defined in Section 1.2: *What factors affect the quality of SRS in ASD and how those factors affect the work of the team?*.

A multi-case study was conducted in six companies to investigate the phenomenon in industrial practice and gather data to evaluate the initial theoretical background identified in SM. The case studies are classified as instrumental since our goal is to understand the constructs and then build a model (described in Chapter 5). Six independent case studies were conducted following a single standard protocol and the guidelines suggested by Runeson and Martin (2009) composed of five steps: Planning; Preparation; Collecting evidence; Analyzing the data collected; and Synthesis. The phenomenon was investigated from the perception of the software engineers (analysts, developers, testers, among others).

4.1.1 *The Sample*

Usually, two levels of sampling are necessary in qualitative case studies: the cases that will be investigated and the participants (FRANÇA et al., 2014). The pre-condition for selection of cases was the use of agile practices in the software development. To increase data diversity, we looked for companies with different characteristics regarding the size, sector of activity, the maturity and core business. To conduct the research, we depended on acceptance and willingness of the companies. The initial sampling was formed by companies that the researchers had contact with the directors or software engineers. Six companies fulfilled the preconditions. The projects investigated were selected by the companies.

The second level of sampling defines the software engineers participating in each case study. All the software engineers of the projects selected by the companies were pre-assigned to participate in the study. Although all have participated in the observation activities, we did not interview all the pre-assigned software engineers due to the unavailability of some (vacation, travel, sick, etc.), while in others we had already reached a satisfactory understanding of the phenomenon in the company. So, conducting more interviews would not add new information relevant to our study. However, we tried to achieve a good coverage (i.e. diversity) of age, background and education, years of experience and employment in the company, work experience on different activities within software development, among others, to ensure a fertile sample (See Appendix D, Table D.1).

4.2 Procedures

First, the study protocol was created describing the procedures to collect and analyze the data, as described in next sub-sections. For 10 months, we conducted observations, interviews, and document analysis to collect and analyze data.

4.2.1 *Data Collection*

Observations

Even being agile projects, the daily team meetings were not mandatory in all companies. They occurred whenever necessary and at least, once a week. Meetings aimed to share best practices, problems and difficulties faced by the team. The researchers (the author and a master student) participated in some meetings acting as observers, taking notes, perceiving how software engineers were using the SRS and the difficulties faced. The researchers did not interfere in the way the project was being conducted (for example, making suggestions or criticisms).

Interviews

A questionnaire was elaborated to guide the interviews (See Appendix C). The questions were presented in a funnel format, beginning with general questions and moving towards more specific ones (RUNESON and MARTIN, 2009). A pre-test was conducted with two pilot interviews. The initial version of the questionnaire had 30 questions. After the first case study, we reviewed the questionnaire, and excluded 5 questions because their answers turned out not to be useful to our investigation. The remaining 25 questions were used in the subsequent case studies.

After obtaining the authorization of the companies to contact directly each potential participant, an initial contact with them was established via email inviting them to participate in the study. The interviews were scheduled and conducted individually. Before each interview, the purpose of the study was clarified with the participants. We made the interviewee aware of the importance to detail the answers as much as possible. In addition to registering answers, we also recorded the interviewee's observations that were out of the original questions list but were useful for the study. At the end of each interview, the questionnaire was reviewed with the interviewee to confirm that the answers had been correctly noted, and to capture complementary information, if needed.

Analysis of Documents

Documents are an important source of data for qualitative research (MERRIAM, 2009). Documentation helped in the analysis of data collected during the interviews. The companies elaborated such documentation before the study. Therefore, it had no kind of bias from this research. The researchers analyzed different artifacts, such as User Stories (US), use cases, diagrams, among others. In some projects due to confidentiality constraints, the documents were analyzed with the presence of a member of the team. In these projects, the company did not provide a copy of the artifacts to the researchers.

4.2.2 *Data Analysis in each case study*

Data analysis is the process of making sense out of the data (MERRIAM, 2009). It involves consolidating, reducing, and interpreting what people have said and what has been seen and read. The goal of data analysis is to find answers to research questions. The data collection and analysis were conducted simultaneously, in incremental and iterative steps.

The interviews were transcribed, and then techniques of ground theory (SJØBERG et al., 2008) proposed by the constant comparison method (SEAMAN, 2008) was used to code, categorize and synthesize the data in each study. The process begins with an open coding of the field notes, which involves attaching codes, or labels, to pieces of text that are relevant to a particular theme or idea of interest in the study. After coding the transcriptions, the codes were reviewed to identify similarities, duplicates, and undue codes. The codes arising from each interview were constantly compared to codes in the same interview and from other interviews. From the constant comparisons of the codes, we grouped them into categories that represent factors that affect the quality of SRS in ASD. Figure 4.1 shows an example of the codes building process. The data collected in the interviews were triangulated with the data obtained from the analysis of documents and observations in order to increase the credibility of the data. The triangulation technique prevents the influence of individual analysis based on interviewer-researcher's personal opinion (NORTHCUTT and MCCOY, 2004).

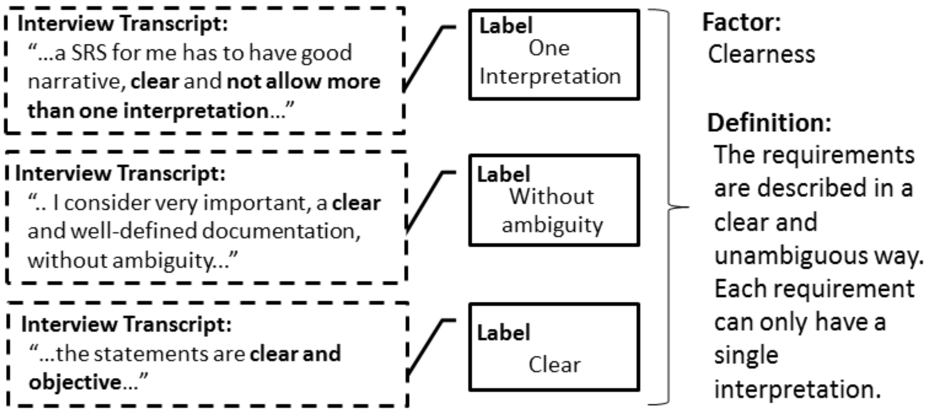

Interview Transcript:
"...a SRS for me has to have good narrative, **clear** and **not allow more than one interpretation**..."
Label
One Interpretation
Interview Transcript:
".. I consider very important, a **clear** and well-defined documentation, without ambiguity..."
Label
Without ambiguity
Interview Transcript:
"...the statements are **clear and objective**..."
Label
Clear
Factor:
Clearness
Definition:
The requirements are described in a clear and unambiguous way. Each requirement can only have a single interpretation.

Figure 4.1 Factors building example

Then, the relationships among factors were analyzed, leading to explanatory propositions. Propositions are causal relationships among concepts that offer an explanation of a phenomenon (PANDIT, 1996). Although the grounded approach produces conceptual rather than measured relationships, propositions can and should be subject to empirical verification. Finally, we created the story for each case study. A story is simply a narrative about the phenomenon of study (PANDIT, 1996).

4.3 Results in each case study

This section describes the context of the companies and the story line about the factors that affect the quality of SRS in the projects investigated.

4.3.1 *Case Study 1 (a government public software organization)*

a) Context

This case study was carried out with software engineers from a company established over 40 years ago by the Federal Government of Brazil. Its core mission is to provide information and communication technology solutions for the improvement and implementation of social policies of the Brazilian state. The company has development units in five states. As a government-owned organization, it is regulated by the laws and norms of the Brazilian public sector.

The entire organization has 3800 employees, 1381 of which compose the software engineering workforce. The case study was conducted with software engineers of two agile projects carried out in one of the development units. Only permanent employees, i.e., no trainees, were involved. The fourteen interviewees had an average of 11 years of experience with software development. The team profile is described in Appendix D (Table D.1). Part of the development was geographically distributed in three cities.

One of the projects aimed to develop an information system. The other intended to construct an On-Line Analytical Processing (OLAP) solution. In the past, the organization used a process based on RUP (Rational Unified Process), but in the last two years it has been adopting ASD in some projects, including the two investigated in this study. Each sprint lasted at most one month. At the end of each sprint, a version was released to validation by the internal Product Owner (PO). However, a version was released to the customer only every two or three months.

Some agile principles were well consolidated in the company, such as priority on customer satisfaction and changes in requirements treated as a competitive advantage. The agile projects used practices such as iterations, user stories, version control, refactoring and continuous integration (AGILE ALLIANCE, 2016). Customers and development team did not work together daily throughout the project. However, the team pointed out that the participation of the customer was adequate and did not affect the team.

b) The Story Line

The requirements were detailed and coded iteratively and incrementally in each sprint, according to the priorities defined by the customer. In each sprint, the coding activity was started when the first requirement was detailed. There was no need to wait for the detailing of all sprint requirements. The SRS had the description of US, rules, mockups, and test cases. This description was fragmented in various artifacts. The developer needed to consult several artifacts to have a complete understanding of a requirement. The analyst had to maintain the consistency of a requirement in various artifacts, as follows:

> *"We spend a lot of time to prepare and maintain the consistency among many artifacts. The ideal would be to have fewer artifacts."*

The SRS had technical aspects identified as positive factors, such as validation and interface rules, error messages, mapping source-destination database and default content of some fields on the forms. Acceptance criteria (AC) and NFR also were considered important for developers. However, they pointed out that the AC should be defined inside the US. The AC were not enough because they were very focused on the business rules, and did not address the technical aspects, as follows:

> *"It would be much easier to code and testing if the acceptance criteria were better defined and did not cover only business rules."*

The developers pointed out that the use of examples could help to clarify understanding of complex rules, as follows:

> *"In the specification of some rules, it is more appropriate to use examples where* [we can] *already see clearly all possible values of an attribute. When this is done only through a simple description, it is more likely to fail due to not restrict some value that should not be allowed."*

Although software engineers had many years of experience with the traditional software development, they had little experience with ASD. Often the requirements were not written clearly. Many systems analysts had difficulty in defining what was relevant to include in the SRS. Some artifacts had excessive details and redundant information which made it difficult to understand, as reported by some developers:

"The analysts have no experience to specify in an agile way. Although the analysts are experienced, some agile practices have not yet been absorbed. They have no ability to simplify and define what is relevant to the developer."

"Artifacts with confusing structure and repetitive information entail low productivity for the developer. I should not spend time trying to understand what the specification means."

The process to release a version for the customer required five different environments (local, development, testing, approval and production). The software validations were face-to-face, but the customer was located in another city. So, part of the team needed to travel to validate partial versions with the customer. In this context, the company considered that it was not worth to conduct the software validations at the end of each sprint. Usually, a version released to the customer validation consolidated the requirements of two or three sprints. At shorter intervals, validation of requirements was made through the SRS. So, the SRS was focused on the customer in order to facilitate understanding (particularly in requirements validation).

The software validations usually entailed many changes, refactoring, and retest. Some changes in the database structure had a great impact to the development team, and it was necessary to make a formal request justifying the desired change. This request was analyzed by DataBase Administrator (DBA). If approved, the change was made by the DBA according to the queue of requests for all projects of the company. This process distances the development from the agile principles and entails a bottleneck in the project schedule.

Requirements artifacts were elaborated in *Caliber*[7], a specialized tool for the requirements activities. The relationship between the requirements was generated automatically by the tool, which helped on the traceability of the requirements and in the impact analysis of change requests. Other tools like *Mantis*[8] and *Trello*[9] were used for project management, bug-tracking, and communication. Some developers pointed out the need for integration between the tools. *Caliber* also controlled the history of the changes made in the artifacts. The teams had difficulty in maintaining the SRS updated due to the large number of artifacts that had to be maintained. Moreo-

[7] https://www.microfocus.com/products/requirements-management/caliber/

[8] https://www.mantisbt.org/

[9] https://trello.com/

ver, the large number of changes had a negative impact on team motivation and in the reliability of the SRS, as described as follows:

> *"Many changes make the work tiring and cause dismay in the team because they result in refactoring and work being thrown away."*
>
> *"The lack of update causes the documentation to become unreliable."*

The contracts established that the payment was made by functionality delivered according to the number of function points, despite having a pre-defined global value. The customer only paid for functionality that he/she had requested. So, the company had a great concern for not wasting time specifying and developing features that would not be paid. Customers were also Brazilian public institutions and they followed the laws of the public sector. The contracts agreements established that all the artifacts should be approved by the customer.

The SRS was important to transfer knowledge when new engineers were allocated to the development teams and also for the teams responsible for the maintenance after the end of development. There was a lot of turnover of people between the teams, especially when a project was finished or when more resources were needed to meet the schedule.

Some disagreements between the software engineers (developers and analysts) were found. One of the differences concerns the level of detail for the SRS. All developers pointed out that an objective SRS, without prolix details, facilitates the understanding and improves the productivity. However, some analysts considered that the SRS should be as detailed as possible. Certain characteristics of the process can explain these disagreements. First, the SRS was used to validate requirements with the customer. Second, the SRS was provided to the customer as set out in the contract. Moreover, the traditional culture of development was still present in some people who could not think "agile" and they believed that a more detailed SRS would be better for the customer.

We also identified differences of opinion on the best way to perform the acceptance tests. Some software engineers advocated that the functional tests must be conducted from the SRS without using another particular artifact, but others software engineers did not agree. The justification given was that the tests were carried out by a specialized team with a different profile and that the SRS did not have enough coverage to conduct the tests, which would limit the test results.

c) Summary

Figure 4.2 summarizes the factors that affect the quality of SRS in this case study. The company opted for not validating the versions at the end of each sprint due to some *Characteristics of the Process* which establish a large number of environments to release a version and due to the trips required to do the face-to-face validations. But, the *Late Validation by Software* has led to *Excessive Changes* in the requirements. The SRS was *Customer-Oriented* because it was used to perform intermediate validations, as established in the *Contract Agreements* and due to the late validations through software.

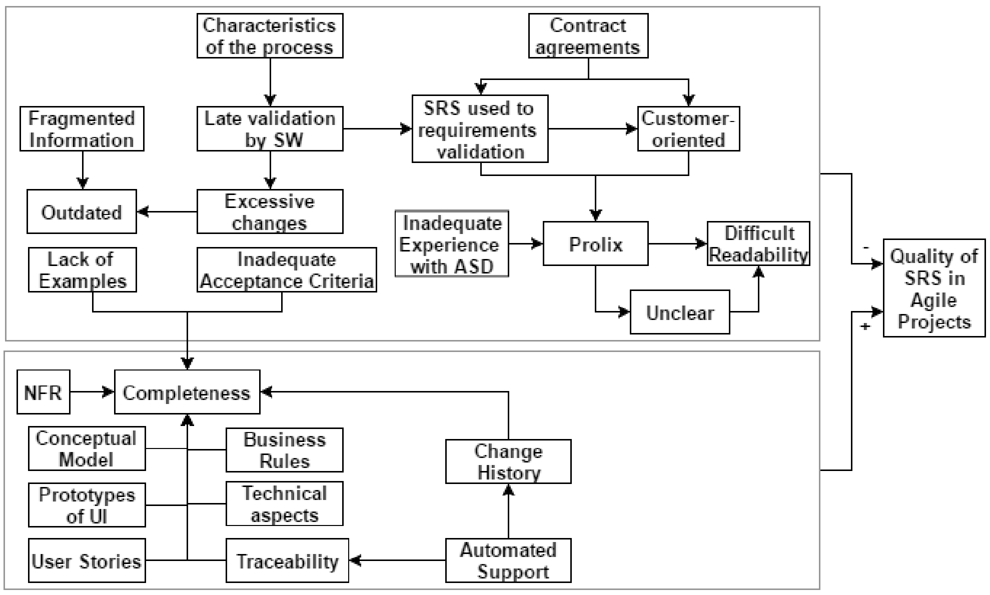


Figure 4.2 Quality Factors of SRS in ASD – case 1

The *Inadequate team experience with ASD* contributed to *Prolix* descriptions that difficult the clearness and *Readability* for the developers. *User stories*, *mockups*, *conceptual model*, *Technical aspects* and *RNF* were positive factors, although described in a *Fragmented* manner, without being related to the acceptance criteria and without examples. A considerable effort was required to create and maintain a large amount of artifacts, making it often *Outdated*. A specialized tool provided the requirements *Traceability* and the *Change History*.

4.3.2 *Case Study 2 (a mature private organization)*

a) Context

The study was conducted in a private software organization in Brazil. The company develops solutions for capturing, processing and management of Electronic

Funds Transfer (EFT) with a focus on banks, telephone operators, and commercial establishments that perform sales with credit and debit cards. The company has an application framework for this area. A team of 46 software engineers included new features in these applications as well as develop new software for the framework.

There was a great diversity in the profile of employees. Regarding the academic formation, the vast majority had only an undergraduate degree, but there were some masters and undergraduate students (trainees). Some engineers had only a few months of experience with software development, while others had more than 20 years. Considering only the software engineers, the average number of years of professional experience was 6 years. Except for two project coordinators, all fourteen interviewees worked as developers. Some of these also played the analyst role.

The company has over 19 years of existence, and it was certified in Mps.Br (2012) and CMMI (2010). Agile development was introduced using Scrum around 2008. Currently, the company has used agile practices like user stories, story splitting, iterations, continuous integration, code reviews, version control, daily meetings and automated builds (AGILE ALLIANCE, 2016). Each sprint lasted from 2 to 4 weeks. The customer validations were made face-to-face or remotely, depending on customer needs. The company operates a payment system for fixed price. The customer paid after the software validation in the productive environment.

b) The Story Line

The company adopted different strategies to specify requirements depending on the type and size of the project. For complex projects lasting more than a year, the functionalities, AC, mockups, field restrictions and execution flows were described in a requirements document. This artifact was used to validate the requirements with the customer and was also used by the development and test teams. The requirements document was not elaborated in the small and simple projects which had a maximum of six months duration. In these projects, the team wrote the user stories directly from conversations with the customer or the internal business area. A requirement was divided into multiple US, and if necessary, the developers broke down into tasks that were registered in *Redmine*[10], a project management tool.

[10] http://www.redmine.org/

The company had short sprints, but, in large projects, the software validations were not so frequent, occurring every two or three months. The type of software required a certification process carried out by external bodies in agreement with Brazilian law. At the end of each sprint, a specialized team performed internal validations. Next, the partial versions were certified by the customer or by third-party companies. Then, the versions were released on a homologation environment to verify their integrity and compatibility with other systems. Given the maturity of the framework, the customer validations resulted in just a few change requests. In this context, the company considered that it was not worth performing validations in shorter periods due to the expensive certification process required.

Software engineers pointed out some difficulties in the use of the same document to validate the requirements with the customer and to develop the software. The language used to describe the requirements was not targeted to the developers, leaving the SRS with extensive texts and a lot of unnecessary information, as follows:

> *"The SRS uses long texts, which most often leads to misinterpretation. It is necessary to write the SRS in a more objective way."*

> *"Obvious things are repeated, and practically a dissertation is written to make the text more presentable to the customer."*

The developers pointed out that the lack of examples of error messages and iteration with the user in the SRS delays the coding activities, as follows:

> *"It would facilitate to the developer if the SRS had examples of messages of error and request/response with the user."*

Objective and unambiguous descriptions were identified as a key factor to avoid waste of time trying to understand the SRS, as described as follows:

> *"The developer should not waste time interpreting the text. We should have clear requirements, simple, and direct."*

The company adopted other artifacts beyond the requirements document, such as user stories, architecture document and test scripts. Technical descriptions also were described, such as tables and fields, and log operations to be performed. But these artifacts were not available to the customer. They were only used by the development team. However, the developers preferred a consolidated SRS instead of wasting time trying to locate the information in many artifacts:

> *"It is unproductive having to switch between artifacts. It is easier to find what you need in a few artifacts."*

"We have information spread in various artifacts. It would be more productive to have the most consolidated information."

The use of various artifacts also hindered the update of the SRS. Often the SRS was prepared in parallel with the coding activity. It also contributed to the SRS become outdated. In the opinion of all developers, inadequate experience of people in carrying out the requirements activities was a factor which affected the quality of the SRS. The company did not use a specialized tool for the specification and management of requirements. All artifacts were constructed in a general-purpose word processor (like MS Word) which brought many problems. The most experienced developers pointed out that a specialized tool could help improve the team productivity, the readability, and the traceability of the requirements, as described as follows:

"We need to automate how the revisions of artifacts are done. Some tools could help to control the change history and the readability."

"An automated support facilitates the tracking of information when changes occur in the requirements."

The company did not keep information on the dependency relationships between requirements. When there was a change request in the requirements, the team met to discuss how the change would be incorporated and the associated impact. The consistency of the SRS did not appear to be something that worries the developers. Eventually, if a problem was identified, it was cleared up quickly by communicating with the person who had knowledge of the requirement. Software engineers did not consider that it compromised in their productivity.

The face-to-face communication was an established practice. Every day, the teams met to track the progress of activities, discuss problems, and sharing best practices. The customers were not integrated with the development team. The communication with them was not frequent, but all interviewees consider that it was appropriate, and the quality of SRS was not compromised. Communication with the customer was on demand, whenever the team needed clarifying any doubts.

c) Summary

Figure 4.3 summarizes the factors that affect the quality of SRS. Given the stability of the applications framework and the *characteristics of software*, the company adopted a strategy to delay the validation of the partial versions. The *SRS was used to validate requirements*. The *customer-oriented* SRS and the *inadequate expe-*

rience of the team entailed an SRS with unnecessary details that affected the clarity and *readability*. US, AC, conceptual model, mockups, technical aspects, and NFR were positive factors. However, this information was fragmented into various artifacts which made it difficult to understand and update. The lack of examples, traceability and change history, and the inadequate automated support were negative factors.

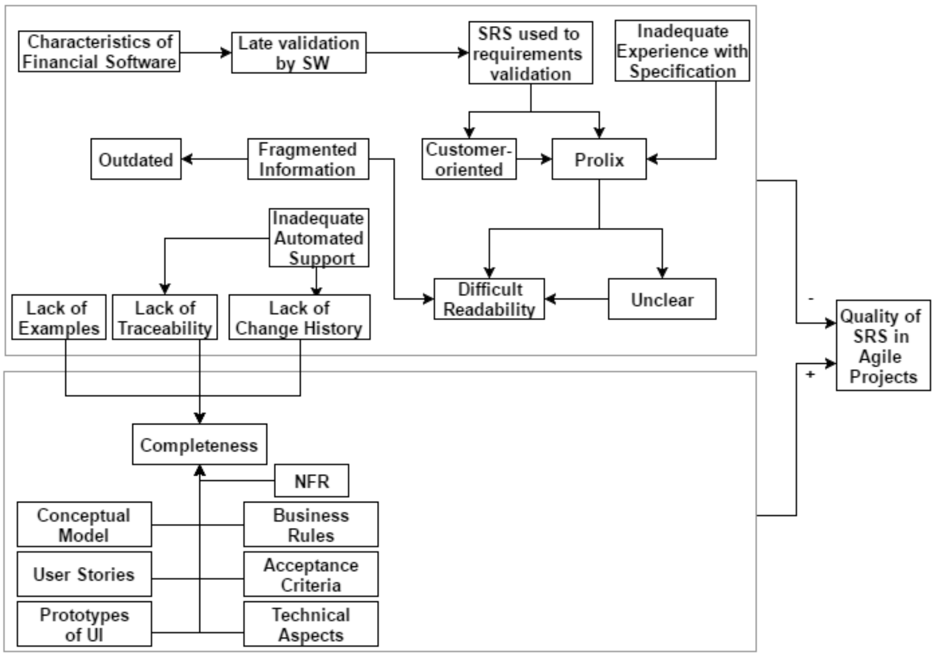


Figure 4.3 Quality Factors of SRS in ASD – case 2

4.3.3 *Case Study 3 (A young institute of research and innovation)*

a) Context

The case study was conducted in an innovation institute in Brazil. The institute has a year of existence, with focus on value creation and competitive advantage for large industrial companies. The customer portfolio has more than ten multinational and national companies from various sectors, such as smartphones, entertainment, health, and safety.

The institute has 155 employees among graduates, specialists, Masters, and Ph.Ds. The average years of experience with professional software development of the interviewees were 4 years. The case study was conducted in projects to develop mobile applications for 3 different customers, all smartphone manufacturers. In these projects, the payment system was not based on delivered functionality. There was a pre-agreed value, and fixed payments were made throughout of the development.

Since the foundation of the institute the ASD has been adopted using Scrum. Software engineers had experience with ASD. Several agile practices were well established, such as backlog, frequent prioritization of requirements, iterations, daily meeting, kanban board, planning poker, peer review, version control, continuous integration and automated build. The sprints lasted for 1 to 2 weeks. The partial versions were released at the end of each sprint to be validated remotely by the customer.

The institute was organized in development units and the projects were grouped by customer. The units had autonomy to adopt procedures, practices, and tools according to the needs of each customer. Many of the institute's customers were competitor players in the smartphones industry. To ensure customers' confidentiality, the institute had separate development teams to serve each customer. Thus, a team had no knowledge of the features that were being developed in teams serving other customers. Even the internal tests were performed privately for each customer. Testers of a project did not perform tests on applications of other customers.

b) The Story Line

The SRS was based on use cases, mockups, business rules and NFR. The SRS was not used to validate requirements, but it was more focused on the client because the artifacts were provided to him/her, as set out in the contract. The team pointed out that the SRS needed more technical information, for example, examples of interface constraints. The productivity was undermined because the SRS was not enough to the developer, as described as follows:

> *"The SRS lacks details. For example, in mobile apps, when the user enters in a screen, should the keyboard appear automatically or only when you select a text edit field?"*

The institute had a team of designers responsible for the specification of visual requirements. Due to the lack of the designers' experience with the requirements specification activity, the SRS was not clear, which made it difficult to understand, as follows:

> *"The SRS is not clear. It has ambiguities, and the functionality is not adequately detailed. The SRS could be more understandable. I believe that this is due to lack of time and the lack of designers' experience."*

Customers and development team were located in widely separated cities. The Institute maintained a continuous schedule of meetings to encourage greater iteration with the clients. In general, the features were requested by email, but it also occurred through video conferences or telephone calls. The versions were released remotely at the end of each sprint and the results of validations were sent by email to the team. The low availability of the customer impaired the detailing of the features, the clarification of doubts and the validation of the partial versions, as follows:

> *"The customer feedback is not adequate and affects the understanding of the needs and the direction of the project. Usually, take time to analyze the doubts that arise during the development cycle."*

The type of software developed affected the requirements activities. The mobile applications had many innovations features. The customers delegated to development teams the identification of the innovation aspects and the requirements of user interfaces. Some characteristics of mobile apps and inadequate availability of the customer entailed a lot of changes in the requirements, as described as follows:

> *"The mobile applications change the interface too often. Much effort is required."*

> *"Communication with the customer is not always as frequent as it should, which makes several changes to be postponed to the next sprints."*

The applications had a limited use of database management systems (DBMS). Some projects used NoSQL database and others used SQLite. The developers had autonomy to change the database structure. However, wrong changes entailed an enormous impact on the productivity because they implied evolving the source code accordingly, and then to repeat the test cycles.

To analyze the change requests, first, the internal PO and the team assessed the impact of changes. Then, a meeting was made with the customer to align the changes and sometimes to reevaluate the scope. Usually, the duration of a sprint was not modified due to a change request. It only happened in exceptional cases, if it was not possible to relocate some functionality for a future sprint. Just one project had information on the dependency relationships among the requirements. The remaining projects had no traceability of the requirements to support the analysis, but this absence was not pointed out as a negative factor. Given that software deliveries were frequent, the impact of changes was minimized, but, they affected the team performance and the project management. The updating of the user interface requirements was made by designers who usually worked on several projects at the same time. Overloading of the designers affected the update of the SRS.

The projects used different strategies for testing. In some projects, the tests were conducted directly from the use cases. In other projects, the functional, unit, load, and stress tests were automated. In these projects, another artifact was drawn up from the use cases given that the SRS was insufficient to automate the tests. None of the projects adopted strategies of TDD, such as ATDD or BDD. There were no acceptance criteria in these projects. In all projects, the team pointed out that it was necessary to enhance the SRS to facilitate the test.

Software engineers pointed out that the structure based on few artifacts was essential because the developer did not waste much time to find information, and it was easier to update the SRS and avoid inconsistencies, as described as follows:

> *"The SRS with few artifacts helps the team, because we have fewer sources and therefore less chance of inconsistencies."*

> *"A reduced SRS favors productivity because the team does not waste a lot of time searching information and updating various artifacts."*

The company did not use a specialized tool for requirements activities. Some projects used the *Tuleap* tool[11] for project management, and others used the *Taiga* tool[12]. However, software engineers did not identify the need for a specialized tool to improve the quality of the SRS. However, some engineers pointed out the need to have better control over the changes in the requirements. None of the engineers had previously worked on projects that used any specialized tool for requirements.

c) Summary

Figure 4.4 summarizes the factors that affect the quality. The functional requirements were represented by *use cases* and *prototypes of UI*. *NFR* also were described. However, the SRS was not intended for the developer given that the artifacts were provided to the customers, as set out in the *contract agreements*. The superficial details of the functional requirements, *the lack of description of technical aspects*, *acceptance criteria* and *examples* have been identified as negative factors.

The small customer availability compromised the breakdown of the requirements, the clarification of doubts and the validation of the partial versions. The inadequate customer collaboration and innovation aspects of mobile applications contributed to a high number of changing requirements. The lack of control of changes in the requirements impaired the SRS. Sometimes, the SRS was *outdated* due to fre-

[11] https://www.tuleap.org/

[12] https://taiga.io/

quent changes. The small experience of the team also impaired the detailing of the requirements, the clarity and the *readability* of SRS.

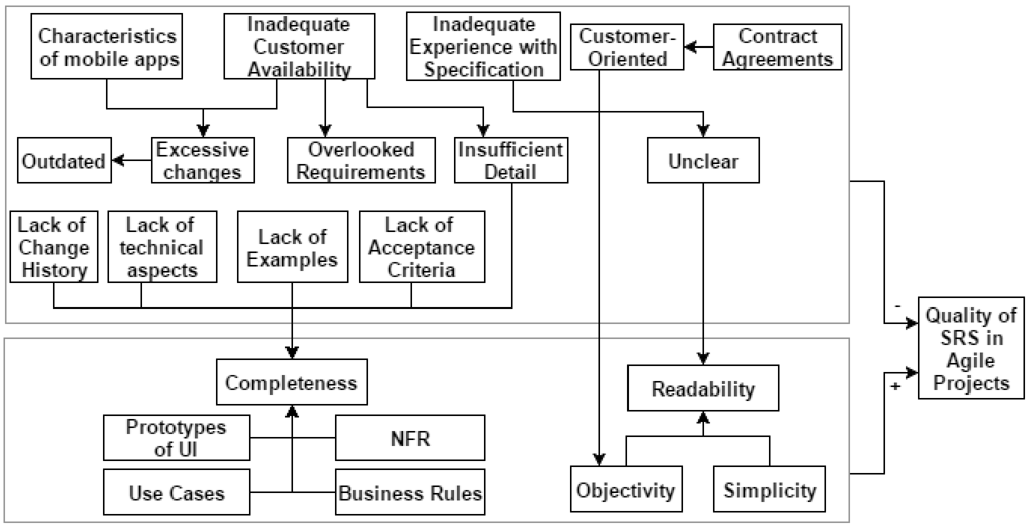


Figure 4.4 Quality Factors of SRS in ASD – case 3

4.3.4 *Case Study 4 (a micro software company)*

a) Context

This case study was conducted in a software micro-company of England that operates in the education sector. The company had a small team composed of only 7 software engineers. The company has been using ASD since its founding in 2011 and has adopted agile practices, such as iteration, frequent releases, daily meetings, user stories, retrospective, pair programming and continuous integration. Over the past 5 years, the company has developed an integrated solution that is operating in over a dozen educational institutions in Europe. The team worked only in the evolution of this application. New versions were released to evolve the functionalities or to fix critical bugs. No other application was being developed. Sprints lasted a week.

The company used a fixed monthly payment system based on the number of user licenses with unlimited access to application functionality. Customer validations were made remotely. The entire company worked remotely. Software engineers worked in multiple different locations. All communication was done remotely, either with employees or with customers when necessary. Every day, with a fixed schedule, the development team used virtual meetings to monitor the project status.

b) The Story Line

The company used a reduced SRS based on user stories and mockups. Sometimes scenarios were described in the US. The description of the AC within the user story facilitated the coding and testing. The SRS was directed to the team. It was written by and for the developer in a very clear and objective way. The company did not use the SRS to validate requirements. Validations were conducted through frequent software deliveries. The software processes were frequently changing, and the company was aligned with the agile value which states that individuals and interactions are more important than processes and tools, as follows:

> *“We do not write anything "in stone". Our processes often change to adapt to the current culture and strategic planning of the quarter.”*

The CEO played the role of PO and also participated in the coding activity. The stories were written from conversations between developers and PO. All team members played the analyst and developer roles simultaneously. The principle of simplicity guided the company. The team had the culture of writing the stories as simple as possible, containing only the necessary to code, and without ambiguities. They had autonomy to refine or split stories to facilitate coding, as follows:

> *“Extensive and bureaucratic SRS hinder rather than help. The stories are simple enough to facilitate coding, allowing a feature can be delivered as soon as possible. We use the "divide and conquer" approach. The smaller a story, the higher the success rate.”*

The company did not use a specialized tool for supporting requirements-related activities, but this was not pointed out as a negative factor. The *Asana* tool[13] was used to track the team tasks and to write the US. The company has a well-defined scope of operation based on the evolution of a stable application. In this context, some agile practices did not make much sense for them, such as developers and customers working together daily. Due to the type of development, it was not necessary to have daily communication with the customer. The developers had little contact with the end user. The PO made the bridge with the customer. The discovery sessions were conducted to understand the client's goals and requirements of the business. Only the PO participated in the discovery sessions. Rarely, the developer needed to meet with the customer to code a particular feature. The low frequency of communication was not considered inappropriate and did not impair the SRS.

[13] https://asana.com/

The company's development process did not address the changes as a competitive advantage for the customer, as set out in agile manifesto. The company avoided making specific customizations for a particular customer. These were done only in exceptional cases. The company did not maintain information on the relationships among the requirements. However, this absence was not critical given that the software development was just to evolve a stable application. Thus, changes were neither structural nor frequent.

The minimal SRS directed to the team contributed to it to stay updated, most of the time. In each sprint, before coding activity, the software engineers performed a design activity to analyze the best solution to coding the requirements. The average of years of experience of the developers with coding was high, but the novice developers had little experience with ASD and with the specification of the requirements which entailed in prolix descriptions that hindered the clarity and legibility of the SRS, sometimes. Given the nature of development restricted to evolve one mature software, the description of NFR and the conceptual models were not used in the current stage. Also, some technical descriptions were delegated to each developer, such as the content to fill the combobox widgets, necessary validations between some fields, and secondary operations to persist the data in the database. The lack of such descriptions in the SRS undermined the coding activity for the novices.

c) Summary

Figure 4.5 summarizes the factors that affected the SRS. Despite being operated entirely remotely, the company used a minimal SRS to evolve a mature application. The SRS was objective, unambiguous and focused on the development team which improved readability for it. Validations were made through frequent releases. The SRS was not used to validate requirements.

The functional requirements were represented by mockups, US, and AC. The lack of technical aspects was a negative factor. The company adopted a strategy of avoiding customizations to meet specific demands of a customer. The company did not maintain information on the traceability of requirements, such information was required, but only at the strategic level. The small experience of novice developers led to prolix descriptions, sometimes.

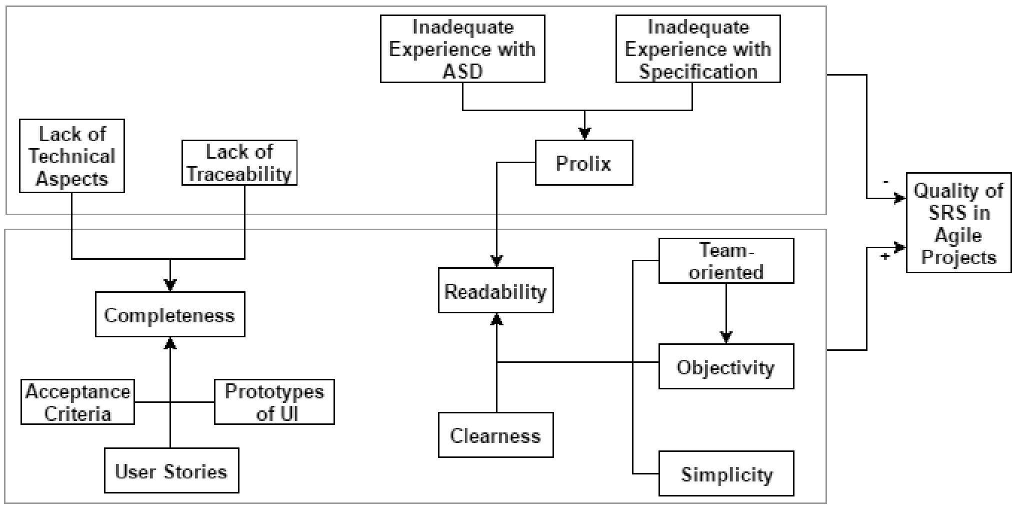


Figure 4.5 Quality Factors of SRS in ASD – case 4

4.3.5 *Case Study 5 (A mature and very large software company)*

a) Context

The case study was performed with fifteen software engineers in a very large company that had more than 40,000 employees and operated in more than 100 countries. The company had over 20 years of existence and offered solutions for various sectors, such as public administration, security, transport, energy, and financial services. The case study was carried out with software engineers who worked on two agile projects in a development unit located in one of the states of Brazil. In this unit, the ASD was introduced just 3 years ago.

The average experience of interviewees was five years. The development process used XP and Scrum practices, such as iterations, frequent releases, kanban board, standup meetings, refactoring, and pair programming. The sprints lasted from 2 to 4 weeks. Some agile principles were well established, such as frequent communication with the customer and acceptance of change as a competitive advantage. The customer did not work on the same site of the team, but communication was frequent using the *Slack* tool[14], email or phone.

b) The Story Line

The SRS was based on use cases, business rules, mockups and conceptual model. In some projects, the use cases were broken down into user stories. The acceptance criteria were defined in another artifact that aimed to describe the quality

[14] https://slack.com/

metrics of the source code. NFR were described in the architecture document. The SRS was not used to validate requirements. Validations were made through frequent software deliveries.

The little experience of analysts entailed prolix and unnecessary texts to the developer. The specification and coding activities were not performed by the same person; there was a distinction between analysts and developers. Many analysts did not have the proper perception of what was necessary to the developer, as described as follows:

> *“The SRS is prolix due to the lack of experience of the analysts. I think they have never worked as a developer. There is a lot of information directed to the customers that does not add value to the developer. There are business rules that are written with large texts, but often only one paragraph is enough.”*

The information conveyed by the SRS was considered insufficient for supporting the implementation. Developers pointed out that the technical information, such as validation rules, tables, and fields were not described in the SRS that was considered insufficient for developers, as described as follows:

> *“Often, the SRS is not sufficient. It is incomplete leaving many doubts, making it more complicated the coding activity.”*

The development unit was exclusively directed to develop software for public administration. Customers followed the laws of the Brazilian public sector. The contract required some specific artifacts, and it established that all SRSs had to be provided to the customer. The projects produced many artifacts to address different stakeholders. So, the SRS was intended for the customer. The contract also established a payment system by product delivered. In addition to the incremental versions in each sprint, the SRS was also considered a software product for payment purposes and had a different pricing.

The company had a specialized department for testing. The testers considered that the acceptance criteria were insufficient, so another artifact was prepared (test cases). At the end of each sprint, the partial versions were internally tested and then released for the client. There was not a consensus in the team about the best way to structure the SRS. Analysts and testers considered that the SRS divided into various artifacts was more appropriate because each artifact had its characteristics. But, developers pointed out that a consolidated view facilitated the general understanding of the requirements, and it was easier to find information, as described as follows:

> *"The description of the requirements had different views, so, the structuring in various artifacts is more appropriate than consolidated."*

> *"A consolidated documentation is better for the developer because it makes it easier to find the information."*

The SRS more focused on the customer and with unnecessary details for the developer impaired the clearness and understanding, as described as follows:

> *"The SRS contains gaps that make understanding it difficult. It should be more straightforward and direct. Many times the SRS is unclear, allowing different interpretations."*

The customer communication was appropriate. However, given that the SRS was insufficient to the developer, a strong communication was needed with the analysts to clarify doubts, so, the team productivity was impaired. Much information was obtained from the daily meeting, but it was not enough. During the internal tests, many of detected Non-Conformities (NC) could be traced back to a misunderstanding of the SRS. Developers' productivity was impaired because they often needed to fix these NC. The SRS was elaborated in MS Word. The developers pointed out that a specialized tool could help control and maintain the change history.

c) Summary

The Figure 4.6 depicts the factors that affect the SRS.

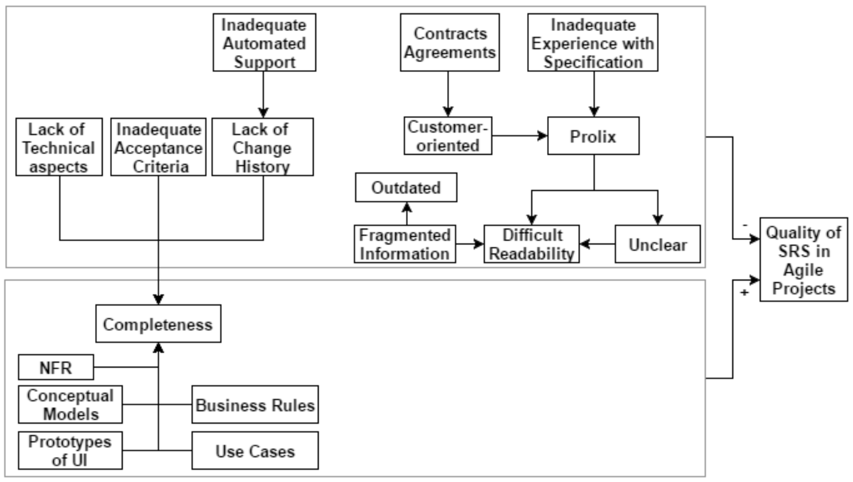


Figure 4.6 Quality Factors of SRS in ASD – case 5

The SRS was not used to validate requirements, but it was customer-oriented due to the contract. Sometimes, the inadequate experience of the analysts entailed unnecessary details that affected the SRS clarity and legibility. Functional requirements were represented by use cases, screens, and conceptual models. Despite describing the NFRs, the SRS did not explain technical aspects. The acceptance criteria were inadequate for code and tests activities. A lot of artifacts impaired the understanding of the developer. The absence of a specialized tool made it difficult to control changes in requirements.

4.3.6 *Case Study 6 (A company whose core business is not software)*

a) Context

The case study was conducted in a large private company that operates in the mortgage sector in the Netherlands for the past 20 years. The core business of the company is not software development. The customer and the end-users are employees of other sectors of the company. The development team had around 150 software engineers to develop applications only for internal use. The software engineers had many years of experience with the software development. One of them had more than 35 years of experience, and the others had more than 15. However, they had little experience with ASD. For many years, they had used traditional development (waterfall), but since the last 3 years, ASD was adopted in all projects. They adopted practices, such as kanban, backlog, user stories, planning poker, iterations, acceptance testing, code review, daily stand up, and frequent releases, among others.

b) The Story Line

The SRS was based on user stories, conceptual model, and business rules. Despite having a small SRS that facilitated the readability, the SRS was very focused on the customer, as follows:

> *"The SRS is written in a waterfall method while running in agile projects. Often the SRS is difficult to understand. Written for the end user, not for programmer."*

The SRS was insufficient and did not have the information required for coding. Software engineers pointed out the need to describe NFRs and also be able to translate the functional requirements into technical requirements, as described as described as follows:

> *"The functional requirements need to be translated into technical requirements and have a good division between functional and NFR."*

The software engineers pointed out that the acceptance criteria should be described in a more efficient way to decrease the effort to perform the tests. To achieve this, all requirements should be considered, not just the functional requirements, as follows:

> *"Create more understanding of our system with designers. Being smarter in acceptance criteria. An adequate SRS for the developer should contain acceptance criteria, functional and NFR."*

The little experience of the team and company with ASD impaired the simplicity and legibility of the SRS. Prolix descriptions unnecessary for the developer was a negative factor, as described as follows:

> *"The requirements specification must be as simple as possible and have a unique interpretation. It is necessary to standardize and write the SRS in a more objective way."*

The communication with the end user was done exclusively by the PO. Every two weeks, new versions of applications were available for the user validation. However, the users took a long time to validate the software, causing a bottleneck in the development process because the backlog of the sprints was open waiting for customer feedback, as described as follows:

> *"We lost sight of the stories. Sometimes, the user complains that a problem is not fixed, but actually, it is waiting for his validation. This costs time and rework."*

There was no standard for the testing activity. Some projects performed the tests directly from the SRS, but, other projects prepared another artifact for testing. In both cases, the tests were made in a non-automated way. The team suggested adopting TDD and automation to enhance the SRS and hence the tests, as follows:

> *"Testing would be better if there is a default test set (automated) in combination with test-driven development."*

Jira tool[15] was used to support project management. The company did not use any specialized tool to support the requirements activities. The stories also were described in *Jira*, but none of the interviewees pointed out that this affected the quality of the SRS.

The company developed new applications and had a constant activity of maintenance of the existing applications. There were some maintenance teams to add new features and fix bugs in these applications. Some older applications had no SRS. The business rules were only in the minds of some people. Sometimes, the

[15] http://technologyadvice.com/products/atlassian-jira-reviews/

software engineers wasted a lot of time to find relevant information. Other times, there was a lot of unnecessary information, as follows:

> *"There's a lot of knowledge not written on paper, but it is in the heads of colleagues."*
>
> *"Often, old projects don't have a relevant SRS. We lose a lot of time when we need to maintain these projects."*

c) Summary

Figure 4.7 summarizes the factors that affected the SRS. Despite the frequent software releases, the end users took a long time to validate these releases, which caused problems in the requirements management. The SRS was not used to validate requirements. The company and team had little experience with ASD which affected the simplicity of SRS. The SRS were very focused on the end user. It had unnecessary details for the developer (prolix), and it lacked technical aspects and NFRs. The acceptance criteria were not described properly.

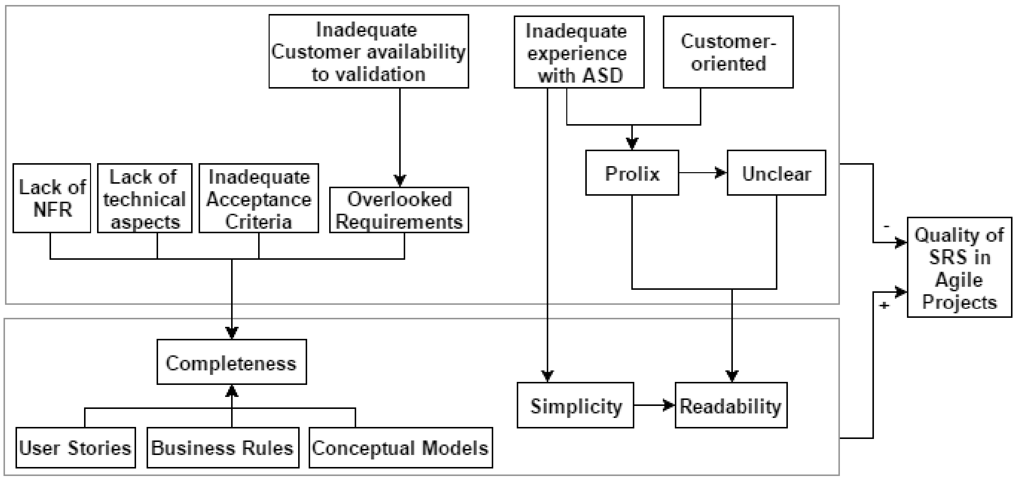

Inadequate Customer availability to validation
Inadequate experience with ASD
Customer-oriented
Prolix
Unclear
Lack of NFR
Lack of technical aspects
Inadequate Acceptance Criteria
Overlooked Requirements
-
+
Quality of SRS in Agile Projects
Completeness
Simplicity
Readability
User Stories
Business Rules
Conceptual Models

Figure 4.7 Quality Factors of SRS in ASD – case 6

4.4 Summary

In this chapter, we present the procedures performed in six companies to investigate the phenomenon of the specification of requirements in ASD. The individual results of each study are presented, describing the factors that affect the quality of SRS in each company, and how the factors affect the work of the development team.

In next chapter, we answered the RQ2 (defined in Section 1.2) by presenting the model that emerged from the cross-case synthesis of the data collected in the six companies. Threats to the validity of case studies are also presented in Chapter 5.

5
A MODEL ABOUT QUALITY OF SRS IN AGILE PROJECTS

This chapter presents the model emerged from the investigation into the industrial practice described in Chapter 4. Section 5.1 describes the steps to conduct the cross-case analysis and synthesis. The results of the cross-analysis of the data are described in Sections 5.2, 5.3 and 5.4. The model resulting from the data synthesis is presented in Section 5.5. In Section 5.6, the model is compared to the SM and others studies. The findings of the cross-case synthesis are discussed in Section 5.7.

5.1 Procedures for cross-case analysis and synthesis

A cross-case analysis was conducted to achieve synthesis through subsuming the concepts identified in case studies. The idea behind the searching for cross-case patterns is to force the investigators to go beyond initial impressions, in particular through the use of structured and diverse lenses on the data (EISENHARDT, 1989). The key to good cross-case comparisons is to look at the data in many divergent ways. This improves the likelihood of an accurate and reliable model. Cross-case analysis enhances the probability of capturing novel findings which may already exist in the data. This is also consistent with our goal of building a model about the phenomenon that explains similarities and contradictions among different cases.

Models are used to facilitate a clear understanding of phenomena. They are one of the main instruments of modern science (FRIGG and HARTMANN, 2012). A model is a physical, symbolical, or verbal representation of a concept, which constitutes an appropriate scientific mechanism to guide the search and the explanation of facts. These representations are useful because they help organizing and narrowing down the scope of the phenomena, predicting new facts and relationships based on previously known facts and relationships, and identifying facts that have not been convincingly explained (FRANÇA et al., 2014). A solid model of quality of SRS in the context of the ASD would represent a relevant and timely contribution.

To build the model, we followed the guidelines proposed by Eisenhardt (1989) to build theories: Getting started; Selecting cases; Crafting instruments\protocols; Entering the field; Analyzing data; Shaping the theory; Enfolding literature. The first four activities were described in Chapter 4. This chapter describes the other activities.

Procedures of meta-ethnography (NOBLIT and HARE, 1988) were used to translate the concepts and propositions of the case studies during the cross-case synthesis. Meta-ethnography is an approach that enables a rigorous procedure for deriving substantive interpretations about any set of constructive studies. It involves some degree of induction and interpretation, which is consistent with the nature of the qualitative case studies that we want to analyze and synthesize (FRANÇA et al., 2014).

As shown in Figure 5.1, initially, we reviewed each case to understand the concepts and their relationships. Then, we searched for cross-case patterns interpreting and summarizing the core similarities and differences between the studies in order to extend our understanding of the phenomenon in different contexts, and to explain it. Next, we translated the concepts and propositions from individual case studies. Finally, we build a model considering the concepts and propositions, and compared it to the others studies in the literature.

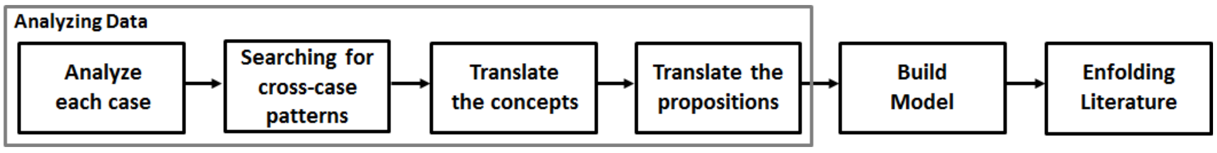


Figure 5.1 Cross-case analysis and synthesis steps

5.2 Searching for cross-case patterns

To have a maximum variation sampling strategy, different companies were chosen, as described in detail in Chapter 4 and summarized in Table 5.1. Public and private companies of different sizes were investigated. Young companies with only one year of existence and enterprises with more than 40 years of experience in software development. Three of the companies are headquartered in Brazil and provide services to clients located in it. The other 3 companies have headquarters in Europe, but one of them has a branch in Brazil (fifth case study - cs5).

Table 5.1 – Context of the companies

Characteristics	cs1	cs2	cs3	cs4	cs5	cs6
Years of Existence	42	19	1	5	23	19
Size	Large	Small	Medium	Micro	Very Large	Large
Area of Operation	Brazil	Brazil	Brazil	Europe	+100 countries	Netherlands
Nature of Sector	Public	Private	Public	Private	Private	Private
Type of Customer	Public	Private	Private	Private	Public	Private
Core business	Solutions for Government	Solutions for EFT	Innovation Solutions for various sectors	Solutions for education sector	Solutions for various sectors	Solutions for mortgage sector
Nature of Products	Information Systems and BI	Applications for mobile devices	Applications for smartphones	Information Systems	Information Systems	Information Systems and BI
Number of Employees						
Entire Company	3800	71	155	12	+40000	600
Unit investigated	200	71	155	12	+100	600
Software Engineers	183	46	135	7	90	150
Case Study Participants	26	20	25	7	15	10
Interviewees	14	14	15	3	9	4
Average team experience	11 years	6 years	4 years	8 years	5 years	21 years
Experience with ASD	2 years	8 years	Since its foundation	Since its foundation	2,5 years	3 years
Duration of sprints	4 weeks	2/4 weeks	1/2 weeks	1 week	2/4 weeks	2 weeks
Client communication	Product-Owner	Product-Owner	Project Manager	Product-Owner	Team	Product-Owner
SRS based on	User Stories	Use Stories	Use Cases	Use Stories	Use Cases	Use Stories
Requirements Validation						
By SRS	Always	In most projects	Not used	Not used	Not used	Not used
By Software Running	Every 2/3 sprints	Every 2/3 sprints	Every sprint	Every sprint	Every sprint	Not often (depend on the user)
Type	Face-to-face	Face-to-face and Remote	Remote	Remote	Face-to-face	Face-to-face and Remote
SRS directed to	Customer	Customer	Customer	Team	Customer	Customer
Payment System	Value-based	Fixed price	Fixed price	User-based	Value-based	Not applicable

The companies operate in different business segments. In most of them, the software development is the primary activity of the company, except one that works in the mortgage market (cs6). In all companies, the customer was located in different cities of the development team. Software validations were done face-to-face or remotely. In one of the companies, the development team works completely distributed (cs4). All companies adopt short sprints, ranging from 1 to 4 weeks. However, the frequency of software validation was different.

We interviewed software engineers with university graduation, specialization, master, as well as graduate students. In three companies, most interviewees were male, and in one enterprise the majority was female. In two companies, all respondents were male. We interviewed software engineers who were acting as developer, system analyst, tester and solution architect. Although not being part of the analysis unit, some managers were also interviewed to clarify doubts about the context. The profile of software engineers is summarized in Appendix D (Table D.1).

5.2.1 *Similarities between the studies*

The collected data were analyzed to capture the similarities and discrepancies, and to find ways to explain them. Similarities in the findings of the case studies increase their reliability and external validity (FRANÇA et al., 2014). Despite the diversity in the context of companies and in the profile of the software engineers, we identified several similarities, which can be grouped into two categories, as shown in Figure 5.2: *Insufficient content for the developer* and *Difficulty of understanding the SRS due to how the requirements are described*. The difficulty of understanding is due to prolix texts, which are customer-oriented and unnecessary for the developer. Ambiguous descriptions and fragmented in various artifacts also undermine the understanding.

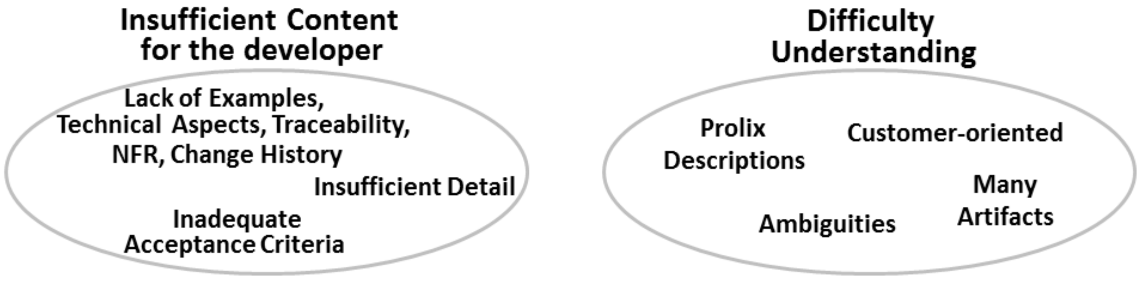

Insufficient Content for the developer
Lack of Examples, Technical Aspects, Traceability, NFR, Change History
Insufficient Detail
Inadequate Acceptance Criteria
Difficulty Understanding
Prolix Descriptions
Customer-oriented
Many Artifacts
Ambiguities

Figure 5.2 Overview of SRS quality factors in case studies investigated

Regarding the content, the functional requirements were described by US, use cases, conceptual model, and mockups. Regardless of the form of representation, the developers pointed out that much effort is required to clarify doubts when the SRS is insufficient and focus only on the functional requirements. The lack of NFRs, technical aspects such as constraints of design and interface, validation rules were pointed out as a negative factor. The lack of examples impaired the understanding of the complex rules. The history of changes and the dependency relationship between the requirements were important to analyze the impact of the change request. Sometimes, the content is impaired due to insufficient detail. The lack or inadequacy of acceptance criteria was also identified as a negative factor.

It is important to emphasize that in none of the companies the client worked in an integrated way on the same site with the development team. All software engineers pointed out that the customer's availability is important to detail the requirements, clarify doubts and validate the functionality, but it is not required the daily presence of the customer for this. The customer should be available when the team needs. In two of the companies, the customer availability was inadequate, and this was perceived as detrimental for the contents of the SRS and software validation.

Insufficient experience with ASD and with the activity of requirements specification affected the content and how the requirements were described. Some company used the SRS to validate requirements with the customer aiming to meet some contract clause or due to the impossibility of carrying out frequent validations through software. As a result, the SRS was customer-oriented, which made it inadequate for the developer. The SRS was insufficient for supporting coding because technical aspects and NFRs were not described; the focus was only on the functional requirements. Insufficient SRS for the developer and the difficulty in understanding it increased the effort to coding, testing, and maintaining. The SRS fragmented on many artifacts increased effort to specify the requirements and made it difficult to update.

The list of the factors identified in each case study is shown in Table 5.2. The factors that affected positively and improved the quality of SRS are represented as '+'. The factors that jeopardized the quality are represented as '–'.

Table 5.2 – Quality factors by case studies

Factor	Impact	Case Studies					
		1	2	3	4	5	6
«Customer-oriented»	-	✓	✓	✓		✓	✓
«Team-oriented»	+				✓		
«Automated Support»	+	✓					
«Inadequate Automated Support»	-		✓			✓	
«Fragmented Information»	-	✓	✓			✓	
«Simplicity»	+			✓	✓		✓
«Unclear»	-	✓	✓	✓		✓	✓
«Clearness»	+			✓			
«Difficult Readability»	-	✓	✓			✓	
«Readability»	+			✓	✓		✓
«Objectivity»	+			✓	✓		
«Prolix»	-	✓	✓		✓	✓	✓
«Completeness "agile"»	+	✓	✓	✓	✓	✓	✓
«Insufficient Detail»	-			✓			
«NFR»	+	✓	✓	✓		✓	
«Lack of NFR»	-						✓
«Technical aspects»	+	✓	✓				
«Lack of Technical aspects»	-			✓	✓	✓	✓
«Traceability»	+	✓					
«Lack of Traceability»	-		✓		✓		
«Change History»	+	✓					
«Lack of Change History»	-		✓	✓		✓	
«Acceptance Criteria»	+		✓		✓		
«Inadequate Acceptance Criteria»	-	✓				✓	✓
«Lack of Acceptance Criteria»	-			✓			
«Prototypes of User Interface»	+	✓	✓	✓	✓	✓	
«User Stories»	+	✓	✓		✓		✓
«Use Cases»	+			✓		✓	
«Business Rules»	+	✓	✓	✓		✓	✓
«Conceptual Model»	+	✓	✓			✓	✓
«Lack of Examples»	-	✓	✓	✓			
«Characteristics of mobile applications»	-			✓			
«Characteristics of financial software»	-		✓				
«Characteristics of Process»	-	✓					
«Inadequate Customer Collaboration»	-			✓			
«Inadequate Customer Collaboration to validate»	-						✓
«Inadequate Experience with ASD»	-	✓			✓		✓
«Inadequate Experience with Specification»	-		✓	✓	✓	✓	
«Late validation by SW»	-	✓	✓				
«Outdated».	-	✓	✓	✓		✓	
«Excessive changes»	-	✓		✓			
«SRS used to requirements validation»	-	✓	✓				
«Contract agreements»	-	✓		✓		✓	
«Overlooked Requirements»	-			✓			✓

(+) Positive effect, improve quality; (-) Negative effect, jeopardize quality

5.2.2 *Explaining the differences between the studies*

The diversity of the companies, processes and the type of developed software opens the opportunity for explanations based on the contextual differences, thus enriching the understanding of the phenomenon in highly distinct environments. The main differences between the case studies are briefly described as follows.

a) SRS used to validate requirements

Two companies used the SRS to validate requirements, instead of performing validations through frequent software deliveries. Some peculiarities explain the adoption of this strategy. In both companies, the effort required to make available a version to the customer is very high due to the characteristics of the process and of the software. The company of the second case study (cs2) develops software that involves EFT, so the software must be tested and certified by control bodies in Brazil. In the first case study (cs1), to provide a version for customer validation requires a prior validation process in the local, development, testing, and homologation environments. Also, the contract agreements establish that the SRS must be validated by the customer before the coding.

In both contexts, the companies consider that is not worth to validate versions at the end of each sprint. The versions are available only for internal testing. Thus, the SRS is used as a mechanism to perform intermediate validations with customers. In these companies, the validations of partial versions occurred every two/three months, at least. In the first case study, the late validations entailed many changes in applications. In the second case study, given that the development aimed to evolve a stable framework that was already working on several clients, late validation did not bring big troubles to the development team.

b) Maturity of the companies and Experience of the teams

The older companies (cs1, cs2, cs5, cs6) still have some vestiges of traditional development. Some artifacts were maintained to meet different stakeholders' needs. The exception is the company of the sixth case study: despite being over 19 years old and having three years adopting the ASD, it used a simple SRS with few artifacts. In this case, it is important to note that the end users and the customer were internal, given that the software developed by the company was for their own use.

The adoption of ASD is recent in companies. The interviewed software engineers pointed out problems with SRS described in a prolix way because of small experience of some engineers with the activity of requirements specification and with ASD. The company of the third case study did not identify problems arising from excessive detail, but, the lack of experience of the team entailed an SRS with ambiguity. Also, the low customer availability undermined the detailing of the features.

c) Billing system for software products

Companies used different billing systems: value-based, fixed-price and user-based. In the company of the sixth case study, the software was developed for internal use only, i.e. the software was not sold.

Two companies used a payment system based on delivered functionality (value-based), but they had some different characteristics. The company of the first case used the SRS to validate the requirements with the customer before coding, as set out in the contract. The customer only paid for the features requested by him/her and after the software validation. Thus, there was a great concern not to lose time specifying and developing features that were not requested by the customer. In the company of the fifth case, the SRS was not used to validate requirements with the customer, but it was considered a software product and the customer paid for it and for the functionality available in the software.

Two companies (cs2 and cs3) adopted a system of monthly payments of fixed price, regardless the functionality delivered in the sprint. Sometimes, features not requested directly by the customer were added to enhance the operation to it or to facilitate the inclusion of new requirements in the future. This practice is contrary to the agile principle of maximizing the amount of work not done, but it was an approach that aimed at improving customer satisfaction.

In the fourth study, the company uses a payment system based on the number of users of the application (user-based). In this case, the company avoided making customizations aimed to meet demands of a particular customer. The features were only included if they made sense for the evolution of the application as a product.

d) Sector of activity of companies and customers

Two companies are public institutions (cs1, cs3) and four are private. However, one of the public companies only had private customers (cs3), and one of the private companies was providing services only to public institutions (cs5). Public and private companies that provide services to public institutions were required to follow the sector legislation in Brazil. In these cases, the SRS was made available to the customer and regulatory agencies to meet the contractual agreements. Thus, the SRS was written in a language customer-oriented, and not to the team. Nevertheless, the private company that had public clients adopted a simple SRS (but customer-oriented), based on few artifacts, unlike public companies that used an extensive documentation.

e) Type of Software

Most companies were engaged in the development of information and management systems. Two companies developed applications for mobile devices. The applications for smartphones developed in the third case study had many innovation requirements. Many changes were requested after customer validation, which was pointed out as a negative factor by software engineers. According to them, the high number of changes occurred as a result of these characteristics of the software combined with the small customer availability to clarify doubts, detail the requirements and to validate the partial versions.

The company of the second case study works in the evolution of an application framework for capturing EFT. In the fourth case study, software engineers worked on the evolution of a single application focused on the education sector. Common to these two companies is the maturity of the applications that are already in operation in several clients for some years. Thus, few changes were requested when new features were available for customer validation. Changes in database structure were minimal and also had no significant impact on the development team.

Although only a few mobile applications of the third case study used database, these projects had many rework problems arising from change requests that involved in changes in database structure.

f) Requirements validation (face-to-face/remotely)

We found some differences regarding how the software validations were conducted. Two companies (cs1 and cs5) performed only validation face to face with the customer. In two other companies, the validations were conducted only remotely by the customers, without the participation of the development team (cs3 and cs4). Two companies performed validations face-to-face and remotely, depending on the needs of the project or stakeholders (cs2 and cs6).

In two of the four companies (cs3 and cs6) that perform validations remotely, the customer had limited availability to clarify doubts, detailing requirements or validating the software. Thus, the remote validations entailed problems for the project because the customer often took a long time to verify the partial versions. Although the quality of the SRS was not affected, this practice hindered the planning of the next sprints, because some requirements were forgotten without being validated by the customer, leaving the product backlog with uncompleted tasks, waiting for the customer feedback.

Enterprises that perform the validations face-to-face had no such problems because the dates of the meetings were agreed between the stakeholders. The remote validations also did not affect the project execution when the customer had an adequate collaboration (cs2 and cs4).

g) Specialized Automated Support

There was no unanimity among software engineers interviewed about the importance of using a particular tool for the quality of SRS. We presume that the importance of tool depends on the complexity level of the SRS.

Only the company of the first case study used a specialized tool (Caliber) to specify and manage requirements. The SRS was extensive and fragmented into various artifacts. Thus, the use of the tool was pointed out as a positive factor because it provided information on the history of changes and it was also crucial in the analysis of change requests because it provides information about the dependency relationship between the requirements automatically.

Companies from second and fifth case studies did not use any particular tool but pointed out that the absence of such a tool affected the quality of SRS. These companies also had an SRS fragmented on various artifacts.

On the other hand, in the remaining case studies (cs3, cs4 and cs6), no tools were used, but the team did not consider that this compromised the quality of the SRS. These companies adopt a simple documentation, based on a few artifacts. Despite considering that the absence of a tool had no impact on the SRS, in the third and fourth case studies, the teams considered important to have information about the historical changes and the traceability of requirements, respectively.

h) Using the SRS for the tests

Although investigating the factors that affect the tests of software is beyond the scope of this study, we tried to understand whether the content of SRS could contribute in some way with that activity. In none of the companies, there was an internal consensus among software engineers about the SRS to be used to guide the tests without the need of a particular artifact for this activity. Some software engineers consider that it is more appropriate to draw up a specific artifact because the test activities have their particular characteristics, and the tester and the developer have different needs. However, other software engineers consider that it is more productive to have a single artifact describing AC adequately and closer to what will be implemented and tested.

TDD is an agile practice that has been growing in companies. It establishes the use of tests as a mechanism to specify requirements to create executable specifications. This movement started at the end of the 90s, and then grew through other initiatives such as ATDD and BDD (ALLIANCE, 2016). These initiatives require a reformulation in the SRS to be used effectively to code and to perform the tests. These initiatives were not identified in the case studies. Despite the differences regarding the use or not of a particular document to guide the tests, the software engineers agreed on the need to automate the tests.

5.3 Translate the concepts

After analyzing the context, the story line of each study, the similarities, and differences, we performed the translation of concepts from the cases to unify the nomenclature of factors presented in Table 5.2. To anticipate possible situations that could occur when translating the concepts from one case to another, we adopted the strategy used by FRANÇA et al. (2014) based on principles of meta-ethnography to generalize and rename concepts, as described in Appendix D (Table D.2). It is important to remark that this strategy could entail some degree of threat to validity. Further, it is debatable that, from a pure interpretive or constructive stance, this type of generalization is appropriate. Therefore, we used this type of translation with parsimony and tried to be careful with the claims we based on it. Additionally, we anticipated that refutations could arise, but did not find any instance in this current study.

Identical and *Localization* translations (Appendix D, Table D.2) were trivial. In the former case, the names and meanings of the concepts were the same. In the latter there was no translation because the concept is context dependent. *Renaming* and *Generalization* required a bigger challenge. In Table 5.3, we present some translations performed and a brief definition is presented below. For example, in some cases, the use of the *«Acceptance Criteria»* improved the quality of SRS. However, in others cases the *«Inadequate Acceptance Criteria»* and *«Lack of Acceptance Criteria»* were pointed out as negative factors. So, these concepts were renamed in *«Acceptance Criteria»* concept that has a positive effect in SRS (Table 5.3, id #4).

Table 5.3 – Examples of concepts translations

Concepts	Type	Translation	id
Inadequate Team Experience with Specification	Identical	Inadequate Team Experience with Specification	#1
Outdated	Identical	Outdated	#2
Technical Aspects (+) and Lack of Technical Aspects (-)	Renaming	Technical Aspects (+)	#3
Acceptance Criteria(+), Inadequate Acceptance Criteria(-) and Lack of Acceptance Criteria(-)	Renaming	Acceptance Criteria (+)	#4
Clearness (+) and Unclear (-)	Renaming	Clearness (+)	#5
Insufficient Detail (-) and Completeness Agile (+)	Renaming	Completeness Agile (+)	#6
User Stories, Use Cases, Conceptual Model, Prototypes of UI, Examples and Business Rules	Generalization	Functional Requirements (+)	#7
Characteristics of Mobile Apps and Characteristics of Financial Software	Generalization	Characteristics of Software	#9
Low Customer Availability to validation and Low Customer Availability to detail and test	Generalization	Inadequate Customer Availability	#9
Characteristics of process (c1)	Localization	Characteristics of process (c1)	#10
Overlooked Requirements	Not applicable	-	#11

«User Stories», *«Use Cases»*, *«Conceptual Model»*, *«Prototypes of UI»*, *«Examples»* and *«Business Rules»* were generalized in *«Functional Requirements»* concept (#7). The *«Overlooked Requirements»* factor was reported by some software engineers (cs3, cs6), but it was not included in the model because although it is a problem that affects the project management, it is not a problem related to the content or how the requirements are described in the SRS (#11).

5.3.1 *Identifying the constructs*

According to the cross-case synthesis, the factors that can improve the quality of SRS in the perception of software engineers are:

- *«**Team-oriented**»*: The requirements are described using a language directed to the developer. Technical terms and jargon are used, if needed;
- *«**Clearness**»:* The requirements are described in a clear way, without ambiguities. Each requirement can only have a single interpretation*;*
- *«**Readability**»*: The description of the requirements is easy to understand;
- *«**Objectivity**»*: The requirements are described in an objective manner, without long, prolix, redundant and unnecessary details to developer;

- «***Simplicity***»: It is the art of maximizing the amount of work not done (AGILE MANIFESTO, 2001). In the context of the requirements specification activities, simplicity is to describe the requirements in the simplest way as possible, with few representations, models, and structures;
- «***Completeness "agile"***»: The requirement description is enough to be coded without the need to consult complementary sources. The completeness should be evaluated under the perception of the developer in the scope of each sprint, and not the whole project;
- «***Technical aspects***»: Consist of the description of the design and interface constraints, validation rules, error messages or any other technical information for the operationalization of the requirements;
- «***Acceptance Criteria***»: It is a description of criteria that support the acceptance, or not, of a requirement;
- «***Functional Requirements***»: It is the description of the functions or tasks to be performed by the system. Requirements can be represented using a textual format, for example, by user stories, use cases, business rules and concrete examples; or using a visual representation, such as conceptual models and mockups, or others representations;
- «***Non-Functional Requirements (NFR)***»: Consist of the description of the requirements to operate the system, such as security, performance, multilingual support, among others;
- «***Traceability***»: Consists of the description of the source of each requirement, as well as, the dependency relationships between them;
- «***History of Change***»: It is the history of changes made in the requirements;
- «***Automated Support***»: The activities of specification and management of requirements are carried out with the support of a specialized tool;

The factors that jeopardize the quality of SRS:

- «***Inadequate Customer Availability***»: The availability of the customer to collaborate with the development team or to validate the software is not adequate;
- «***Inadequate Team Experience with ASD***»: The development team has little experience with the values, principles, and practices of the ASD;
- «***Inadequate Team Experience with Specification***»: The development team has little experience with the requirements specification activity;
- «***Late validation by SW***»: Partial software versions are not released to customer at the end of each sprint, which delays the software validation;
- «***SRS used to requirements validation***»: SRS is used to validate requirements with the customer and the end user;

- *«**Contract agreements**»*: Contractual clauses that establish the availability, validation or approval of the SRS by the customer;
- *«**Outdated**»*: The description of the requirements is outdated, no longer corresponds to the needs requested by the customer;
- *«**Excessive changes**»*: The development team receives many requests for changes in partial versions of the application;
- *«**Fragmented Information**»*: The requirements are described in a fragmented manner in various artifacts that have different and complementary views;
- *«**Characteristics of software**»*: Intrinsic characteristics of the type of software developed or business area that the application is turned requiring an additional effort for the team;
- *«**Characteristics of process**»*: Bureaucratic practices used to release, validate and certificate partial versions of software, and control changes in requirements which require an additional effort from the team.

The factors were grouped according to the manner they affect the quality of SRS (*direct* or *indirect*), as shown in Table 5.4.

Table 5.4 – Quality factors by categories

	Category			**Factor**
DIRECT	Content (C)		+	«Team-oriented»
			+	«Clearness»
			+	«Readability»
			+	«Objectivity»
			+	«Completeness "agile"»
			-	«Outdated»
	Structure (S)	Type of information	+	«Functional Requirements»
			+	«Technical aspects»
			+	«Acceptance Criteria»
			+	«NFR»
			+	«Traceability»
			+	«Change History»
		How	+	«Simplicity»
			-	«Fragmented Information»
INDIRECT	staKeholders (K)		-	«Inadequate Customer Collaboration»
			-	«Inadequate Team Experience with ASD»
			-	«Inadequate Team Experience with Specification»
	Organizational (O)		-	«Late validation by SW»
			-	«SRS used to requirements validation»
			-	«Characteristics of process»
			+	«Automated Support»
	External (E)		-	«Characteristics of software»
			-	«Contract agreements»
			-	«Excessive changes»

Factors affecting quality directly are grouped into two categories: *Content* (C) and *Structure* (S). The *Content* category groups the factors related to how the requirements are described. The *Structure* category groups the factors related to the type of information contained in the SRS and how the structure is organized. Factors affecting quality indirectly are grouped into three categories: s*taKeholders* (K), *Organizational* (O) and *External* (E). The factors represented as '+' mean that their presence can improve the quality of SRS and their absence can impair it. The factors represented as '-' mean that the presence can jeopardize the quality of SRS and the absence can improve the quality. We did not find contradictory concepts between studies, but those may be found in other cross-case analysis.

5.3.2 *Outcomes*

The presence or absence of these factors affects the quality of SRS and consequently affects the ASD in a positive or negative way, as follows:

- *«**Effort required to specify**»*: The effort required for the development team to elaborate the SRS, with or without the presence of the client;
- *«**Effort required to code**»*: The effort required for the development team to conduct the coding activity;
- *«**Effort required to test**»*: The effort required for the development team to test the software;
- *«**Dependence between stakeholders**»*: Level of dependency that exists among stakeholders due to content of SRS or how it is specified;
- *«**Non-conformities in the software**»*: Quantity and type of nonconformities detected in the software due to SRS problems;
- *«**Knowledge Transfer**»*: The capacity of the SRS to be used as a mechanism for knowledge transfer among stakeholders, especially when there is turnover, and in distributed and maintenance teams;
- *«**Impact Analysis**»*: The capacity of the SRS to support the analysis of the impact of requests for changes in requirements;
- *«**Reuse**»*: The capacity of the SRS to contribute to the reuse of requirements during the specification and coding activities.

5.4 Translate the propositions

After the unification of concepts, the relationships among the factors (propositions) were analyzed to translate it across the case studies. The propositions were translated using the strategies proposed in meta-ethnography according to FRANÇA et al. (2014), and we add the *Localization* type, as described in Appendix D (Table D.3). As a result of cross-case analysis and synthesis, we extracted eleven propositions as shown in Table 5.5. The propositions translated received a sequential numbering different from that defined in the individual studies.

We did not find refutation instances in this study, but we believe it can be found in other cross-case analysis. The propositions identified only in the context of a particular case study were maintained. For example, the proposition number seven (#P7) in Table 5.5, in the third case study, the proposition 8 (C3P8) relates the inadequate customer collaboration with an insufficient SRS resulting in excessive changes in requirements. The reciprocal translations (RTA) were also trivial since we need only rewrite the propositions after the unification of the constructs (#P1, #P2, #P4, #P5, #P6 and #P9).

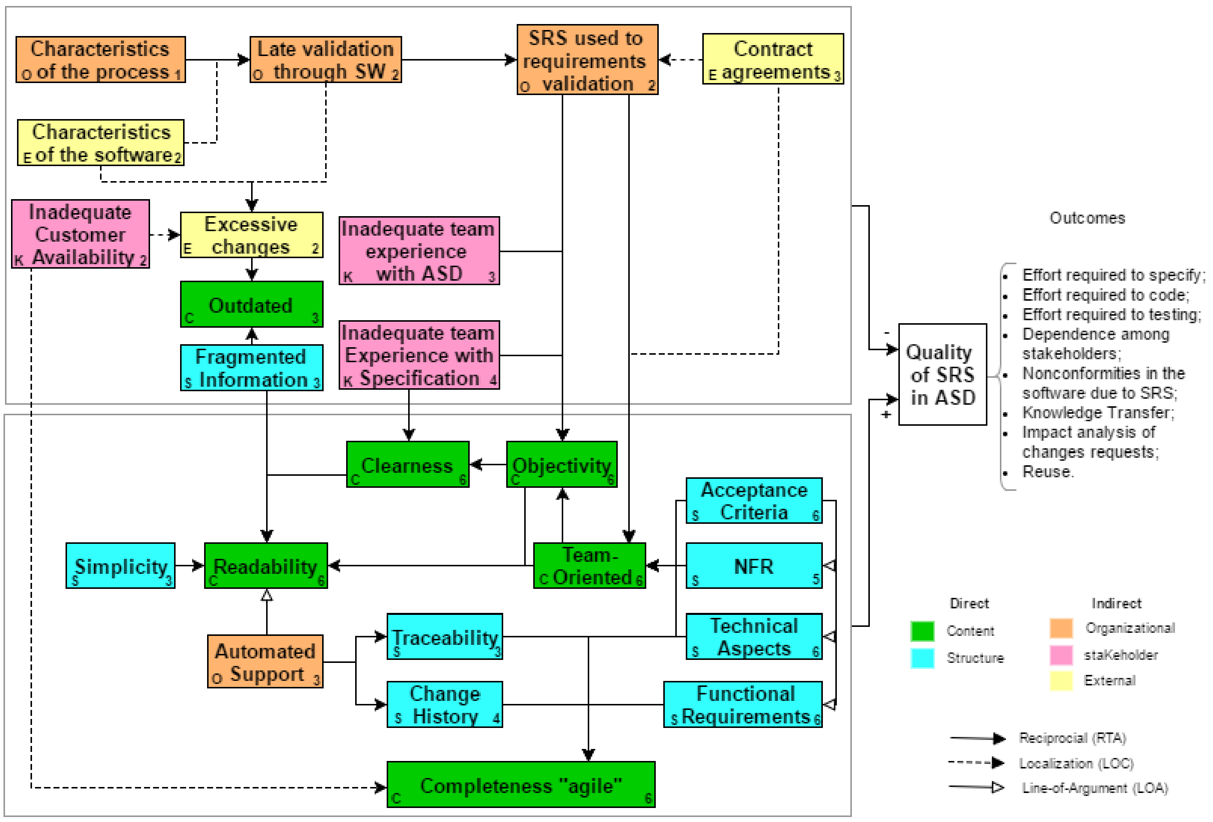
Table 5.5 – Translation of Propositions

Proposition	**Proposition**	**Translation**	**Type**	
C5P7: A *«customer-oriented»* SRS entails *«prolix»* descriptions which lead to a SRS with *«difficult readability»*.	**C4P5**: A *«team-oriented»* SRS tends to have more *«objectivity»* which facilitates the *«readability»*.	The «team-oriented» SRS contributes to the requirements to be described a more «objectivity» way which facilitates the «*readability*» for the developer.	RTA	**#P1**
C1P6: *The «completeness "agile"»* is obtained from the description of the *«functional requirements»*, *«technical aspects»* and *«non-functional* requirements».	**C3P5:** *«Insufficient Detail»* undermines the *«completeness»* of SRS.	*The «completeness "agile"» is obtained from the description of the «functional requirements», «acceptance criteria», «technical aspects» and «non-functional requirements». The content is undermined when the requirements have «insufficient detail».*	RTA	**#P2**
C4P6: The *«acceptance criteria»* improves the *«completeness»* of SRS.	**C6P10**: The *«inadequate acceptance criteria»* compromises the *«completeness»* of SRS.	The description of *«acceptance criteria»* related to *«technical aspects»*, *«non-functional requirements»* and *«functional requirements»* contribute to the SRS becomes more *«team-oriented»*.	LOA	**#P3**
C3P7: *The «simplicity»* and *«objectivity»* of SRS facilitate the *«readability»* for the developer.	**C5P2:** *«Prolix»* descriptions and *«fragmented information»* difficult the *«readability»* for the developer and compromise the *«clearness»*. **C1P2:** *«Fragmented information»* contributes to the SRS become *«outdated»*.	The *«simplicity»* and *«clearness»* of SRS facilitate the *«readability»* for the developer. On the other hand, The *«fragmented information»* in various artifacts hinders overall understanding of the requirements and contributes to the SRS become *«outdated»*.	RTA	**#P4**
C1P4: «*Inadequate team experience with ASD*» entails *«prolix»* descriptions that undermine the *«clearness»*.	**C4P7:** «*Inadequate team experience with specification*» entails *«prolix»* descriptions. **C3P4:** *The «objectivity»* of SRS influences in the *«clearness»*.	«*Inadequate team experience with ASD*» and «*inadequate team experience with requirements activities*» compromise the *«objectivity»* of SRS that influences in the *«clearness»*.	RTA	**#P5**
C1P9: Some *«characteristics of the process»* contributes to *«late validation through software»*.	**C2P8**: Some *«characteristics of the process»* contributes to *«late validation through software»*.	Some *«characteristics of the process»* and *«characteristics of the software»* may contribute to the validations through software running happen at longer time intervals than the duration of sprints.	RTA	**#P6**
C3P8: «*Inadequate availability of the customer*» undermines the breakdown of the requirements, the clarifying of doubts and the validation of the partial versions. As a result, the SRS gets with *«insufficient detail»* with for the developer and *«excessive changes»* are requested.	-	«*Inadequate availability of the customer*» undermines the breakdown of the requirements, the clarifying of doubts and the validation of the partial versions. As a result, the SRS affects the «completeness "agile"» and *«excessive changes»* are requested.	LOC	**#P7**
C3P1: *«Characteristics of mobile apps»* and *«inadequate customer collaboration»* lead to *«excessive changes»* which affect the quality of SRS.	**C1P3**: *«Late validation through software»* leads to *«excessive changes»* which affect the quality of SRS.	*«Excessive changes»* contribute to the SRS become *«outdated»*. *«Excessive changes»* could be due to some *«characteristics of software»*, the *«late validation through software»*, or due to *«inadequate customer collaboration»*	LOC	**#P8**
C2P9: *«Inadequate automated support»* contributes to the *«lack of traceability»* and *«lack of change history»* that impaired the *«completeness»* of SRS.	**C1P11**: *«Automated support»* can provide the *«traceability»* and *«change history»*.	*«Automated support»* can provide the *«traceability»* and *«change history»* that enhance the *«completeness "agile"»* of SRS.	RTA	**#P9**
		The *«automated support»* facilitates the *«readability»* of SRS.	LOA	**#P10**
C1P12: *«Contractual agreements»* and the *«late validation through software»* require the use of *«SRS to validate the requirements»* which makes the SRS customer-oriented with *«prolix»* descriptions.	**C5P5:** *«Contractual agreements»* requires a SRS customer-oriented resulting in *«prolix»* descriptions.	Sometimes, *«contractual agreements»* and the *«late validation through software»* require the use of *«SRS to validate the requirements»* leading to a SRS less *«team-oriented»* and *«objective»*.	LOC	**#P11**

Translations of Line-of-Argument type (LOA) demanded a greater effort of interpretation and induction. For example #P3, the presence of AC in the SRS was cited as a positive factor (C4P6). However, the inappropriate use of AC was pointed out as a negative factor which affected the completeness of the SRS (C6P10). Analyzing the data collected and the story line, we have identified a new proposition relating the acceptance criteria to NFR and technical aspects.

5.5 Building the model

The model illustrated in Figure 5.3 shows the factors that improve the quality of SRS (+ effect) and those that impair the quality (- effect). The relationship between the factors (propositions) and the outcomes resulting from the impact of these factors are also presented in the model. On the left side of each factor is displayed the acronym (C, S, K, O, E) corresponding to its category (Table 5.4). The right side shows in how many case studies the factor was pointed out (Table 5.2). The arrows (RTA, LOC, LOA) follow the nomenclature defined in Appendix D (Table D.3).



O Characteristics of the process 1
O Late validation through SW 2
O SRS used to requirements validation 2
E Contract agreements 3
E Characteristics of the software 2
K Inadequate Customer Availability 2
E Excessive changes 2
K Inadequate team experience with ASD 3
C Outdated 3
S Fragmented Information 3
K Inadequate team Experience with Specification 4
Outcomes
Effort required to specify;
Effort required to code;
Effort required to testing;
Dependence among stakeholders;
Nonconformities in the software due to SRS;
Knowledge Transfer;
Impact analysis of changes requests;
Reuse.
-
+
Quality of SRS in ASD
C Clearness 6
C Objectivity 6
S Acceptance Criteria 6
S Simplicity 3
C Readability 6
C Team-Oriented 6
S NFR 5
O Automated Support 3
S Traceability 3
S Technical Aspects 6
S Change History 4
S Functional Requirements 6
C Completeness "agile" 6
Direct
🟩 Content
🟦 Structure
Indirect
🟧 Organizational
🟪 staKeholder
🟨 External
Reciprocial (RTA)
Localization (LOC)
Line-of-Argument (LOA)

Figure 5.3 Model on quality of SRS in ASD

Following the agile principle that the SRS should not be used to validate requirements with the customer, a *«team-oriented»* SRS is a key factor to improve the quality of SRS in ASD. The description of the *«functional requirements»* is essential, but it is not sufficient for the coding activity. The description of the *«technical aspects»* and *«non-functional requirements»* makes the SRS closer to what will be coded, thus reducing the «effort required to coding». The specification of a requirement should also describe how it can be verified (ISO 29148, 2011). This information improves the understanding of the requirement and the performance of coding and testing activities. Defining adequate *«acceptance criteria»* supports an objective approach to conduct the acceptance tests of software with the customer. This information contributes to the completeness of the content.

The *«completeness "agile"»* had a different meaning compared to traditional development. First, in ASD, the completeness of the SRS should be evaluated under the scope of the developer perception. The SRS should not be used as a communication mechanism with the customer aiming at the validation of requirements. The SRS should be designed to be used by the developer for coding. Moreover, in ASD, the concept of completeness refers to detail the prioritized requirements for a given development cycle (sprint). In contrast, in waterfall development, all requirements are exhaustively detailed and validated, before starting the coding activity. In other traditional methods like RUP, the requirements are detailed only by iteration as ASD, but the SRS is used to requirements validation.

A *«team-oriented»* SRS becomes more objective with fewer redundancies and facilitates the *«readability»* for the developer. The *«simplicity»* and *«clearness»* of an SRS are also essential for the understanding of the developer. The customer-oriented SRS lead to excessive details, unnecessary to the developer, which impair the readability. The *«inadequate experience with ASD»*, the *«inadequate experience with specification»*, and the use of the *«SRS to validate requirements»* resulting in a SRS customer-oriented affecting the *«objectivity»* for the developer.

The ASD community does not recommend the requirements validation through SRS. However, sometimes this practice is carried out to satisfy *«contractual agreements»*, due to the *«characteristics of the process»* or *«characteristics of the software»* that lead the company to choose to make a late validation through software, as previously detailed in Section 5.2.2 (a).

Late validations and *«inadequate customer availability»* can lead to *«excessive changes»*. Low customer availability to detail requirements and clarify doubts often leads to excessive requests for changes in the partial versions. When the customer's unavailability is to validate the partial versions, the main problem is with the requirements overlooked. Although ASD builds on the notion that changes are opportunities for the companies, *«excessive changes»* may impair the team productivity and contribute to SRS becoming *«outdated»*.

The impact of automated support on quality of SRS depends greatly on the complexity of the documentation used. *«Automated support»* specialized is not considered as essential. However, it can provide information such as *«traceability»* of requirements and control of *«change history»* that are important in managing complex projects or with large teams. The description of the stakeholder who requested the requirement (source) and the relationships among requirements are essential to analyze the impact that the changes may cause in the project. These *«traceability»* allows the team to have a better understanding of time and people resources that need to be allocated to implement a given change. The *«traceability»* can also increase the *«reuse»* of requirements. The analysis of the relationships between requirements can point out those that are much referenced and therefore have the potential to be reused.

Documentation with details fragmented in various artifacts tends to be more complex and redundant, which undermines the understanding of the developer. Also, a greater effort is required to elaborate and maintain the artifacts updated. An *«outdated»* SRS jeopardized the quality and entailed more rework for the team. The SRS can reduce the *«dependence among stakeholders»*. An adequate SRS avoid that the developers interrupt their activities to clarify doubts. Similarly, an SRS of quality can be used to *«knowledge transfer»*, especially when new collaborators are introduced to the team and in distributed teams. Requirements described in a clear and straightforward way, and closer to what will be implemented facilitates the understanding of the developer and can reduce the *«non-conformities in the software»* arising from the erroneous interpretation of the SRS.

5.6 Enfolding Literature

An essential feature of the theory building is the comparison of the emergent concepts, propositions, or hypotheses with the extant literature (EISENHARDT, 1989). This involves asking what is similar to, what does it contradict, and why. So, we compare the findings of the cross-case synthesis with the systematic mapping (Section 5.6.1), values and principles of ASD (5.6.2), with practices of traditional development (5.6.3) and some related works (5.6.4), discussing similarities and differences.

5.6.1 *Systematic Mapping*

The investigation of the phenomenon in the literature identified twenty-one factors that affect the requirements specification activity in ASD as described in Chapter 3. The case studies investigated in the industrial practice, the factors of the *Techniques* Category, as shown in Table 5.6. In column "=", we identified the factors present in the SM and the case studies. Column "!" shows the factors that the investigation in the industry showed some conflict with the SM. Factors that were not evaluated during the case studies are identified in the last column (?).

Table 5.6 – Compliance between factors (SM and Cross-case synthesis)

Category	ID	Factor Description	=	!	?
	1	User stories are ambiguous	✓		
	2	Beginners professionals have much difficulty in writing useful stories	✓		
	3	The level of detail of user stories is inadequate	✓		
	4	Stories are inadequate to describe technical aspects	✓		
	5	Story cards are an incomplete notation	✓		
	6	Stories require the daily presence of the customer		✓	
	7	Use cases require a daily collaboration with the client		✓	
	8	Difficult to include technical aspects in the scenarios	✓		
	9	Much effort is required to write scenarios	✓		
	10	Use cases have too much information presented	✓		
TECHNIQUES	11	Difficulty in identifying what is relevant in the specification	✓		
	12	Developers are not accustomed to writing tests before coding			✓
	13	TDD requires a daily collaboration with the customer			✓
	14	TDD requires a thorough understanding of requirement			✓
	15	Insufficient SRS for coding and maintenance	✓		
	16	SRS is not appropriate for knowledge transfer	✓		
	17	Documentation is not useful for identifying faults	✓		
	18	Difficulty understanding the documentation	✓		
	19	Lack of clarity between the customer needs and the solution	✓		
	20	The structure of SRS leads to ambiguous requirements	✓		
	21	Inadequate to represent Non-Functional Requirements	✓		
	22	Customer expectations are not met			✓
	23	Inadequate user-developer interaction	✓		
CUSTOMER	24	Users don't know what they want			✓
	25	Low availability of customer	✓		
	26	Tedious scoping sessions with customers			✓
	27	Validation without the customer's perspective			✓
	28	Requirements are not agreed with the team			✓
	29	Inefficiency in the requirements analysis and inspection			✓
	30	Inadequate automated support to specify requirements	✓		
	31	Overscoping			✓
	32	Inefficient sharing of documentation			✓
	33	Tendency to omit architectural issues			✓
	34	Reusability of requirements does not occur adequately	✓		
MANAGEMENT	35	Team unmotivated because of constants changes	✓		
	36	The control in changing requirements is inefficient	✓		
	37	Architectures are not scalable due to constant changes			✓
	38	Much time spent with changes in requirements	✓		
	39	Difficulty in creating estimates accurate of cost, schedule, performance			✓
	40	Frequent reprioritization of requirements			✓
	41	Difficulty in keeping the updated SRS	✓		
	42	Conflicts due to many sources of requirements			✓
	43	Communication gaps			✓

	44	Difficulties with Distributed Teams	✓	
	45	Difficulty in promoting the sustainability of teams		✓
	46	Misunderstandings due to the absence of key people		✓
	47	Difficulty in the managing large backlogs		✓
	48	Extra effort to integrate the requirements		✓
	49	Essential Requirements are not adequately treated		✓

None of the analyzed projects used TDD. Thus, it was not possible to analyze the factors #12, #13, and #14. None of the interviewees in the six companies considered that the daily presence of the customer was required. Software engineers pointed out that the SRS based on user stories and use cases was insufficient because it was focused on functional requirements. But they did not point out that daily customer participation was required. Regarding the functional requirements, the doubts were initially cleared with the internal PO, and then with the customer, if necessary. Thus, the investigation in practice showed contradictory findings with the factors #6 and #7 of the SM. The results of the cross-case analysis confirmed that all other factors related to Techniques category identified in SM affect the quality of SRS in ASD.

The factors related to the customer and the management requirements were not part of the scope of the investigation in the case studies (#22 to #49). Nevertheless, it was possible to confirm the impact of some of them. In two case studies, software engineers pointed out problems arising from low availability (#25) and inadequate iteration with the customer (#23). Excessive changes in requirements have been identified as a factor that contributed to the SRS become outdated (#41). Also, the productivity (#38) and the motivation of the development team (#35) were affected due to excessive changes. Inadequate control of changes was also mentioned as a negative factor (#36). In all case studies, reuse depended in large part on prior knowledge of the existence of a requirement or business rule. Reuse only happen in requirements specified by the same software engineer. There was not an established practice for a systematic reuse of the requirements (#34).

The case studies also showed contradictory results with SM regarding the SRS in the distributed teams (#44). One company worked in a fully distributed manner. Another company had part of the development team distributed. In all companies, the customer and the development team stayed in different cities. Only in two companies, the validations were conducted in a face-to-face way. However, the distribution of stakeholders in different locations was not pointed out as a negative factor.

The investigation of the phenomenon in practice introduced new factors that had not been pointed out in primary studies analyzed in SM as follows: lack of acceptance criteria, examples, and traceability of requirements. Also, some characteristics of the software and the process, late software validation, SRS used to requirements validation, contract agreements, and the SRS structured in many artifacts.

5.6.2 *Agile Practices*

The findings of cross-case synthesis pointed out accordance with all agile values and with ten principles of Agile Manifesto (2001). However, we identified conflicts with two agile principles. In all cases studies, we found disagreements regarding the principle that says *"Business people and developers must work together daily throughout the project"*, as mentioned previously. The customer's presence is important to understand "what" he/she needed. However, the software engineers conduct other technical activities to design "how to" implement the requirements. During these activities, the client's presence is not necessary. The customer's presence in these activities would damage their performance because would be an additional communication channel.

Another conflict is regarding the principle that says *"The most efficient and effective method of conveying information to and within a development team is the face-to-face conversation"*. The face-to-face conversations are used to share best practices, problems, solutions, and impediments. However, sometimes, the SRS is, in fact, the most appropriate way to transfer knowledge. There is a very high turnover in development teams (TECHREPUBLIC, 2015). So, knowledge could be lost if relevant information is only in the minds of some people. Whenever new people are allocated to the team, some software engineers interrupt their activities to pass on knowledge to the new members. Geographically distributed development teams may work in different time zones and schedules, making it harder to establish face-to-face conversations. In some companies, after the end of the project, the teams are allocated to other projects. Changes and new requirements are made by another team, responsible for the maintenance of all software. In these cases, the SRS plays a fundamental role in knowledge transfer from the development to the maintenance teams.

The user story is the most used technique in ASD (LUCASSEN et al., 2016). XP argues that stories focus on business rules, the description must be made in the customer's language, without describing technical details (BECK, 1999). According to Cohn (2004), stories must be written in the language of the business, not in technical jargon. However, the findings of case studies differ from these assertions of Lucassen, Beck and Cohn. In the other hand, according to Ambler (2016), the goal of the ASD is to implement requirements, not document them. He argues that the SRS should capture design details. Bjarnason et al. (2011) argues that an SRS closer to what will be implemented may reduce the effort required to gather information for coding, testing, and maintenance. Our findings are in accordance with these assertions of Ambler and Bjarnoson. The model argues that *«acceptance criteria»*, and *«technical aspects»* are essential to ensure a *«team-oriented»* SRS and the *«completeness»*. These descriptions are closer to what will be implemented, reduce the gap between the customer's needs and the details required to produce the solution, and thus, optimize the time required for the coding activity.

One of the principles of the Agile Manifesto is "*Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage*". Changes can be requested due to support legislation, in the customer's business process, new customer needs, among others. The changes require the evaluation the impact in resources, schedule, personnel, budget, and infrastructure, among others. Requirements traceability can help to conduct this analysis. Maintaining the requirements traceability requires a great effort. According to Ambler (2016), maintaining the requirements traceability makes sense only in the following situations: 1) Automated tooling support exists; 2) Complex domains; 3) Large teams or geographically distributed teams, and 4) Regulatory compliance. Otherwise, it is simpler and makes more sense to ask some people familiar with the system, to estimate the change, rather than devote time to updating the traceability, he argues. The traceability is a positive factor when there is an automated support.

5.6.3 *Traditional Development*

The goal of this study was to investigate the quality of SRS in ASD, and as a result, to build a model that describes the factors that affect the quality and how they influence the work of software engineers. Some of the factors identified in the model also affect the SRS in the traditional development, such as the low availability of the customer, the outdated documentation, the inadequate experience of the team, the lack of clarity and of automated support. However, the model has many factors that are peculiar to the ASD, as explained following.

The ISO quality attributes (ISO 29148, 2011) are widely adopted in traditional development, where the SRS is used to validate requirements. However, in ASD, the requirements validation is often done upon frequent software deliveries. Thus, the perception of quality of SRS tends to be different. We identified some agreements and differences between the ISO attributes and the quality factors of the model. The model address *Unambiguous*, *Traceable*, *Complete*, and *Verifiable* ISO attributes. The main difference between the model and ISO attributes is about the SRS to be *«team-oriented»* describing *«technical aspects»* to have an SRS closer than what would be implemented. These factors are conflicting with the *Implementation Free* attribute which lays down that *"The requirement while addressing what is necessary and sufficient in the system, avoids placing unnecessary constraints on the architectural design. The objective is to be implementation-independent. The requirement states what is required, not how the requirement should be met"*.

Specifying the requirements using a language focused on the developer is one of the main factors in the model that is particular to ASD. The description of technical aspects and acceptance criteria were identified as factors that are closer to what will be implemented. However, in traditional development, the SRS focuses on describing what the customer needs, rather than how the requirements must be met. Moreover, it is not recommended to use technical terms and jargon because the SRS is used to validate the requirements, thus, it should use a language oriented to the customer. In traditional development, the SRS often has prolix texts to facilitate the understanding of the customer, but that are unnecessary to the developer. In the model, the objectivity of SRS is defined as one of the factors that facilitates the understanding of the developer and improves his/her productivity.

Singular and *Feasible* attributes were not identified as significant factors in the case studies. At the beginning of the interviews, we asked developers about what they considered a high-quality SRS and which attributes they would like to find in an SRS to implement a feature. These attributes were not mentioned by any of the interviewed software engineers in the six companies. In a second stage, when we explained each of these attributes to the developers, and then we asked them about the importance of each attribute, none of the interviewees considered that the *Single* and *Feasible* attribute were relevant in ASD.

Regarding the *Necessary* attribute, the developers did not have a clear perception of the extent to which a requirement was necessary for the client. So it was not pointed out by the interviewed software engineers as something that affected the quality of SRS. However, this attribute was relevant in the companies that adopted a payment system functionality delivered, given that only necessary requirements were paid by the customer. Software engineers had difficulties in maintaining the *consistency* of a requirement when it was described in a fragmented way in various artifacts. However, in neither case study, software engineers were concerned to maintain or check the consistency between requirements. When the sprints have a short duration, and the validations were frequent, this attribute was not considered critical.

In non-agile approaches that adopt an iterative and incremental development, such as the RUP, it is common to use the SRS to validate the requirements before starting the coding activity. Thus, the validation of partial versions is not as frequent as in ASD. In the model, *late validation* is pointed out as a factor that compromises the ASD, since it causes excessive changes in the SRS and the application. The late validation through software implies the use of the SRS to validate the requirements partially, which compromises the adequacy of the SRS to the development team because it requires the use of a language oriented to the customer.

The use of an extensive documentation containing several models, structures, and artifacts is common in traditional software development to represent various views of the requirements to meet the needs of different stakeholders, unlike what is recommended in ASD. In accordance with the agile manifesto, one of the factors identified in the model is the *simplicity* which favors minimum documentation with few structures, describing just what is needed for the coding activity, taking the developer's perspective.

One of the four values of the agile manifesto is "*Customer collaboration over contract negotiation*". According to the model, contracts can have a negative impact in ASD, when they establish that the SRS must be validated by the customer before coding starts, or when they require that the SRS should be customer-oriented. However, in traditional development, the contract is seen as a key factor, because it establishes the conditions for the development of the software and is a legal guarantee for the parties involved, especially for government agencies.

5.6.4 *Related Works*

Heck and Zaidman (2014) proposed a framework for verification of the quality of requirements. Twenty-one verification criteria are classified under three high-levels criteria: i) completeness; ii) uniformity and iii) consistency & correctness. The framework was not designed specifically for ASD. The framework is based on a certification model of software products previously proposed by one of the authors, rather than designed specifically for ASD. The framework is based only on literature studies. It is not based on studies conducted in agile projects in the industry. Regarding the content, the model presents accordance with the framework in the use of AC, mockups, conceptual model and description of possible solutions. However, the framework does not define criteria related to readability, the language used, structure, traceability and or level of detail as defined in the model (see Table 5.4). Furthermore, given that the framework evaluates only the final quality of SRS, factors that affect the quality indirectly as defined in model are also not considered by it.

Lucassen et al. (2016) proposed a framework that defines a collection of 13 criteria to assess the quality of user stories regarding syntax, pragmatics, and semantics. However, the framework focuses on the intrinsic quality of the user story text. In addition, it does not consider characteristics of the teams and companies that can affect the quality of the SRS. One of the criteria is *problem-oriented* which states that a user story must only specify the problem, not the solution to it. This criterion is conflicting with the factors identified in our model aimed at making the team-oriented SRS and closer to what will be implemented. In agreement, we identify the *Unambiguous* criteria, *Minimal* (*simplicity*) and *Complete*. The framework also does not address factors that affect the quality of SRS an indirect way, as defined in the model.

Although Heck and Zaidman (2014) and Lucassen et al. (2016) describe some criteria that should be considered in the SRS in ASD, their goal is to evaluate the final quality of SRS. Unlike our study, their purpose is not to investigate how these criteria relate. Those frameworks do not cover *Organizational*, *Stakeholders* and *External* factors as defined in the model which also affect the quality of the SRS. Moreover, they do not investigate how the criteria influence the work of the team. These previous studies lack in explanatory power about the factors that affect the SRS in ASD, are inconclusive or have been of limited use in practice.

5.7 Findings

The cross-case synthesis reveals ten findings (F) that can have implications for software organizations in their quest to create better work environments and teams, and thus to improve the quality of SRS in ASD, as follows:

- **F1**: The SRS should not be used as a mechanism for requirements validation, which in turn must be done through frequent software deliveries as established in agile manifesto. If this practice is not possible due to some limitation or characteristics of the process or the software, the validation must be done through the same SRS used by the developer.
- **F2**: The SRS should not be focused on the customer; instead, it should be directed to the development team and closer to what is implemented describing technical aspects, NFR, acceptance criteria, as well as functional requirements;
- **F3**: The use of acceptance criteria is an excellent strategy to make the SRS more directed to the developer. This practice also facilitates execution of tests. However, the AC must address all types of requirements and technical aspects, and not just the functional requirements;
- **F4**: A fragmented SRS structured in various artifacts tends to be more complex and hard to maintain. Independent of the form of representation of requirements (user stories, rules, use cases, etc.), the organization must describe the information required for coding, in an integrated way and with few artifacts;
- **F5**: The experience of software engineers with ASD affects the quality of the SRS. Companies need to promote the sharing of best practices to help software engineers obtaining experience on how to specify requirements in ASD;
- **F6**: Smaller requirements are easier to estimate and implement. Sprints of short duration with smaller requirements tend to produce a simpler, clearer and readable SRS. Long sprints tend to produce an unnecessarily detailed SRS;
- **F7**: Contractual agreements must consider the values and principles of ASD. Thus, a contract must establish that the working software is the mechanism for requirements validation, instead of the SRS. In cases where the availability of documentation for the customer or regulatory agencies (e.g., government) is required, the same SRS produced for the developer should be provided. The team should not waste time preparing different artifacts for several stakeholders;
- **F8**: Specific tools to support the requirements activity are not mandatory, but it can improve the traceability and the change history of requirements and thereby facilitating the impact analysis of requirements changes, especially in complex projects or large teams. It also can improve the reuse, integration and consistency of requirements and reduce the effort required to elaborate the SRS;

- **F9**: When customer collaboration is inadequate, the adoption of validations face-to-face tends to be more appropriate than remote validations. Remote validations are aimed at giving greater flexibility to the customer and also at reducing stakeholders travel costs, if they are based in geographically far sites. A too loose collaboration with the customer entails a bottleneck in the project, and this challenge can be better faced with face-to-face communication;
- **F10**: The daily presence of the customer integrated into the development team is not essential, and can sometimes be detrimental to the development process. The customer's availability for answering questions, detailing requirements and validating the partial versions according to the team's demand is more relevant, which can also be performed remotely through email, phone, instant messaging or other communication tools.

5.8 Threats to Validity

5.8.1 *Internal*

Regarding the internal validity, to ensure that the results represent the reality, we interviewed team members with different roles in each study. Moreover, we analyzed the meanings attributed by the participants in their answers to the questionnaire. This analysis was triangulated with the data collected from the documents and with the notes collect in meetings that the researchers acted as observers. In each study, we tried to ensure that the concepts that emerged from the coding process faithfully represented the reality of the projects investigated.

In the translation and synthesis of concepts, the generalization of six different concepts that emerged from different realities was a threat. The concept translated could not accurately represent the six realities (companies), or worse, would any of them. To address this threat, we coded the data, including transcription of fragments, and we used the strategies recommended in meta-ethnography. We also consulted the literature to validate the translated names and meanings.

To increase the credibility, we used member checking (MERRIAM, 2009), also known as respondent validation, to avoid misinterpretations of what participants said. We devoted special attention to data collection and analysis, to allow the identification of contradicting evidence and complementary explanations. The results of this work were frequently scrutinized in follow-up meetings with company representatives, to ensure its validity with their help.

5.8.2 *Reliability*

An important question in qualitative research is whether the findings are consistent with the data collected (MERRIAM, 2009). Reliability is related to the extent to which the research findings can be replicated by the same or other researchers. The question is not whether findings will be found again but whether the results are consistent with the data collected. Therefore, we hope to achieve the consistency between our findings and the data collected in each study.

We do not expect that the results of each study will be reproduced. Even if we repeat the same protocol, in the same companies, with a similar set of participants, their perception of quality of SRS in ASD and the organizational context are likely to have evolved. To increase the reliability, we used data triangulation and peer review of the findings by another researcher. Also, we described a detailed account of the methods, procedures, and decision points in carrying out the studies.

5.8.3 *External Validity*

External validity is concerned with the extent to which the research findings can be applied or used in contexts different from those in which the study was first conducted (MERRIAM, 2009). This is related to how generalizable the results are. In qualitative research, it is the reader or user of the study that should primarily be engaged in the generalization of the research findings. They are best equipped to decide to what extent the findings can be applied to their own situations.

We used two strategies to promote the transferability of results. First, the research method was detailed, so that other researchers can use the procedures to produce similar and comparable studies. The research instruments were tested in six studies. We provided rich descriptions of each context investigated to contextualize the study such that readers will be able to determine the extent to which their situations match the research context, and, hence, whether findings can be transferred. We conducted a deeper investigation of the differences between the six contexts to build the integration of the studies. In addition, we sampled the participants to achieve maximum variation since this would provide richer data and a richer model.

5.9 Summary

From the initial theoretical background identified in SM, we conducted an investigation in the industry through a cross-case analysis of six companies with distinct characteristics to achieve high data variation and richness of results. A model emerged from this analysis that provides a deeper understanding of the phenomenon describing the factors that affect the quality of SRS in ASD and how the factors affect the work of the software engineers. The initial findings from the synthesis of two case studies were published at an international conference (MEDEIROS et al., 2016b). The complete model was described in a paper that is under review by an international journal.

The model establishes that the SRS in ASD should be directed to the development team, so it should be closer to what will be implemented. A simple SRS that is written clearly, without ambiguity and redundancies was also considered a relevant quality factor. On the other hand, the description of the requirements fragmented in various artifacts and prolix compromise the quality of the SRS. Further, the model shows that the quality of SRS is also affected indirectly by *Stakeholders* factors such as the inadequate experience of the team, low customer availability; *Organizational* factors such as the use of SRS to validate requirements, late validations, and *External* factors such as contract agreements and characteristics of the kind of software.

Finally, we compared the model with some studies in the literature discussing similarities and differences. Based on the findings and factors of the model, in the next chapter, we propose an approach, designed to advance and solidify our current knowledge in this area, and to support the activity of the requirements specification in ASD.

6
AN APPROACH TO SPECIFY REQUIREMENTS IN ASD

This chapter details the Requirements Specification for Developer (RSD) approach proposed to support the requirements specification activity in ASD. Section 6.1 describes how the RSD approach was derived from the results of systematic mapping and industrial case studies. Section 6.2 introduces the approach and describes its related metamodel. The practices of RSD approach are detailed in Section 6.3. Section 6.4 presents the structure of the RSD. Some possibilities for extending the approach are described in the Section 6.5. Finally, the related works are discussed in Section 6.6.

6.1 Derivation of the RSD approach

The SM pointed out several factors related to the software requirements specification in ASD (Chapter 3) which have been confirmed in the investigation in the industry (Chapters 4 and 5) as previously summarized in Table 5.6. Among these factors, we highlight the client-oriented SRS that lacks design information needed for coding activities. The findings of these investigations are also reinforced by other studies that indicate that the SRS used in agile projects is insufficient for the coding activity (HEIKKILA et al., 2015; INAYAT et al., 2014). Even with the continuous presence of the customer during the software develop, the design information cannot be gathered because the client is not capable of perceiving it. The design information is neglected in user stories (HEIKKILA et al., 2015), making it difficult the activities of coding, testing and maintaining (BJARNOSON et al., 2011) as well as the knowledge transfer in distributed development and high turnover teams (INAYAT et al., 2014).

We based in these findings to propose an approach that aimed to produce an SRS with a single integrated view of functional requirements and design information directed to the software engineers in ASD, answering the RQ3 defined in Section 1.2. The approach was named as Requirements Specification for Developers (RSD).

The definition of the approach was guided by the model (Chapter 5) and the agile manifesto that establishes the validation of requirements through the frequent software deliveries, rather than using the SRS for that purpose. Thus, the SRS should be directed to the software engineers, rather than the customer. In this context, the approach proposes the adoption of three well-established design practices that make the SRS targeted to the software engineer: Specification of Acceptance Criteria+, Conceptual Modeling and Mockups Modeling.

According to the systematic mapping and the case studies, the SRS currently used in ASD focuses only on functional requirements. NFR and other restriction necessary for the operation of the system are not adequately described in SRS. In order to support the coding activities, the RSD approach adopts concepts of ATDD and BDD through the acceptance criteria (AC+) to describe customer needs and system requirements using a single integrated view of the requirements, as defined in Section 2.3.1 and detailed in Section 6.3.3.

The findings of the literature and industry investigations also pointed out that the constant change requests are factors that compromise the productivity of the development team because the software architectures usually present scalability problems resulting from the negligence in the design activities. To reduce these problems, the RSD approach adopts the conceptual modeling to identify the business concepts (entities, attributes, and relationships). This practice contributes to building a more scalable architecture and reduces the rework required to meet the constant changes.

The mockups modeling practice identifies the visual interface elements between the system and the user. 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 (FERREIRA et al., 2007), nor integrated into the functional requirements outlined by the user stories. The innovation of our approach is to systematize the use of these practices in ASD, and integrate the description of the functional and system requirements in a single view in order to provide a SRS with the information required for coding.

RSD approach attempts to address some of the factors identified in the model resulting from the cross-case synthesis detailed in Chapter 5 and reviewed in Figure 6.1 (adapted from Figure 5.3). The RSD approach addresses some factors of *Content* and *Structure* categories that directly affect the quality of SRS, as described in Section 5.3.1.

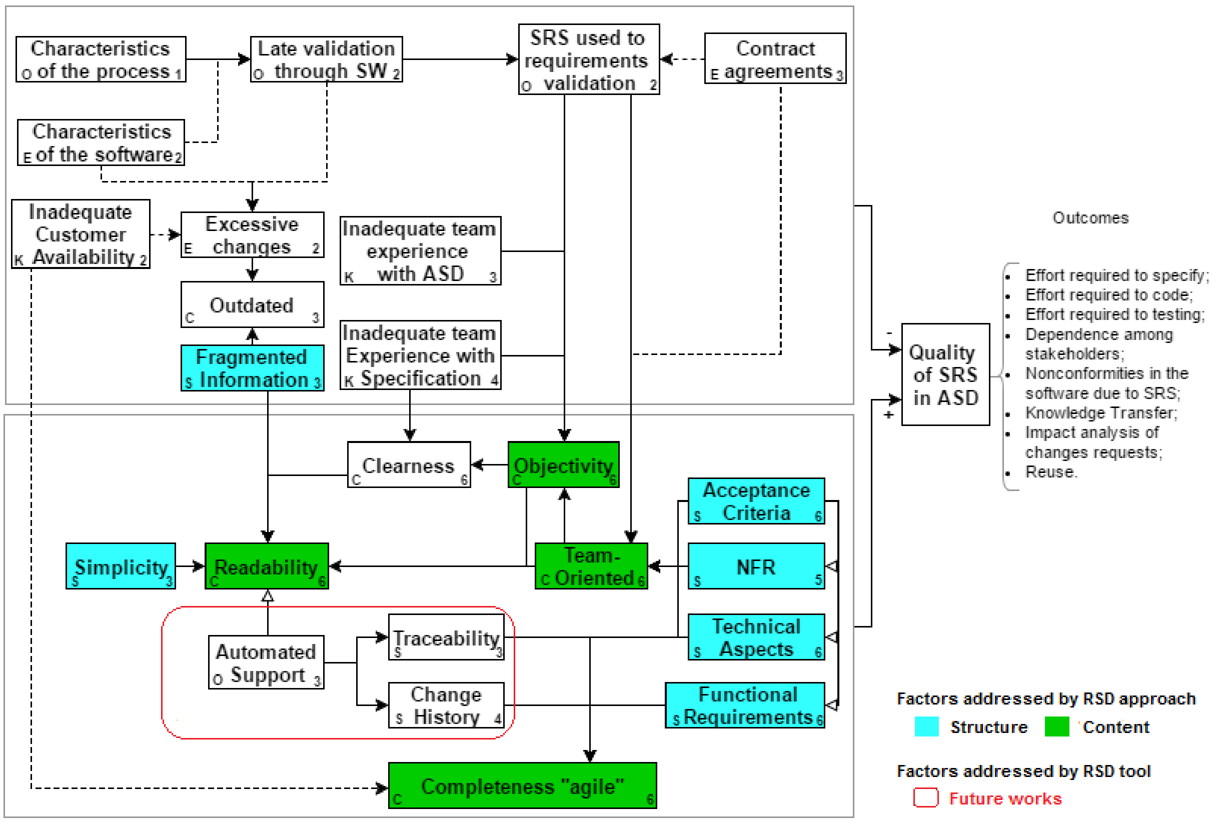

Characteristics of the process O 1
Late validation through SW O 2
SRS used to requirements validation O 2
Contract agreements E 3
Characteristics of the software E 2
Inadequate Customer Availability K 2
Excessive changes E 2
Inadequate team experience with ASD K 3
Outdated C 3
Fragmented Information S 3
Inadequate team Experience with Specification K 4
Outcomes
Effort required to specify;
Effort required to code;
Effort required to testing;
Dependence among stakeholders;
Nonconformities in the software due to SRS;
Knowledge Transfer;
Impact analysis of changes requests;
Reuse.
Quality of SRS in ASD
-
+
Clearness C 6
Objectivity C 6
Acceptance Criteria S 6
Simplicity S 3
Readability C 6
Team-Oriented C 6
NFR S 5
Automated Support O 3
Traceability S 3
Technical Aspects S 6
Change History S 4
Functional Requirements S 6
Completeness "agile" C 6
Factors addressed by RSD approach
Structure
Content
Factors addressed by RSD tool
Future works

Figure 6.1: Quality Factors addressed by RSD approach

The structure of RSD is oriented to the developer which makes it very objective and thus contributes to having clearer requirements. However, the acceptance criteria are described using a natural language, so, it is possible only merely reduce the ambiguity and not prevent it entirely. Moreover, the clearness of the SRS also depends deeply on the experience of those who write the requirements.

As described in Table 5.4, the model also points out other factors that indirectly affect the quality of SRS. However, the RSD practices are not intended to address these factors. For example, i) Inadequate participation of the client and Inadequate team experience factors (category *Stakeholders*), ii) Characteristics of the process, the late software validation and the use of SRS to requirements validation factors (*Organizational* category) and iii) Characteristics of the software and contractual agreements factors that depend on external issues the organization (*External* category). These factors are related to the companies' context and affect the quality of SRS, regardless of the approach used to specify the requirements (user stories, use cases or RSD, for example). Thus, these factors are not addressed by RSD approach.

The three factors (Automated Support, Traceability and Change History) highlighted in red (Figure 6.1) will be addressed in the future through a tool that will provide an automated support to elaborate the RSD, generate the traceability of the requirements and control of change history, among others features, as described in Section 8.3. The RSD approach address some of the findings (F2, F3, F4 and F5) pointed out in cross-case synthesis (See Section 5.7). The other findings are related to organizational factors, external or related to stakeholders.

6.2 MetaModel

According to Gonzalez-Perez and Henderson-Sellers (2008), a metamodel is a domain-specific language oriented towards the representation of software development methodologies and endeavors. The metamodel shown in Figure 6.2 summarizes the constructs and rules needed to build SRS using the RSD approach.

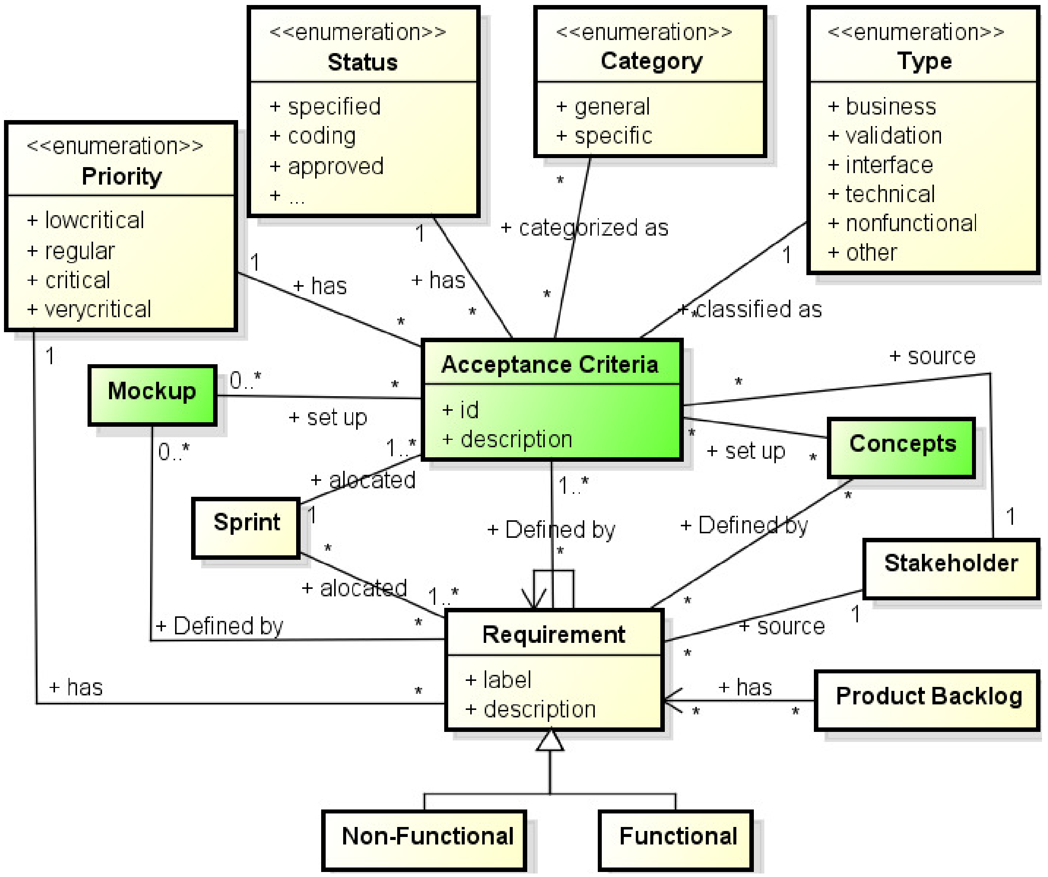


Figure 6.2: Metamodel of RSD approach

In the RSD metamodel, the ***Requirement*** element (***functional*** or ***non-functional***) is identified by a ***label*** and has a high-level ***description*** that succinctly describes a user or system requirement. An example of a requirement described using the metamodel is presented in Section 6.4. Each requirement is detailed through the description of the ***Concepts*** (from the conceptual model), ***Mockup***, and the ***Acceptance Criteria*** (AC+) related, as defined in Section 2.3.1.

The ***Product Backlog*** (PB) contains a set of requirements that are allocated to ***Sprints*** (cycles of development) according to the customer ***Priority*** (low critical, regular, critical, very critical). A requirement may have zero or more related *Mockups*. In fact, mockups are not mandatory, since there may be requirements that do not require a visual interface to the user. The requirements and AC+ are defined by a ***Stakeholder***, who can be a customer (or representative), or any member of the development team. The following assertive should be considered:

- AC+ may be applied to more than one requirement;
- All requirements must have at least one associated AC+;
- A requirement might have AC+ with different priorities that can be allocated to different sprints;
- AC+ does not need to be associated with a Mockup.

An AC+ is categorized as ***General*** (G) when it has the potential to be reused. Otherwise, it is categorized as ***Specific*** (S) to a particular requirement. To have an SRS targeted for the developer, the AC+ defines 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 classified into six **types** as shown in Table 6.1.

Table 6.1 – Acceptance Criteria+ Type

Type	**Description**
Business (B)	Represents a restriction related on to intrinsic nature of the business. For example: All foreign athlete must have a number of passport;
Validation (V)	Represents a validation that the application needs to perform, but that is not directly related to the core business. For example: The athlete's email address must be valid;
Interface (I)	Represents any restriction related to the user interface. A widget must be enabled (or disabled) depending on some condition or the content of another widget. For example: The list of cities depends on the selected state.
Technical (T)	Represents a technical restriction on how the solution should be implemented. Most of the time, they are rules associated with the confirmation action buttons. For example: Before including an athlete, the application should automatically include the corresponding person.
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.

A requirement may have AC+s with different priorities that can also be allocated to different sprints. In addition, new AC+ can be identified in any sprint throughout the development process.

The state diagram of an AC+ is shown in Figure 6.3. The initial status of an AC+ is ***Specified*** and it occurs when it is identified by any stakeholder. The status is ***Coding*** when the implementation starts. The status is ***Implemented*** when the developer considers that the implementation was finalized. The status is ***Awaiting Fixes*** if the acceptance tests identify any non-compliance, otherwise the status is ***Approved***. The status is ***Modified*** when any change is made after approval. The status is ***Canceled*** if the AC+ is deemed not applicable.

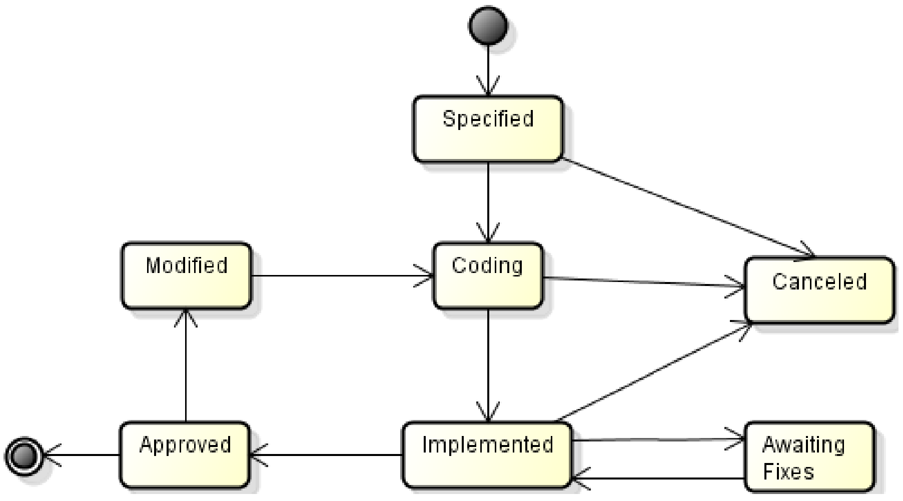


Figure 6.3: State Diagram of an AC+

6.3 Systematizing the use of Design Practices

RSD approach can replace other techniques used in ASD, such as user stories (US) and use cases, but overcoming the limitations described previously. Thus, RSD can be used with XP, Scrum, or any other agile method where the client validates the requirements through working software, as established in the Agile Manifesto (2001). Based on this agile principle, RSD approach focus on the development team and the SRS produced is not intended to be used as mechanism to requirements validation with the customer.

In RSD approach, the customer is involved throughout the development process, describing and prioritizing his/her needs, and validating software deliveries, frequently. In the RSD structure, customer needs and system requirements are represented using a single view that integrates the 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, NFR and technical constraints (acceptance criteria). The three design practices provide a wider requirements coverage when compared to US, which only addresses user requirements.

The practices of RSD aim to detail each requirement. As shown in Figure 6.4, the process of detailing starts with the creation of the conceptual model which should be done considering all requirements of a sprint.

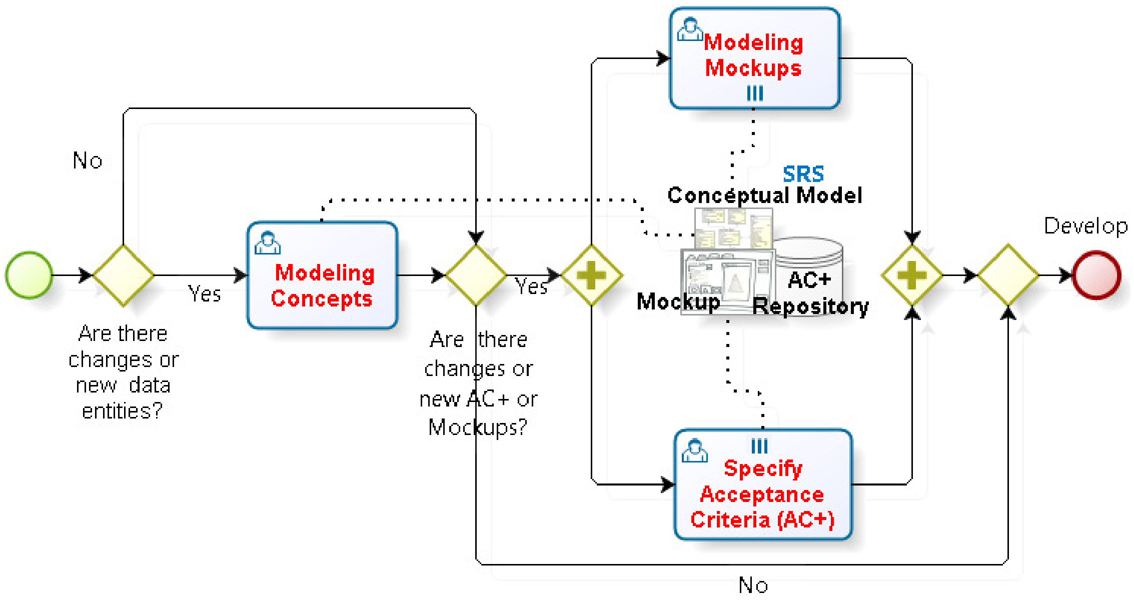

No
Yes
Are there changes or new data entities?
Modeling Concepts
Yes
Are there changes or new AC+ or Mockups?
Modeling Mockups
SRS
Conceptual Model
Mockup
AC+ Repository
Specify Acceptance Criteria (AC+)
Develop
No

Figure 6.4: Practices of RSD approach in each sprint

Then, the mockups are modeled, and the acceptance criteria 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+ practices being carried out in parallel, one instance for each requirement of the sprint. The coding of each requirement starts as soon as an initial set of the AC+ and related mockups are specified. There is no need to wait for the detailing of all AC+ neither all requirements of the sprint.

The frequency of performing these practices depends on customer availability and duration of sprints. When the client is integrated into the development team, these two practices can be carried out every day and start coding immediately. When the customer availability is limited, these practices can be conducted once a week, for example.

Usually, at the end of this process it is expected that there is one RSD (document) for each requirement. However, it is not mandatory; the team may consider dividing a requirement into several RSDs. In this case, it is worth to asses if it would be more appropriate to modify the PB as a means of sub-dividing the requirement, so that each requirement has only one RSD associated. Moreover, the approach recommends that each RSD describes only one requirement in order to facilitate the readability. But this is also a team choice.

There are companies that adopt the strategy of all team members interact with the customer to identify and specify the needs, and also work in coding activity. Some companies assign the analyst and developer roles to different people. The RSD approach applies equally in both situations.

6.3.1 *Modeling Concepts*

This practice aims to model the concepts (data entities) related to the requirements and should be performed at the start of each sprint. The conceptual modeling is one of the differentials of RSD compared to other agile approaches. As previously reported (Section 2.3.2), the current agile approaches focus on behavioral modeling. There is no concern in defining the conceptual model in a systematic way together with the specification of user stories. An inadequate data 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 data model have a very large impact on implementation, for example, changes in the multiplicity of relationship between data structures. Hence the importance of dedicating time to properly design the application's data model.

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 data 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. RSD approach not restricts the use of any tool in particular. However, some tools[16] have additional features that allow the generation of the database structure in SQL format, for example.

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 the agility.

The initial data structure 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 it.

The conceptual model that is expected as a result of this task can be represented by a class, conceptual or entity-relationship diagram, depending on the company's culture. Figure 6.5 shows an entity-relationship diagram as example of output for this practice. This example shows the data concepts of the requirements allocated to a sprint of a doping control system (PETRONIO et al., 2016).

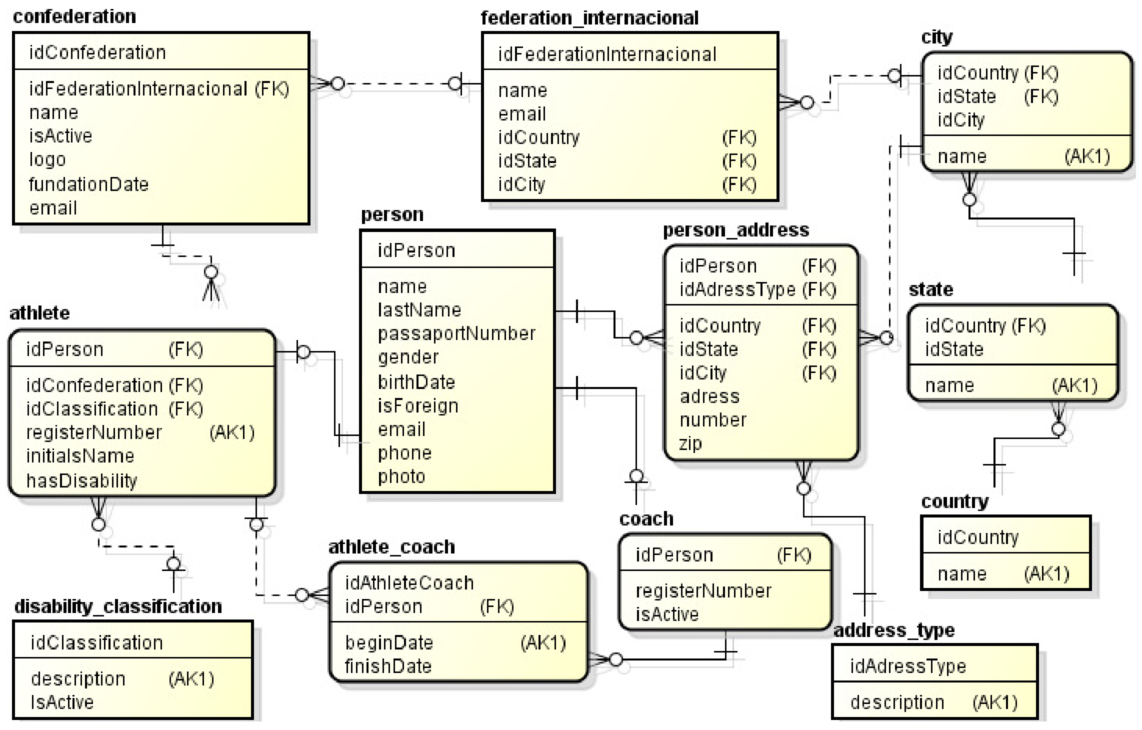


Figure 6.5: Example of a Conceptual Model

[16] http://www.databaseanswers.org/modelling_tools.htm

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.

Once the conceptual model has been created covering all requirements of the sprint, the development team must model mockups and specify the AC+. These activities must be performed at the same time and must be repeated for each requirement allocated to sprint, following the priority previously established by the customer.

A summary of the modeling concepts practice is presented in Table 6.2.

Table 6.2 – Summary of the practice MODELING CONCEPTS

MODELING CONCEPTS	
Purpose:	Model the concepts which represent the requirements of a sprint
Precondition:	Planning of the backlog has been done
Inputs:	Product Backlog
Outputs:	• RSD (Conceptual Model)
Actions:	• Review the requirements previously and identify doubts; • Identifying data entities together with the client; • Analyze requirements that are allocated to the upcoming sprints if they have a strong impact on the conceptual model; • Register any AC+ that have been identified; • Register new requirements (or changes) that have been identified; • Create conceptual model.
Tools:	• Whatever data modeling Tool;

6.3.2 *Modeling Mockups*

This practice aims to model the mockups of the requirements that require some interaction between a user and the application. As previously described in Section 2.3.3, mockups have proven an efficient practice to capture and defining functional requirements (INAYAT et al., 2015; RICCA et al., 2014). Mockups improve requirements gathering, without implying an additional effort in the process. One of their advantages is that they are technically valuable for developers and, at the same time, fully understandable by end-users (RIVERO et al., 2014).

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 development team should conduct this practice together with the customer. The mockups can be drawn using a piece of paper or on a chalkboard. In this case, photos can be incorporated into the RSD. Also, the software engineers can use any tool that he/she has knowledge. RSD approach does not restrict the use of any tool in particular to the creation of mockups. However, some tools[17] have additional features that allow exporting mockups to PNG, or HTML, for example. The development team must validate the mockups with the customer, making the necessary corrections that may be requested.

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.

Figure 6.6 shows an example of the mockup related to a requirement labeled as "*Registration of athlete*" of a doping control system (PETRONIO et al., 2016). The high-level *description* of this requirement is "*The system should enable the inclusion and updating of data of national and foreign athletes of sports federations recognized by the International Olympic Committee*".

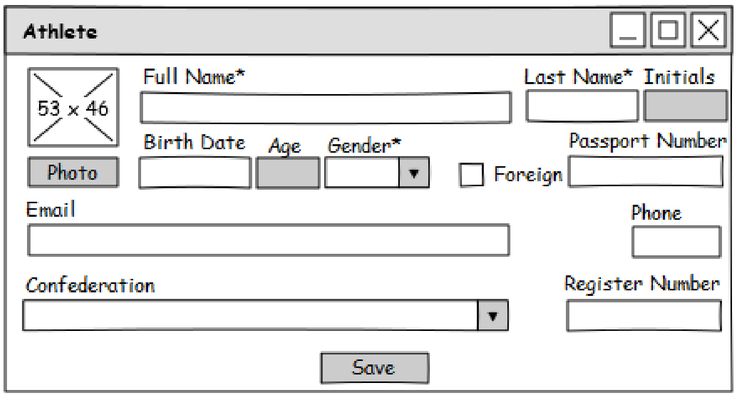


Figure 6.6: Example of a Mockup

A summary of this practice is presented in Table 6.3. The development team must identify and specify the related AC+ simultaneously with this practice.

[17] http://www.creativebloq.com/wireframes/top-wireframing-tools-11121302

Table 6.3 – Summary of the practice MODELING MOCKUPS

MODELING MOCKUPS	
Purpose:	Model the mockups of the requirements that require some interaction between a user and the application
Precondition:	The data model has been constructed
Inputs:	Product Backlog, RSD (Conceptual Model)
Outputs:	RSD (mockups)
Actions:	• Sketch an early version of mockups from the conceptual model; • Elaborate and validate the mockups with the customer, making the necessary corrections that may be requested.
Tools:	• Whatever Prototyping Tool.

6.3.3 *Specifying the Acceptance Criteria+*

This practice aims to identify or specifying the AC+ associated with a requirement. The RSD approach is based on the agile principle that the understanding of requirements is obtained in an iterative and incremental manner as the client validates partial versions of the software. In this way, the team should not waste too much time trying to identify all the acceptance criteria exhaustively. The identification of new criteria throughout the development cycle is part of agile development.

The specification of the AC+ should be performed in parallel with the Modeling Mockups practice. 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 are of the business type (Table 6.1). However, the team can extract other AC+ from the conceptual model and the knowledge gained from other conversations with the customer. Moreover, many technical, interface, NFR, and validation rules can be reused from other requirements. AC+ of this type can also be identified by developers during the coding activity.

AC+s are described in natural language (textual). The RSD approach does not define a particular format (structure) to write the AC+, unlike user stories that have a standard format (*As a <role>, I want <desire> [so that <benefit>]*). The AC+ must be described in objective way directed to the developer, without prolix texts. The smaller the better, and should have a binary result: pass or fail.

Terzakis (2016) offers a detailed checklist to detect ambiguity during description of textual requirements, such as vagueness and subjectivity. Also, ISO 29148 (2011) provides some terms, such as superlatives or vague pronouns, which should be prevented to ensure bound and ambiguity. Moreover, there are some tools available in the industry and academia that assess the quality of textual requirements and signal when a review is necessary. The teams can use some of these practices and tools to improve the quality of AC+. Génova et al. (2001) has developed a tool (called RQA) and compared it with seven other tools. The RQA tool assesses textual requirements described in Microsoft Excel format using the following indicators:

- Size (characters, words, sentences, paragraphs);
- Readability based on average syllables per word;
- Punctuations per sentences and quantity of acronyms/abbreviations;
- Imprecise terms and usage of verbal forms;
- Dependences and overlapping with other requirements.

Some examples of AC+ are presented in Table 6.4. These AC+ are related to "*Registration of athlete"* cited in previous section. The Type column refers to one of the types defined in Table 6.1. The last column defines whether the AC+ is of the *General* category (G) because it has the potential to be reused, or if it is *Specific* (S) of a requirement. AC+ initially classified as *Specific* may be reclassified as *General* when the opportunity for reuse is identified later.

Table 6.4 – Acceptance Criteria Examples

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

The AC+s must be stored in RAC to encourage the reuse during the specification and coding activities. The RAC can be operationalized through a shared document among stakeholders or using any tool to support the requirements activities or even the project management that provides features for registering the needs of stakeholders. The approach does not restrict the adoption of any particular tool.

A summary of this practice is presented in Table 6.5. Acceptance criteria+ besides being used to specify and implement the requirements can also be used to guide the testing activity.

Table 6.5 – Summary of the practice SPECIFY THE AC+

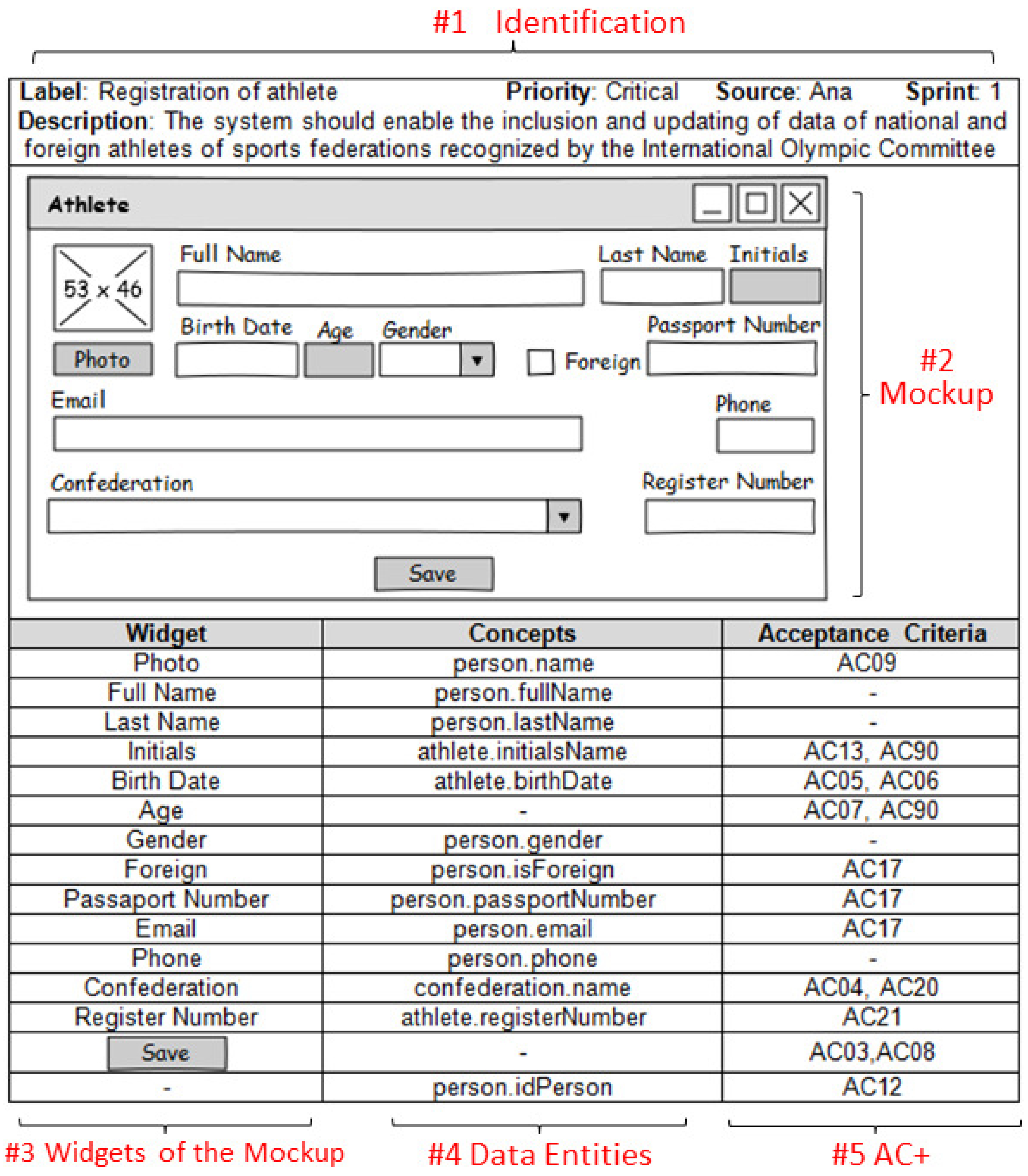
SPECIFY THE ACCEPTANCE CRITERIA+	
Purpose:	Specify the AC+ of a Requirement
Precondition:	The conceptual model has been identified
Inputs:	Product Backlog, RSD (Conceptual Model, Mockups)
Outputs:	RSD (Conceptual Model, Mockups, AC+)
Actions:	• Identify AC+ from the conversation with the customer; • Extract AC+ from the conceptual model; • Identify AC+ that can be reused from other requirements; • Elaborate the RSD.
Tools:	• Whatever Tool.

6.4 Structure of RSD

The structure of RSD joins the concepts, mockups and AC+ offering an integrated vision for the developer. To illustrate, Figure 6.7 shows the RSD related to "*Registration of athlete"* requirement. The RSD is divided into five parts.

The first part (top) identifies the requirement through the label, high-level description, priority, requestor stakeholder, and sprint, as defined in the metamodel. The second part 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 third part (the left column in the table) presents widgets that are present on the mockup. The fourth 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.

Finally, the fifth part (right column) shows the AC+s related to the widgets and the data entities. 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 to a data entity (e.g. AC3) nor to a widget (e.g. AC12). In general, AC+s 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.



Label: Registration of athlete **Priority**: Critical **Source**: Ana **Sprint**: 1
Description: 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
Passaport Number	person.passportNumber	AC17
Email	person.email	AC17
Phone	person.phone	-
Confederation	confederation.name	AC04, AC20
Register Number	athlete.registerNumber	AC21
Save	-	AC03,AC08
-	person.idPerson	AC12

Figure 6.7: Example of an SRS using RSD approach

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 customer. 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 clarify understanding for the team (e.g. AC13).

If user stories were used to describe this requirement, the language used would be customer-oriented and focused only on business requirements. As a complement to the user stories, the company could use mockups and the conceptual model, but in this case, the use of such practices would not be associated with the AC+. 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 user stories, but by business need, regardless the roles related to them. Thus, the 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 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.

6.5 Extensions

The use of the RSD practices can be adapted according to the type of project, software developed or to meet any particularity of the development teams. The modeling of mockups is a practice that fits perfectly in some types of projects, such as development of information systems and mobile applications. However, some software does not have much visual interaction with the user, such as web services and batch processing applications. In these cases, the RSD approach can be more focused on modeling the concepts and specifying the AC+. The same applies to some companies use frameworks to generate a basic user interface from the data model.

Some projects aim the maintenance and the evolution of existing applications. Typically these projects have a stable data model, so, the development activities focus on the inclusion of new features and bug fixes. In these types of projects that do not yet have a RSD and the application is already running in production, the use of the approach can be simplified to just describe new concepts, widgets and AC+ and those that will be changed. Instead of describing all the widgets and concepts that are already implemented in the software.

The RSD structure can also be adapted to meet the particular needs of the teams. In the doping control project (PETRONIO et al., 2016), the development team defined a new notation (*) in the RSD structure to identify in a visual way fields that require a mandatory fill. And other notation (#) to identify fields that did not permit editing, i.e., they are read-only. Moreover, the company adopted a notation to highlight the AC+ that should be implemented in a different Sprint. For example, the AC09^2, where the number 2 identifies that the AC+ should be coded in the second Sprint.

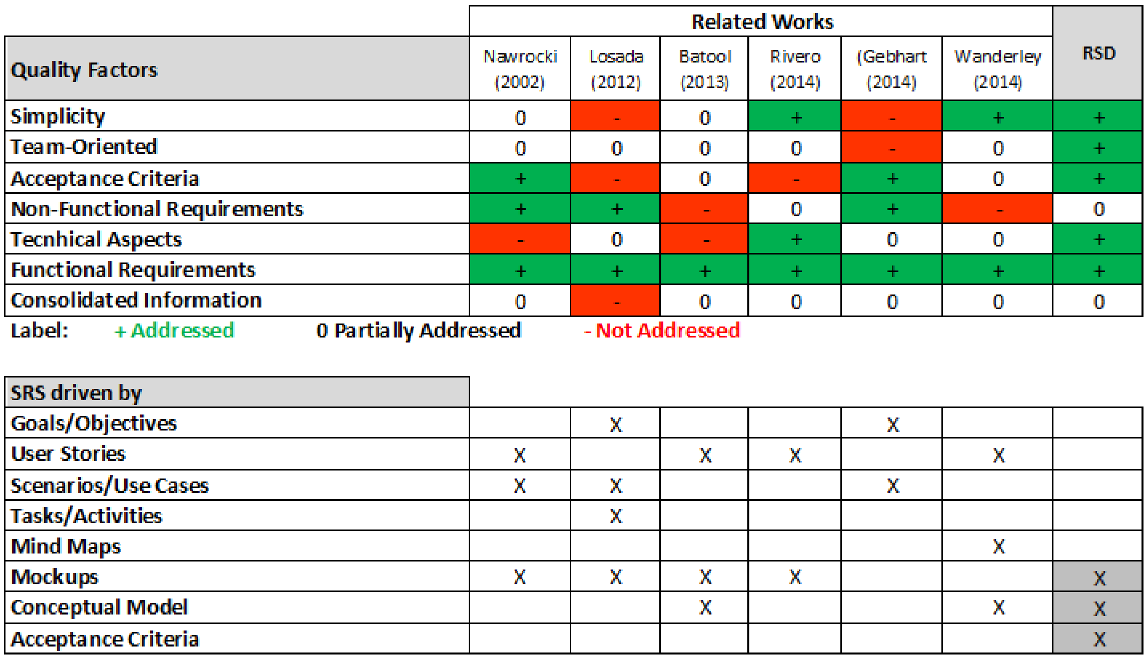
As described in Section 2.1, ISO 29148 (2011) removed the prioritization attribute (ranked for importance). The initial RSD structure did not represent the AC+ priority. However, the software engineers of the first case study extended the approach to representing the priority of the criteria to help in the planning of sprints, as described in Section 7.4.1. Thus, the AC+ priority has been incorporated into the RSD approach.

The approach suggests that the AC+ be classified into some types to encourage reuse as defined in Table 6.1. This classification can and should be adapted according to the particularities of each company. For example, the second company that evaluated the approach (see Section 7.4.2) chose to unify the *Validation* (V), *Interface* (I) and *Technical* (T) types into a single type called System (S). The team also created a new attribute to order the priority of an AC+ in the coding activity in each sprint.

6.6 Related Works

Faced with the challenges related to RE in agile projects, this thesis proposes a new approach (RSD) in order to support the requirements specification activity in ASD given the limitations of the User Stories and Use Cases previously described. In this context, six studies related to this research were identified: (NAWROCKI et al., 2002), (LOSADA et al., 2012), (BATOOL et al., 2013), (RIVERO et al., 2014), (GEBHART et al., 2014) and (WANDERLEY et al., 2014).

These studies were analyzed according to some quality factors of the model defined in Chapter 5. Each factor was evaluated as: i) Addressed (+): if the factor was fully addressed; ii) Partially Addressed (0): if the factor was addressed partially; iii) Not Addressed (-): if the factor was not addressed. Figure 6.8 summarizes the results of this analysis.



	Related Works						RSD
Quality Factors	Nawrocki (2002)	Losada (2012)	Batool (2013)	Rivero (2014)	(Gebhart (2014)	Wanderley (2014)	
Simplicity	0	-	0	+	-	+	+
Team-Oriented	0	0	0	0	-	0	+
Acceptance Criteria	+	-	0	-	+	0	+
Non-Functional Requirements	+	+	-	0	+	-	0
Tecnhical Aspects	-	0	-	+	0	0	+
Functional Requirements	+	+	+	+	+	+	+
Consolidated Information	0	-	0	0	0	0	0

Label: + Addressed   0 Partially Addressed   - Not Addressed

SRS driven by							
Goals/Objectives		X			X		
User Stories	X		X	X		X	
Scenarios/Use Cases	X	X			X		
Tasks/Activities		X					
Mind Maps						X	
Mockups	X	X	X	X			X
Conceptual Model			X			X	X
Acceptance Criteria							X

Figure 6.8 Related Works

Nawrocki et al. (2002) proposes an extension of the XP (eXtreme Programming) by introducing new RE practices. He asserts that an SRS based on user stories is not enough for the developer. Based on some practices of CMMI (2010) and some quality attributes of ISO 830 (2001), the authors argue to link tests cases to requirements and include the use of scenarios. The extension suggests the adoption of user interface prototypes and practices to identify the RNF. The SRS is directed to the client because it is used to validate requirements with it. The conceptual modeling and technical aspects are not address by the extension.

InterMod (LOSADA et al., 2012) is a methodology that uses Model-Driven Development (MDD), and user-centered design to agile development. The methodology is guided by User Objectives (UO) that are user desires. UO can fit with one or more functional requirements (use cases). The eight models proposed offer different views of the requirements but end up compromising the agility of the process. There is no integration between the models which are customer-driven. NFRs can be described in the *system* model. The *user* model is adopted to describe color preferences, font, size, among others. However, other technical aspects such as validation rules are not described. Acceptance criteria are not included in the methodology.

Batool et al. (2013) proposed a scrum framework to improve the RE process. The framework is based on User Stories to describe functional requirements. Class diagrams and user interface prototypes are used, but the SRS lacks non-functional requirements and technical aspects. The framework also adopts other artifacts such as story cards, index cards, and vision document.

ELECTRA (Extensible modeLdriven Enduser CenTRic API) is an approach that allows capturing requirements related to APIs (Application Programming Interface) using mockups (RIVERO et al., 2014). The ELECTRA process is an adaptation of Scrum, and it mandatorily requires building mockups with essential end-user participation to specify the stories targeted to he/she. After all User Stories are associated to mockups, developers use a tool to tag the mockups with API-related annotations. Constraint annotations enable the definition of business rules and action annotations. Technical specifications can be described as annotation. The approach does not use conceptual modeling, and neither defines acceptance criteria.

Gebhart et al. (2014) argues that scenarios are an appropriate way to describe a system from the user's point of view in ASD, and presents an enhancement of existing scenario-based requirements engineering techniques to fulfill the quality characteristics of the international standard ISO 29148 (2011). The methodology establishes three initial activities for the identification of stakeholders, identification and prioritization of goals. The goals are realized through scenarios that are customer-oriented, free of implementation details and do not contain architectural decisions. The methodology also requires the construction of other artifacts, such as Glossary, Constraints, Scenarios for RNF, and Derivation of the acceptance criteria from the scenarios. A lot of artifacts ends up compromising the simplicity of SRS and the agility of the process.

SnapMind provides a visual requirements language based on mind maps to represent both user stories and domain models for agile development (WANDERLEY et al., 2014). The framework aims to make the requirements modeling process more user-centered. The process is composed of three activities. The first activity is the software specification (containing the definition of the domain model and the user story as sub-activities). In the second activity, the SRS is validated by end-user. The third activity is conducted by software engineers to verify the consistence between requirements models. The framework does not treat NFR, mockups and AC and it does not support technical constraints.

In all of these approaches, the SRS is used to requirements validation with the customer, although the ASD suggests the use of frequent releases through software running for this pursuit. This strategy entails an SRS directed to the customer, rather than to the developer. Except for the ELECTRA approach, the SRS focus in the description of functional requirements and do not address technical and design constraints adequately.

The innovation of the RSD approach is to provide a SRS targeted to the development team that systematizes the use of acceptance criteria, mockups and conceptual modeling in ASD and integrates the description of the functional and technical requirements in a single view providing an SRS with the information required for coding. According to some software engineers who evaluated the RSD approach, the description of NFR could be improved if the approach provided an RNF catalog (See Section 7.4.1).

6.7 Summary

In order to answer the RQ3 (*How to produce an SRS with a single integrated view of functional requirements and design information directed to the development team in ASD?*) defined in Section 1.2, this chapter detailed the RSD approach proposed to support the requirements specification activity in ASD addressing some factors of the model and some findings described in Chapter 5. The metamodel presents the constructs and rules needed to build SRS using the RSD approach.

The three design practices of RSD approach are detailed. The practices are conducted with the customer collaboration. The first practice aims to identify the concepts in order to produce a data model and to facilitate the changes of the requirements that are frequent in ASD. The second and third practices are conducted in parallel. The second practice aims to model the mockups in order to capture the visual requirements. The third practice aims to specify the AC+ related to each requirement. But, not only functional requirements, technical aspects, system requirements and NFR are also specified. AC+ is based on the concepts of ATDD and BDD that aim to create tests for the requirements before implement it.

The innovation of the approach is to systematize the use of these practices in ASD, and integrate the description of the functional and system requirements in a single view providing a wider requirements coverage when compared to US, which only addresses user requirements. RSD is based on the principle that frequent software deliveries are used to validate requirements. Thus the RSD is not intended to requirements validation with the customer. RSD approach is directed to developer and recommends an SRS very objective, without prolix and unnecessary descriptions to the developer.

An example was used to illustrate the structure of the RSD. Following the simplicity principle, the approach adopts an SRS with few models integrated into a single artifact.

A paper detailing the RSD approach was accepted at 32nd ACM Symposium on Applied Computing - SAC (See Table 8.1, #1). An outline of the proposed approach was also previously published in an international journal (MEDEIROS et al., 2016) as a partial result of the investigations carried out in this thesis.

Next chapter details the empirical studies that evaluated the RSD approach in practice.

7
EVALUATIONS

This chapter describes the design, procedures and results of two empirical studies conducted to assess how the RSD approach works in the industrial practice. First, Section 7.1 explains the choice of the evaluation method. Section 7.2 presents the design of the empirical studies. Section 7.3 presents the procedures conducted to collect and analyze the data. The results of the evaluations are presented in Section 7.4 and discussed in the Section 7.5. Finally, Section 7.6 presents the threats to validity of the evaluations.

7.1 The evaluation method

Experiments are essentially reductionist – they reduce complexity by allowing only a few variables of interest to vary in a controlled manner while controlling all other variables. If critical variables are ignored or controlled, the experimental results might not generalize to real-world settings (EASTERBROOK et al., 2008). Formal experiments are sometimes difficult to conduct when the degree of control is limited. To impose full control, formal experiments are often small, which is a problem when you try to increase the scale from the laboratory to a real project (KITCHENHAM et al., 1995). We did not choose experiments as the research method to evaluate the use of the RSD approach because the real context in practice is essential in the evaluation process and we do not have control over some variables. The evaluation only through a controlled experiment would not be enough because it would not consider the following variables of the real-world:

- The RSD is built collaboratively with the customer following the principles of the Agile Manifesto. In addition, collaboration with the customer is also required to clarify doubts during the coding activity and to validate partial versions of the software. Thus, the iteration with the customer is an important variable in ASD;

- To assess the use of RSD approach, it is also essential to observe the difficulties and facilities to code and test from the RSD. We cannot observe properly these activities evaluating the requirements in an isolated manner. Thus, it is necessary to investigate, in an integrated way, the entire development cycle;
- Reuse of requirements, dependence among stakeholders, rework, impact analysis of change requests, and knowledge transfer are variables that require the observation of the use of the approach in a real-world setting for some months and involving several sprints. We cannot assess these variables easily through controlled experiments.

We set out from the principle that the assessment of the RSD approach cannot be separated from the real context. Thus, we conducted two empirical qualitative studies to assess how the RSD approach works in practice and gather insights to enhance it. The strength of qualitative approaches is that they account for and include differences - ideologically, epistemologically, methodologically - and most importantly, humanly. They do not attempt to eliminate what cannot be discounted. They do not attempt to simplify what cannot be simplified (MERRIAM, 2009). The empirical studies were conducted following the guidelines suggested by Runeson and Martin (2012) composed of five steps: Planning; Preparation; Collecting evidence; Analyzing the data collected; and Synthesis.

7.2 Design and Preparation

The studies had an explanatory purpose (RUNESON and MARTIN, 2012). The evaluations were conducted in two companies following a single standard protocol. The unit of analysis was the software engineers.

The goal of the empirical studies was to assess how RSD works in practice. The following Specific Research Questions (SRQ) were defined:

- SRQ1: How the team evaluates the SRS produced using the RSD approach?
- SRQ2: How the RSD approach affects the work of the team?

7.2.1 *The Sample*

The selection of the companies was done very carefully to provide relevant data to evaluation. The precondition for selection of the companies was the use of agile practices in the software development. To conduct the investigation, we were conditioned on acceptance and willingness of companies to use the RSD approach in one of their projects. We got two enterprises that fitted in these preconditions. The companies selected the projects to be investigated according to these prerequisites. We interviewed all software engineers of the teams investigated in the two companies.

The first evaluation was conducted over 12 months in the development of an information system for doping control (PETRONIO et al., 2016) for a federation affiliated to the IAAF (International Association of Athletics Federations) which is the maximum worldwide athletics association and is recognized by the International Olympic Committee (IOC). The project covered features from the registration of the athlete to the judgment when the test result is positive. The development team consisted of ten software engineers (two system analysts and eight developers).

The second evaluation was conducted over 3 months in a small software company established in 2004. The investigation was carried out in the scope of one of the three development teams of the company that had four software engineers (all developers) responsible for the evolution and maintenance of three information systems. An information system for the management of residential condominiums that is running on more than 40 customers. An information system for a financial cooperative, and another for the company's internal administrative control.

7.2.2 *Preparation*

Before the empirical study be initiated in each company, we talked to the team about their expectations about adopting a new approach to replace user stories. Then, the studies protocol was elaborated describing the procedures to collect and analyze the data. A questionnaire was designed to guide the interviews with software engineers (See Appendix E).

In the final stage of preparation, a training was conducted in each empirical study to introduce the RSD approach to teams. The RSD practices were detailed for software engineers. Then, a pilot was performed with some requirements that were specified using the RSD approach and then implemented. The pilot aimed to clarify doubts, and familiarize the team with the new approach.

7.3 Procedure for collecting and analyzing data

In each empirical study, data were collected and analyzed simultaneously, in incremental and iterative steps in order to answer the research questions outlined in Section 7.1. As described in Section 6.1, the RSD approach aims to address some of the quality factors established in the model detailed in Chapter 5. Thus, part of the evaluation of RSD approach was done using these factors as a reference. We used three data collection sources: observations, analysis of documents and interviews. To increase the credibility of the internal results, we did a triangulation of the data obtained from the sources, as shown in Table 7.1 . The data collected and analyzed were reviewed by another researcher (master student).

Table 7.1 – Data collected by source

	Data	Observations	Documents	Interviews
Team Evaluation	The content and structure of the RSDs	X	X	X
	The effort required to specify requirements, code and test using RSD	X		X
	The difficulties that software engineers had when using RSD approach	X		X
	Dependence between stakeholders	X		X
	Impact analysis of change requests	X		X
	Amount of changes made in the RSD (volatility)	X	X	
	The type and number of Non-Conformities (NC) identified in the software by the team during acceptance tests	X	X	

7.3.1 *Observations*

The researchers did not work on the activities of the development process, not interfered in the way that the projects were being conducted, nor made any sugges-tions or criticisms. The researchers participated as observers in some activities car-ried out by the team to perceive how software engineers were using RSD approach and the difficulties faced. For example:

- Discovery sessions with the customer to detail their needs, specify the AC+, mockups, and the conceptual model;
- Team meetings to share better practices, problems and difficulties that compromised in the specification, coding and test activities;

- Meetings involving the development team, the PO and the customer to perform acceptance tests;
- Meetings to discuss and analyze the impact of change requests to requirements.

7.3.2 *Documents*

The analysis of the documents helped to understand the difficulties pointed out by the stakeholders during the meetings and interviews. Data were collected from documents, such as RSD, the repository of AC+ and the results of acceptance tests (when they were registered). In the first evaluation, the RSD was elaborated by the systems analyst in collaboration with the PO, and sometimes by the customer. In the second empirical study, the RSD was always elaborated by the PO.

Each RSD was evaluated by the software engineer responsible for its coding. They evaluated how the content of each RSD was specified according to six quality factors (Table 5.4): *Team-Oriented*, *Objectivity*, *Readability*, *Clearness, Completeness* and *Outdated.* Each factor was assessed as in C*ompliance (1)* or *Non-Compliance (0)*. The others quality factors of the model were evaluated during the interviews.

7.3.3 *Interviews*

The interviews were conducted individually with each software engineer at the end of each evaluation. Before each interview, we clarified that the purpose of the interview was to evaluate the approach and propose improvements to it. We made the interviewee aware of the importance to detail the answers as much as possible.

The questionnaire described in Appendix E was used to guide the interviews. It has 34 questions (Q) addressing the following aspects:

- Experience of the Interviewee (Q1 to Q5);
- Learning in using the RSD approach (Q6 to Q8);
- Content and structure of the RSDs (Q9 and Q14);
- Effects of the RSD approach on works of the team (Q15 to Q17);
- Dependence between stakeholders (Q18 to Q19);
- Effort required to specify, coding and testing using the RSD approach (Q20 to Q27);
- Impact analysis of change requests (Q28 to Q30);
- Opportunities for Improvement the RSD approach (Q31 to Q34).

As previously described, the content of each RSD was evaluated only by the software engineer responsible for its coding. Unlike, the RSD structure was evaluated by the entire team. During the interviews, the RSD structure was evaluated according to some quality factors of the model described in Chapter 5 (Section 5.5): *Simplicity*, *Acceptance Criteria*, *NFR*, *Technical Aspects*, *Functional Requirements*, *Consolidated/Fragmented Information*, *Automated Support*, *Traceability* and *Change History*. As described in Section 6.1, these last three factors are not address by the RSD practices. These factors will be addressed by the tool to be developed as future work of this thesis. However, they were also investigated in order to identify their importance in the context of the projects investigated. Each factor was assessed as being in *Compliance*, *Partial Compliance* or *Non-Compliance*.

The interviews were transcribed and open coded (MERRIAM, 2009). The constant comparison method was used to synthesize the data and to explain how the RSD approach works in practice. After coding of the transcriptions, a review of the codes was performed trying to identify similarities, duplicates and undue codes. The codes were successively revised until getting the results presented in next section.

7.4 Results

This section presents the results of the evaluation of the RSD approach conducted in each company.

7.4.1 *First Evaluation*

a) Context

The RSD approach was adopted since the beginning of the project. Each sprint lasted for a month, but partial versions of the software were released every week for the internal acceptance testing. Several agile practices were well established in the project, such as backlog, iterations, frequent releases, version control, continuous integration, refactoring, automated build and retrospective (AGILE ALLIANCE, 2016). Daily meetings were conducted to monitor the project status, to share better practices, and to discuss the problems and difficulties faced in the development process.

The project used different people to play the roles of analyst and developer. The development team was composed of 10 software engineers: 8 developers and 2 system analysts. One of the analysts and six developers worked together on other projects. However, one of the analysts and two trainees (developers) were novice in the team. Except for the two trainees, all other team members had more than four years of experience with ASD. All developers have already played the role of system analysts in other projects, as well as both analysts have already worked as developers. The most experienced developer played the roles of architect and configuration engineer. He was responsible for conducting deploys weekly.

The analysts elaborated the RSD in collaboration with the PO and the customer during the weekly discovery sessions. The Microsoft Word was used to elaborate the RSD. The Java language was used to code the requirements. The PB was controlled through a Microsoft Excel spreadsheet. *Redmine* tool was used for the project management and to report the results of acceptance tests. The analysts used *Pencil* tool[18] for modeling the mockups and *Astah* tool[19] for the conceptual modeling. The evaluation results are presented as follows by research questions.

b) RQ1: How the team evaluates the SRS produced using the RSD approach?

The development team evaluated the RSD from two aspects: content and structure (type and organization of information).

Content

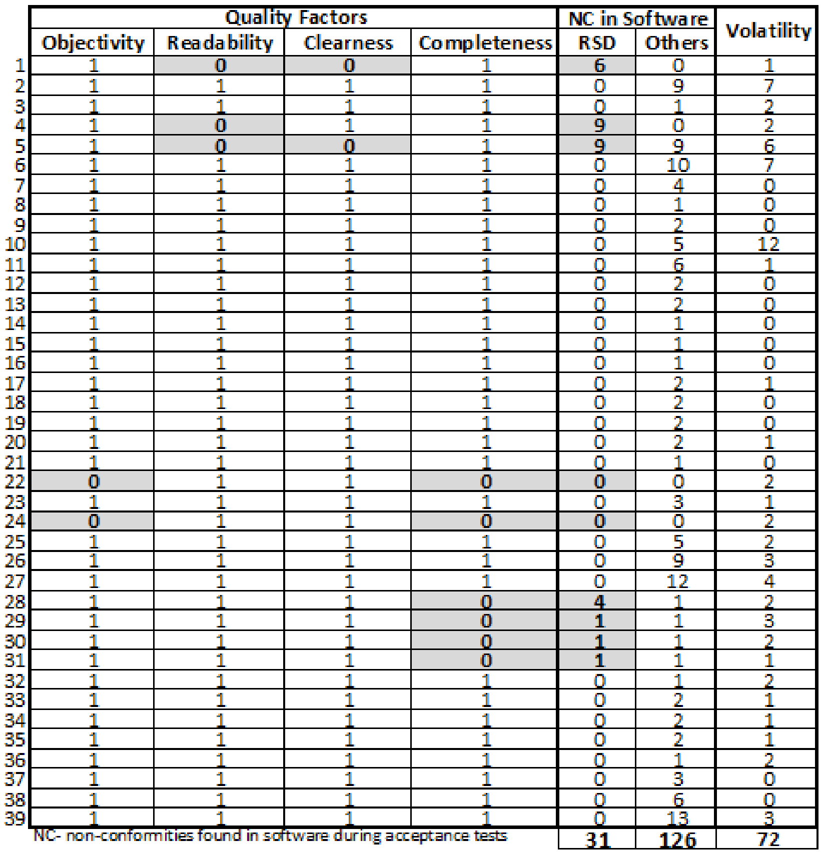
The developer evaluated all 39 RSD and 257 AC+ produced during the empirical study, as summarized in Table 7.2. Some RSDs specified more than one requirement. All RSDs were evaluated as in compliance with *Team-oriented* and *Updated* (Outdated) factors, but they were not included in Table 7.2.

The first four columns show the evaluation performed by the developers according to the quality factors. The Non-Conformities (NC) found by PO in the acceptance tests are shown in the fifth column (NC related to RSD failures) and the sixth column (other types of NC). The last column shows the quantity of changes performed in each RSD (volatility) in order to fix or improve it and meet the needs of stakeholders. The change history was listed on the back cover of each RSD.

[18] http://pencil.evolus.vn/

[19] http://astah.net/editions/professional

Table 7.2 – Content evaluation of each RSD



	Quality Factors				NC in Software		Volatility
	Objectivity	Readability	Clearness	Completeness	RSD	Others	
1	1	0	0	1	6	0	1
2	1	1	1	1	0	9	7
3	1	1	1	1	0	1	2
4	1	0	1	1	9	0	2
5	1	0	0	1	9	9	6
6	1	1	1	1	0	10	7
7	1	1	1	1	0	4	0
8	1	1	1	1	0	1	0
9	1	1	1	1	0	2	0
10	1	1	1	1	0	5	12
11	1	1	1	1	0	6	1
12	1	1	1	1	0	2	0
13	1	1	1	1	0	2	0
14	1	1	1	1	0	1	0
15	1	1	1	1	0	1	0
16	1	1	1	1	0	1	0
17	1	1	1	1	0	2	1
18	1	1	1	1	0	2	0
19	1	1	1	1	0	2	0
20	1	1	1	1	0	2	1
21	1	1	1	1	0	1	0
22	0	1	1	0	0	0	2
23	1	1	1	1	0	3	1
24	0	1	1	0	0	0	2
25	1	1	1	1	0	5	2
26	1	1	1	1	0	9	3
27	1	1	1	1	0	12	4
28	1	1	1	0	4	1	2
29	1	1	1	0	1	1	3
30	1	1	1	0	1	1	2
31	1	1	1	0	1	1	1
32	1	1	1	1	0	1	2
33	1	1	1	1	0	2	1
34	1	1	1	1	0	2	1
35	1	1	1	1	0	2	1
36	1	1	1	1	0	1	2
37	1	1	1	1	0	3	0
38	1	1	1	1	0	6	0
39	1	1	1	1	0	13	3
	NC- non-conformities found in software during acceptance tests				31	126	72

0 (non-compliance) 1(compliance)

Eleven RSDs (28%) presented non-compliance with at least one quality factor. Most of RSDs (72%) were evaluated in compliance with all factors. A summary of the evaluation by quality factor is presented in Figure 7.1. According to the developers, 95% of RSD (37) was objective. Two RSDs were considered non-objective because they had redundant information, unnecessary for the developer. Despite this, the acceptance tests did not present any NC in the functionalities related to these two RSDs. According to the interviews (Question #22 in Appendix E, Q22), the analysts' experience contributed to writing the AC+ in an objective way for the developer.

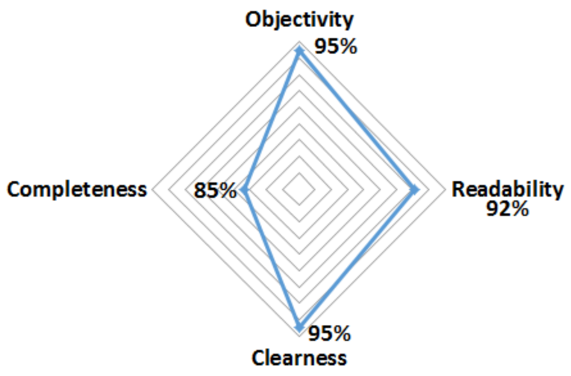


Figure 7.1: Summary of the content evaluation by quality factor

Most of RSDs (92%) were considered easy to understand by the team. *Readability* factor was considered compromised in three RSDs that described more than one requirement in the same document. For example, the requirements *Solicitation of a Doping Kit*, *Search of Solicitations*, and *Approve a Solicitation* have been described in the same document. In the opinion of developers (Q32), each requirement of the product backlog should be specified in a separate RSD.

The requirements were described clearly (without ambiguity) in 95% (37) of the RSDs. Two RSDs were considered in non-compliance with the *Clearness* factor. One RSD did not clearly define how the data access restriction should be done. In the other RSD, it was not clear whether the data should be persisted in the database or not. Two RSDs considered ambiguous were also evaluated as non-compliance in *Readability* factor because they had more than one requirement described in the same document. Regarding the 257 AC+ specified during the period of the empirical study, only 6% (15) presented problems of lack of clarity.

Regarding the *Completeness* factor, 85% of the RSD were considered sufficient for the developer to code, without the need to consult complementary sources. Six RSDs were evaluated as incomplete due to lack of information needed to code some feature. We observed that the two RSDs that had prolix descriptions unnecessary for the developer were also evaluated as incomplete. Thus, the descriptions with excessive detail did not contribute to make a complete documentation. Although the acceptance tests related to these RSDs did not find NC in the software, the lack of objectivity and completeness of these RSDs affected the productivity of the team given that the developers needed to talk with other stakeholders or to look for other sources to clarify related doubts, as reported by one developer (Q21):

> *"Sometimes we identify that the SRS is incomplete, lacks some AC+. So, we lose a lot of time because we need to get the information with the analyst or wait for the documentation to be updated."*

In the interviews, we inquired the ten software engineers about the content of the RSDs compared to the content of the SRSs used in other projects (Q13). User stories and use cases were cited by more than 70% of respondents as techniques that they had used previously. Regarding the *Completeness* factor, nine software engineers assessed RSD as better, just one considered as worse. Regarding the *Clearness* factor, 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. All respondents pointed out that the *Objectivity* and *Readability* of RSD is better than when using other approaches.

Finally, we analyzed the changes (volatility) that were made in RSDs to meet the needs of stakeholders. A total of 72 changes were identified, some RSDs were changed more than once, as shown in Table 7.2 (last column). A large number of changes do not mean that the RSD became more complete, nor that the software had little NC in the acceptance tests. For example, RSD #5 had many changes, but also presented a lot of NC in the software. Although some changes were made in the RSD #29, they were not enough, since the developer has assessed it as incomplete. In such requirements, it is possible that the team did not have the necessary understanding of customer needs. Although many changes have been made in the RSD over these requirements, the RSD was not considered complete and produced a lot of NC. Thirteen RSDs (33%) did not have their content changed.

Structure

The software engineers also were interviewed about how they assessed the RSD structure based on mockups, concepts and AC+ (Q9). Then, they evaluated the RSD using a scale from 1-*Inadequate* to 5-*Very Adequate* (Q10). In general, the structure of RSD is *Very Adequate* in the opinion of most respondents (8). The remaining respondents evaluated it as *Adequate*. Then, the team was inquired about each quality factors (Q11). All ten software engineers pointed out that the structure was in compliance with the *Simplicity*, *Acceptance Criteria*, *Technical Aspects* and *Functional Requirements* factors, as summarized in Figure 7.2.

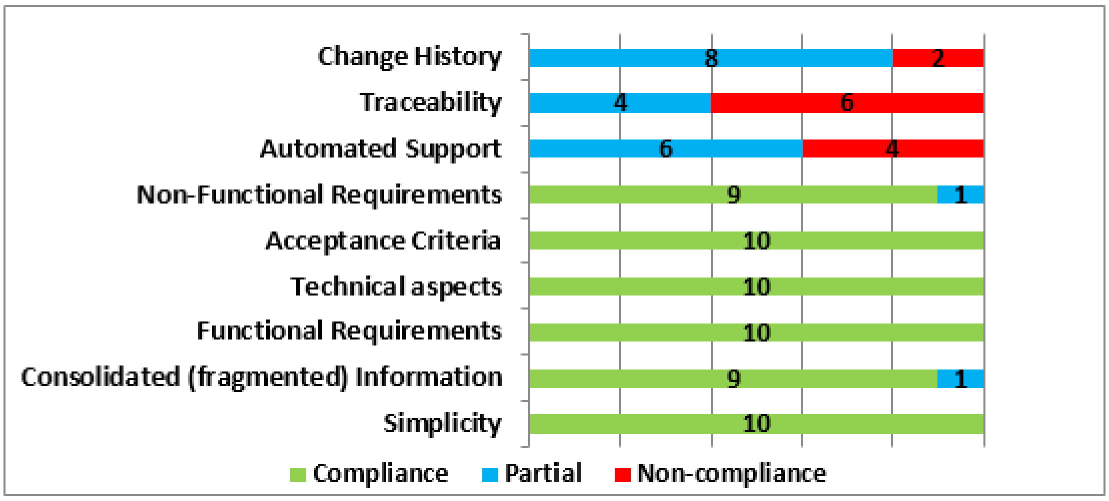


Figure 7.2: Evaluation of RSD structure by quality factor

One system analyst evaluated the *Non-Functional Requirements* factor as in *Partial Compliance* (Q11). He suggested (Q31) the definition of an initial catalog of the NFRs to improve the adoption of the RSD approach, as follow:

> *"Although the structure predicts the NFR type to describe acceptance criteria, in practice, the team generally neglected the specification of these requirements. The project repository had few acceptance criteria of this type. The approach could suggest a catalog with generic NFRs to be part of the initial repository of each project."*

As described in Chapter 6, the RSD approach does not restrict the adoption of a particular tool for the specification of AC+, construction of mockups and the conceptual model. It is up to each company to use the tools according to their needs, processes, and methodologies. The development team reported (Q17) no problems regarding the use of the *Pencil* for the elaboration of mockups and the use of *Astah* to conceptual modelling. However, the team reported difficulties regarding the use of Microsoft Word to store and control the repository of AC+, and to elaborate the RSD.

In the opinion of system analysts (Q17), the operationalization of the repository through a text document does not facilitate the search for AC+ to be reused. Moreover, traceability has been undermined because the relationship between requirements and acceptance criteria was made manually using an MS Excel worksheet instead of being generated automatically. Thus, two analysts and four developers (6) pointed out non-compliance with the *Traceability* factor and partial compliance with the *Automated Support* factor, as shown in Figure 7.2.

The changes control in requirements was done through MS Word and by the SVN tool used for version control. Two developers pointed out non-compliance with the *Change History* factor. The remaining of the team evaluated it as partially compliant. A developer reported partial compliance with the *Consolidated (Fragmented) Information* factor. In his opinion (Q13), the RSD should provide a hyperlink to access each AC+ more quickly given that they were defined in a separate document.

Finally, we investigated the structure of RSD compared with other approaches previously used by the software engineer (Q12). Six interviewees pointed out that the RSD structure is more appropriate than the structure of the other approaches. No interviewee indicated that the RSD structure was worse than other approaches. In the comparative evaluation by quality factor (Q13), the majority of interviewees considered that the RSD structure is better in *Simplicity, Acceptance Criteria*, *Technical Aspects* and *Consolidated (Fragmented) Information* factors, and equal in the others factors: *NFR*, *Automated Support*, *Traceability*, *Functional Requirements* and C*hange History*.

The initial RSD structure did not represent the AC+ priority. If an AC+ had a different priority or it had to be implemented in another sprint, the analyst registered this information as a note in Redmine. After the second month of use of the approach, the team modified the RSD structure in order to highlight the priority of the acceptance criteria in the RSD itself (Q31). One developer pointed out this as a limitation of the approach (Q21), as follows:

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

The AC+s were categorized by the system analysts in *General* (potential to be reused in other requirements) and *Specific* to a particular requirement. This categorization was not part of the structure initially defined for RSD. However, analysts adopted this practice to facilitate the search for AC+ in the RAC and improve the reuse (Q22/Q33). In fact, the priority and categorization of the AC+ improve the RSD structure and therefore has been incorporated into the RSD approach after the first case study, as defined in metamodel presented in Section 6.2.

The results of the interviews (Q14) showed that RSD met the expectations of the software engineers regarding the structure of the RSD, 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."*
>
> *"The RSD structure allows a more integrated view of the rules and constraints applied to a particular requirement. In previous projects, user stories were described independently and unrelated; we did not have an integrated view of a particular requirement."*

c) RQ2: How the RSD approach affects the team work?

The model outcomes defined in Section 5.3.2 were used as reference to guide the analysis on how RSD approach affects the work of the team. The results of this analysis are presented below:

Effort required to specify, code and testing

The RSD approach was quickly absorbed by the team that presented no difficulties in their learning (Q6). The RSD approach introduces new practices to specify requirements 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 (Q24 to Q26), 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. Half of developers considered that RSD required either less effort to code than other approaches. The other half considered that it requires the same amount. According to most of the software engineers (7), implementing RSD requires a reasonable effort; the other software engineers (3) consider that implementing from RSD requires little effort (Q23).

The interviews results showed that the approach met the expectations of the developers regarding their productivity and proved to be a very simple and objective SRS, suitable for coding activities, as follows (Q27):

> *"The detailing of the requirements through the AC+ makes the SRS more straightforward and objective for the developer."*

According to the analysts (Q21), the Traceability Matrix (TM) was not useful in the project due to the high effort required to keep it up-to-date. The TM presents the relationship between requirements and AC+, so it helped in the impact analysis of change requests in requirements. The matrix is not an artifact defined by the RSD approach. It was adopted by the company to manage the requirements. In parallel to drawing up the RSD, the TM had to be updated manually. The same happened when the team needed to change an AC+ and/or a requirement. We identified that the TM was almost always out of date. Only ten requirements were updated in TM. Some tools available in the market, such as *Caliber*, generate the traceability of requirements automatically. However, the project did not use any of these tools. Instead, the TM was operationalized by using a spreadsheet shared among 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 SRS."*

The acceptance tests were conducted by the PO using the RSD as a reference. This was pointed out as a good practice (Q22). The tester validates whether the software complies with each AC+. However, the analyst had to register manually in *Redmine* each NC found, and forward it to the responsible person for the fix. Sometimes, the analyst did not register the AC+ related. As a result, the developer lost a long time analyzing the problem, and clarifying doubts.

Analyzing the difficulties reported by the team during the acceptance tests and from the interview results (Q15), we believe that the test can be optimized by using a tool to generate a roadmap (checklist) automatically from AC+. The tester needs only to check or uncheck the checklist, according to the test result. In addition to making the register of NC by associating the AC+ related, the tool can send an email notification to the responsible to make the necessary corrections. The software adoption will optimize the time of the team. Furthermore, the use of the checklist prevents the tester to forget validating an AC+, thus contributing to improve the testing quality.

Non-conformities in the Software due to issues in the SRS

As previously mentioned, the acceptance tests were conducted weekly to verify the compliance of the software with customer needs. At the beginning of the project, the stakeholders did not have a complete understanding of all customer needs given that the software development was done iteratively and incrementally following the agile principles. The customer's needs were obtained and detailed through partial validations of the software in each sprint. From the analysis of the *Redmine* records, six types of NC in the software were identified as shown in Table 7.3.

Table 7.3 – Types of Non-Conformities (NC)

	ID	Type of Non-Conformity
Bugs	#1	Problems of legibility, clarity or completeness in RSD
	#2	Customer's needs not represented correctly in the RSD
	#3	Requirement not implemented according to RSD
Improvements	#4	New requirements not previously captured but essential for the customer
	#5	Additional features in the requirements already validated by the customer
Changes	#6	Changes in the requirements already validated by the customer

As detailed in Table 7.2, 157 NCs were found during the case study period. 20% (31) of these NC+ were due to issues in RSD (Table 7.3, items #1 and #2) which required corrections in it. The other NCs (126) are not related to RSD failures. They refer to bugs arising from the implementation of the requirement differently from what was defined in the RSD (#3), improvements (items #4 and #5), and changes in functionality previously requested and already validated by the customer (#6).

It was not possible to assess whether the use of the RSD approach reduced the NCs resulting from problems in the SRS because the acceptance tests were conducted in a different way in the previous projects. Another document (test cases) was used to guide the tests before the adoption of the RSD approach. Moreover, the test results were grouped by tested functionality, and a single issue was recorded in *Redmine* describing all NCs found. After the introduction of the approach, the company decided to conduct the tests from the AC+ and registering the result by it, instead of grouping by functionality.

Impact analysis of change requests

The requests for improvements or changes in requirements were analyzed by the PO, the analyst, and the developer responsible for the coding. They discuss the impact on the project schedule. The TM was not used by them to support the impact analysis of change requests because it was outdated most of the time (Q29).

Depending on the impact of the request, new prioritizations and distributions of requirements were made for the next sprints. However, the initial duration of the sprints was always maintained. It was not modified as a result of the requests. During the case study, the project received a total of 54 change requests – 69% of RSDs (27) received change requests. These changes were requested during acceptance tests or at weekly customer meetings.

The changes in the RSDs were not appointed as a negative factor. According to the team (Q28, Q30), despite the large number of change requests, the impact was reduced given that the software validations were frequent and the project architecture was scalable. The team pointed out that the conceptual modeling was a positive factor that contributed to the construction of a more stable data model. So, little changes were required in the structure of the data model during the sprints, which reduced the impact of the rework in software.

Reuse

The team reported that due to the TM being outdated, they did not use it to identify AC+ that could be reused (Q16). At the end of the case study, the TM was updated to examine the reuse rate of the AC+. Many AC+ were reused in several requirements, especially the AC+ of validation, technical, and interface types, resulting in a reuse rate greater than 60% which improved the productivity, as follows:

> *"The RSD approach has contributed to the reuse of AC+ used in other requirements, which improves our productivity."*

By analyzing the TM, it was possible to identify that reuse depended largely on knowledge about the existence of AC+s. There were high reuse rates of the AC+s specified by the same analyst. However, the reuse rate of the AC+ specified by different analysts was much lower. Analysts have reported difficulties to find opportunities for reuse because AC+s were stored as a text document (Q21). A more efficient mechanism is required to share and find AC+s. The support of a tool can increase reuse rates. Besides storing the AC+s in a database, a tool could provide features to locate them more efficiently. Despite the difficulties previously presented, the analysts consider that the reuse of AC+s improved their performance (Q22). Table 7.4 shows the top five most reused AC+. AC90 was the most reused, it was used in 82% of requirements.

Table 7.4 – Excerpt of the most reused Acceptance Criteria+

ID	Description of the AC+	% reuse
AC90	The widget is read-only. It cannot be changed by the user.	82%
AC04	Only active records must be displayed	79%
AC03	To save, it is necessary that all required fields (*) are filled.	74%
AC20	The drop-down list must only display the confederations that the user logged has access permission in your profile.	74%
AC44	It is necessary to include the log by registering the user logged, the current date\time and the operation performed.	62%
...	...	...

Knowledge transfer, Customer collaboration and Dependence between stakeholders

During the the case study, the trainees were replaced twice. The RSDs were used to reduce the effort to transfer knowledge. So, the rest of team did not have to interrupt their activities frequently to impart knowledge to new members. Although the team held daily meetings, the RSD was the main source for developers to code the requirements.

The customer lived in a different city from the development team. Despite this, the customer collaboration was adequate in the team opinion (Q18). Communication with the customer was made on demand through the analysts or PO whenever it was necessary to detail or clarify any requirements. The analysts and developers had a frequent iteration. In the opinion of the team (Q19), the frequency and type of communication used by stakeholders were very adequate.

7.4.2 *Second Evaluation*

a) Context

The company has adopted Scrum since 2013 using practices such as product backlog, planning poker, frequent prioritization of requirements, retrospective, iterations and daily meetings. The development team consisted of four members: two software engineers, with more than three years of professional experience, and two trainees (university students). All members worked in the same activities, there were no different roles. They were responsible for the evolution and maintenance of three applications, one of which is the company's main product that is in operation in more than forty customers.

Before the company adopts the RSD approach, the requirements were specified through stories. In sprint planning, if a story had an estimate of effort greater than 13 points, it was necessary to divide it into two or more stories. Each user story was registered as a ticket in *Redmine*. The dependency relationships between the user stories were not registered although the *Redmine* have a resource for this purpose.

Regarding the adoption of a new approach to specify requirements, software engineers had different (but not conflicting) expectations. The most experienced software engineers pointed out that the management of requirements needed to be improved. The tickets (user stories) were closed after the validation of the software with the customer. If it was necessary to modify some functionality (e.g., an interest calculation), the team registered a new ticket describing how the new calculation should be, instead of finding and changing the original story.

Trainees often needed to consult senior engineers for additional technical information not described in the stories. Therefore, they pointed out the need to have a more complete SRS containing the information needed to code. The trainees also pointed out that they had difficulty in having a general understanding of a requirement because the user stories provided only a limited view of a functionality.

From October 2016, the RSD approach has been introduced in one of three development teams to replace the user stories. The approach was used to specify new requirements and change the existing requirements in the three applications. However, the user stories have been preserved for the team to consult them when necessary. The team continued using the same tools that used before the introduction of the approach. The *Redmine* tool was used to control the product backlog and elaborate the RSD. In this context, the practice of conceptual modeling did not require much effort from the team. Usually, the user interface layout was automatically generated by the ScriptCase tool[20] from the data entities. The layout was only modified after user validation, if necessary. Thus, the practice of modeling mockups had a secondary importance. The specification of the AC+ was the most commonly used practice and the PHP language was used to code the requirements.

The sprints had monthly duration. The sprint planning considered the product backlog of the three applications, since the team was the same. Most of the AC+ were specified by the team during the sprint planning. Sometimes, other AC+ were specified later by the developers.

a) RQ1: How the team evaluate the SRS produced using the RSD approach?

Content

During the three months of the case study, the team worked in 68 requirements. Most of them (51) already existed previously. The team worked in their evolution. The remainder (17) represented new requirements included in the product backlog of the three applications after the adoption of the RSD approach.

The conceptual modeling was necessary in 26 requirements. The modeling of mockups was necessary in 9 requirements. In the other requirements, no mockup was associated with RSD because the team used the user interface generated by *ScriptCase*. 122 AC+ were specified by the team during the investigation period. Most of them described new needs of stakeholders (96). The remainder represented changes in functionalities previously described through user stories (26). Sometimes during coding activity, the software engineer identified new *System*-type AC+ and registered them in the RSD. This occurred in 26 RSDs.

The user stories previously used by the team were written in a very objective and clear way directed to the development team. According to software engineers, these characteristics remained present in the requirements described using the RSD approach. Software engineers perceived that the RSD approach has improved the completeness and readability of the SRSs. The SRS became more complete by the description of the AC+ with technical aspects that were not previously described us-

[20] http://www.scriptcase.net/

ing user stories. These information were not documented, they were only in the minds of the most experienced software engineers.

According to the team, the grouping of AC+ by functionality provided a consolidated view of a requirement facilitating a general understanding of it (Q14). The readability of the RSDs was considered better than the user stories.

Analyzing the changes made in the RSDs (volatility), we observed that only three RSDs were corrected during the period of investigation. Regarding the *outdated* factor, the team considered that the RSDs were up to date.

Structure

Each RSD was represented in *Redmine* describing a requirement as a ticket and the AC+ as sub-tickets. Mockups were attached to the ticket as a file, if necessary. The association between the widgets, data entities and AC+ was made in the description field of the ticket. The team relaxed this association that was only made when there were new widgets, data entities or when there was some related AC+, rather than describing all widgets and data entities, as defined by RSD approach.

The *Redmine* was customized to classify the AC+ according to the potential to be reused (*Specific* or *General*) as suggested in the first case study and incorporated into RSD approach (Section 6.2) and to identify the AC+ types (Table 6.1). However, the team unified the *Validation*, *Interface* and *Technical* types into a single type called *System* to facilitate the adoption of the approach in practice (Q31). AC+ of the *System* type decreased the dependency among the team members helping to meet some expectations of the trainees (Q14). To facilitate the planning and coding activities (Q31), the team extended the approach by creating a new field (*Order*) in *Redmine* to identify the order that the AC+ should be coded within the sprint.

The *Technical Aspects* and *Acceptance Criteria* factors were evaluated as *Very Adequate* (Q10). In the opinion of the team (Q13) regarding these factors, the structure of the RSD approach has improved over the previously used approach. Despite the *Simplicity*, *Functional Requirements* and *Consolidated* (*Fragmented*) *Information* factors have been evaluated as in compliance (Q11), the team considered that there was no improvement over the previously used practice (Q13).

NFRs such as security, accessibility, and storage constraints were defined only at the beginning of projects. The team reported not being able to evaluate the NFR factor because no NFRs were specified during the investigation period.

The change control in the RSDs was done through *Redmine*. The *Change History* factor was evaluated as partial compliance (Q11). The team pointed out that the *Redmine* query to view changes made in the RSDs only had the option to filter by date. It was not possible to search the AC+ that had the only the description field modified, for example.

The structure of the approach was considered non-compliance in the *Traceability* factor because the relationship between the RSDs had to be done manually. In addition, *Redmine* did not provide any report showing the relationships between RSDs, neither relationships between requirements and AC+. Table 7.5 summarizes the assessment of the approach according to the quality factors and compared to the previously used approach.

Table 7.5 – Evaluation of the RSD approach by quality factor – second evaluation

	Compliance	**Non-Compliance**	**Partial Compliance**	**Not assessed**
Better	Acceptance Criteria, Technical Aspects, Completeness, Readability	-	Change History, Automated Support	-
Equal	Team-oriented, Simplicity, Functional Requirements, Objectivity, Clearness, Outdated (updated), Consolidated Information	Traceability	-	-
Worse	-	-	-	-
Not assessed	-	-	-	NFR

b) RQ2: How the RSD approach affects the work of team?

Effort required to specify, code and testing

The grouping of the AC+ by requirement made it easier to find the specification of a feature to be changed (Q20). As a consequence, the team modified the strategy to specify changes in the existing requirements (Q28). Instead of including a new ticket to specify a change as it was previously done, the new strategy dictated that the software engineer must locate the original ticket and make the change in the related AC+ or include a new AC+. In addition, the AC+ were not finalized after the software validation, they remained active throughout the software lifecycle. This new strategy improved the management of requirements.

In the opinion of trainees (Q26), the description of the AC+ with technical aspects (*system* type) reduced the effort to codify the requirements because they did not need to interrupt the team activities to obtain missing information in the SRS.

Previously, the PO conducted the acceptance tests in an exploratory and ad-hoc manner. The adoption of the RSD approach systematized the tests from the AC+, reducing the required effort.

Non-conformities in the Software due to issues in the SRS

Although the AC+s have helped in performing the tests, the team pointed out that the testing activity still is not performed properly in the company and needs to be improved (Q32). Often, features are made available to the customer only with internal tests performed by the developer who coded the functionality, without the PO performing the acceptance tests. In these cases, the developer fixed the identified NCs without registering them. Therefore, we cannot investigate the NCs in the software resulting from problems in the RSDs because there was no recording of these NCs.

Impact analysis of change requests

The impact analysis of changes was done by the PO and the senior software engineer. As a result of this analysis, the priority of the requirements and the AC+ was rescheduled, if necessary.

As previously mentioned, the projects had no information on the traceability of requirements and AC+. However, the PO pointed out that the absence of this information did not affect the analysis because he and the senior engineer had a thorough knowledge of the applications.

Reuse

We observed that the percentage of the AC+ reused was very low, less than 15%. As previously mentioned, the projects were already in progress when the approach was introduced, so the focus was on the evolution of applications. Often, several constraints were grouped into one AC+ rather than each AC+ representing only a single constraint. This made it difficult to reuse AC+.

As stated earlier, the team did not have good visibility on the relationship between the AC+ and the requirements. Perhaps, access to this kind of information could help software engineers identify and reuse AC+ used in similar requirements.

It is necessary to investigate over a longer period and in new projects to have a more effective evaluation if the RSD approach can contribute to the reusability of AC+ in the company.

Knowledge transfer, Customer collaboration and Dependence between stakeholders

Although the AC+ of the *system* type has reduced stakeholder dependence and the effort required to codify a requirement, in the opinion of trainees (Q19), daily meetings continued to be essential to clarify doubts related to AC+ of the *business* type and to share difficulties with other team members. In the opinion of the two most experienced software engineers, the RSD was the main source of information they used to code a feature.

According to the team (Q18), the communication with the customer was adequate. All team members interacted with the customer whenever necessary, or at least once a week. The most frequent communication was via email, telephone, and instant messaging applications. The RSD was not used to validate requirements with customers. The validations were always done through frequent software deliveries.

7.5 Discussions

Table 7.6 summarizes the context of the two empirical studies which evaluated the RSD approach.

Table 7.6 – Context of the empirical studies that evaluated the RSD approach

	Empirical Study 1	**Empirical Study 2**
Company Size	Small	Small
Period Investigation	12 months	3 months
Projects Investigated	1	3
Type of Software Developed	Information System	Information Systems
Project stage when approach was introduced	The project was still in the planning stage	The projects were already in progress and working on more than 40 customers
Development Team	10 software engineers (2 trainees)	4 software engineers (2 trainees)
Roles	System Analysts (2) and Developers (8)	There was no division of roles
Agile practices used	Backlog, frequent releases, iterations, version control, continuous integration, automated build, refactoring	Backlog, planning poker, frequent prioritization of requirements, iterations, version control, retrospective, daily meetings
RSD practices	Use of the three practices equally (Modeling Concepts, Modeling Mockups and Specify AC+)	Focus on the specification of AC+
Tools used to specify the requirements	MS Word	Redmine
Sprint Duration	Monthly	Monthly
Internal Releases	Weekly	Monthly
Acceptance Tests	Weekly	Not frequent

Some characteristics are common to both companies, such as size, type of software developed and sprint duration. However, there are some differences in the context of the two assessments conducted. In the first company, the RSD approach has been used since the beginning of the project and for a longer period (12 months). On the other hand, the second company introduced the approach in three projects that were already in operation in more than 40 customers and the evaluation was carried out only for three months. We can highlight other differences such as the tools used to specify requirements, the RSD practices used, the team size, and the roles within the development team.

Despite the differences in the context of the two studies, we identified some similarities. Table 7.7 summarizes how the teams evaluate the SRS produced using RSD approach (RQ1) according to the evaluations conducted in the two companies. The table was organized according to the categories of the model defined in Chapter 5 (Table 5.4) to facilitate the analysis of the results in relation to the model. Some factors had a positive (+) effect, improving the team performance and others had a negative (-) effect, undermining the team performance.

Table 7.7 – How the teams evaluate the SRS produced using the RSD approach

	id	Factors	Empirical Study 1	Empirical Study 2
Structure	#1	Integration of the AC+, Mockups and Concepts	+	+
	#2	Grouping AC+ by functionality	+	+
	#3	AC+ describing system and business requirements	+	+
	#4	Inadequate support to register the acceptance tests by AC+	-	-
	#5	Inadequate support to track the AC+	-	-
	#6	Representation of AC+ Priority	-	
Content	#7	Readability	+	+
	#8	Completeness	+	+
	#9	Clearness	+	
	#10	Objectivity	+	
	#11	RSD describing multiple requirements	-	
	#12	AC+ grouping several constraints		-
Others	#13	Reuse of AC+	+	
	#14	Difficulty finding the AC+ in the repository	-	
	#15	Acceptance Tests from AC+	+	+
	#16	Learning curve to use the RSD approach	+	+
	#17	Adapting the RSD approach to the company process	+	+

+ (positive effect) - (negative effect)

Regarding the structure of the approach, in the two empirical studies, the integrated view of AC+, mockups and data entities (Table 7.7, #1) and the grouping of AC+ by functionality (#2) improved the readability of RSD (#7) and the understanding on how to code a requirement. Another positive factor was the use of AC+ describing the business and system requirements in an integrated manner (#3), which left the SRS more complete (#8) and targeted to the developers.

Although the AC+ contributed to systematize and improve the performance of the acceptance tests (#15), the two companies presented difficulties in registering the test results (#4) without the support of a specialized tool. Another negative factor was the inadequate support to track the AC+ (#5).

The teams did not have difficulties in using the RSD approach. The learning was very fast (#16). The conceptual modeling and the modeling of mockups were already practices utilized in both companies but were not integrated into the user stories. The innovation in the two companies was the specification of the AC+ and their integration with the mockups and data entities.

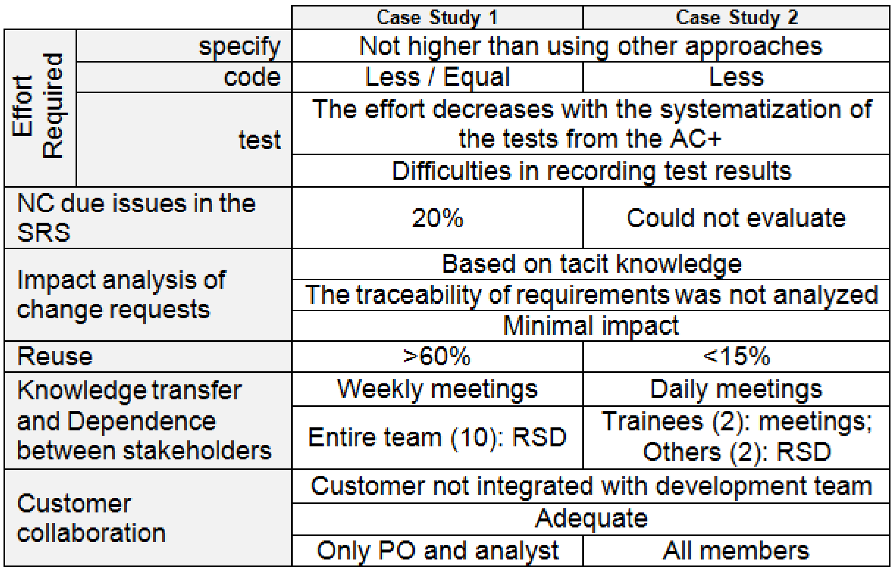
In the first study, the team extended the approach to represent the priority of AC + that was not represented in the initial structure of the approach (#6). And to facilitate reuse, the team categorized the AC + into *Specific* and *General*. These improvements were incorporated into the approach, so they were not reported in the second case study. The team from the first empirical study also created new notations to represent mandatory and read-only widgets. In the second empirical study, the association between the AC+, widgets and data entities was relaxed and was only made when it was necessary to describe some AC+ or if a new data entity or widget was identified. This flexibility in adapting the approach to the development process used by the company (#17) was also pointed as a positive factor.

The other characteristics pointed out in Table 7.7 were only identified in one of the studies. For example, the description of the requirements in a clear (#9) and objective (#10) manner was highlighted in the first empirical study as a factor that has contributed to improve the developer productivity. Although these features were also present in the RSDs of the second empirical study, the team pointed out that they were also present in the user stories previously used.

In the first empirical study, legibility was compromised when several requirements were described in the same RSD (#11), which undermined the productivity of the developers. The specification tool (*Redmine*) used in the second empirical study avoided this problem because each requirement was described as an independent ticket. On the other hand, in the second empirical study, often the AC+ grouped several constraints (#12), rather than describing each constraint as one AC+ in a singular way with a binary result as specified in the first empirical study.

The reuse of AC + (#13) was pointed out as a factor that contributed to improve the productivity of the team in the first empirical study. However, the reuse depended largely on knowledge about the existence of AC+s. There were high reuse rates of the AC+s specified by the same analyst. The reuse rate of the AC+ specified by different analysts was much lower because the Analysts had difficulties to find AC+s in the repository operationalized by a text document (#14). A more efficient mechanism was required to share and find AC+s. This problem was not reported in the second empirical study because the reusability of requirements was very low. Table 7.8 summarized how the RSD approach affects the work of the teams (RQ2).

Table 7.8 – How the RSD approach affects the work of the teams



		Case Study 1	Case Study 2
Effort Required	specify	Not higher than using other approaches	
	code	Less / Equal	Less
	test	The effort decreases with the systematization of the tests from the AC+	
		Difficulties in recording test results	
NC due issues in the SRS		20%	Could not evaluate
Impact analysis of change requests		Based on tacit knowledge	
		The traceability of requirements was not analyzed	
		Minimal impact	
Reuse		>60%	<15%
Knowledge transfer and Dependence between stakeholders		Weekly meetings	Daily meetings
		Entire team (10): RSD	Trainees (2): meetings; Others (2): RSD
Customer collaboration		Customer not integrated with development team	
		Adequate	
		Only PO and analyst	All members

7.6 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 another researcher (master student) checked the results. This was the main strategy for increasing credibility.

The subjectivity inherent in categorizing and classifying the factors that affected the SRS based on the team's perception was another threat. The constructs defined in the model (Section 5.3.1) were used to minimize the impact of subjectivity.

The approach could have been evaluated by other researchers or by using another research method. These might 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 empirical 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.

Another threat was the software engineers answer what 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 to improve it and the work of the team. The researchers had no personal or professional relationship with the software engineers. It was reinforced during the interviews 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, avoiding direct, vague and superficial responses. Thus, we tried to get arguments, examples, and details to help understanding the perception of each software engineer.

7.7 Summary

The goal of the empirical studies was to evaluate the use of the RSD in practice and identify 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.

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 approach produces an SRS that is closer to what will be implemented and allows technical aspects to be represented in an integrated manner with the functional requirements. The feedback collected through interviews suggests that RSD does not add extra effort or may even help to reduce the effort involved in coding, testing and maintenance. The performance of the tests from the AC+ was also pointed out as a positive factor.

Regarding the challenges using the RSD approach, the productivity of the teams was compromised when several requirements were specified in the same

RSD or when an AC+ was grouping several constraints. The mechanism to search the AC+ in the repository is inefficient, which makes reusability difficult. The teams also pointed out the inadequate support to track the AC+ and to register the acceptance tests by AC+.

The evaluations showed some improvement opportunities in the RSD approach like prioritization and categorization of the AC+, and the need of an automated support to elaborate the RSDs. Some of these improvements have already been incorporated in the approach, and the others were allocated for future works, as detailed in Chapter 8 (Section 8.3).

A paper detailing partial results of the first evaluation was accepted at 32nd ACM Symposium on Applied Computing - SAC (See Table 8.1, #1).

8 CONCLUSIONS

This chapter presents the concluding remarks. Section 8.1 reviews the contributions of this research. Some limitations of this research are presented in Section 8.2. Section 8.3 presents our future works. Finally, Section 8.4 presents opportunities for new researches.

8.1 Review of the contributions

To support the requirements specification activity in agile software development, this research brings the following contributions:

- **RSD Approach** (described in Chapter 6) advances the knowledge on the field through the design practices that are proposed to support the development team in requirements specification activity in ASD acting on the problems pointed out in the Figure 1.1. Unlike the current existing approaches, RSD is designed for the developer, and not to validate requirements with customers. 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, NFR and technical constraints (acceptance criteria). Although these views are meant to capture requirements, they are very close to the field of implementation helping in the software maintenance and transfer knowledge. The RSD approach is the main contribution of the research. A paper detailing the approach was accepted in an important international conference (see Table 8.1, #1). The RSD approach answer the RQ3 defined in Section 1.2;

- An explanatory **model** (described in Chapter 5) that contributes to the state of the art by providing a deeper understanding about factors which affect the quality of SRS in ASD and how they affect the work of the software engineers. The cross-case synthesis supports the requirements specification activity in ASD revealing findings that have implications for software organizations in their quest to create better work environments and teams. An initial version of the model that emerged from the synthesis of two case studies was published in an international conference (see Table 8.1, #2). The model answer the RQ1 defined in Section 1.2;
- A **systematic mapping** (described in Chapter 3) of literature about how requirements engineering is being conducted in ASD. A qualitative research investigated 24 primary studies collecting and analyzing evidences of 68 companies, involving a total of 270 people in these companies. The SM presented an investigation of the techniques currently used to elicit and specify requirements in agile projects. It contributes to the state of the art by synthesizing 49 challenges related to requirements engineering in ASD and identifying research gaps. The SM was published in an international conference and in a Brazilian journal, as shown in Table 8.1, items #3 and #4, respectively. The systematic mapping answer the RQ1 defined in Section 1.2;
- **Six exploratory industrial case studies** (described in Chapter 4) were conducted to investigate the requirements specification activity in the context of agile projects. The case studies followed a multi-case design with a precisely defined research protocol that has been tested in practice. The study design and procedures were published in an international conference (see Table 8.1, #2). Other researchers in other contexts can develop similar studies based on our protocol. Further integration of results from other case studies would be important to improve our understanding of the quality of SRS in agile projects;
- **Two empirical studies** (described in Chapter 7) evaluated how the RSD approach works in practice. The design and procedures employed to conduct these empirical studies were described in detail and they can be used by other researchers in other contexts and situations in or-

der to develop new studies. A paper detailing partial results of the first evaluation was accepted in an important international conference (see Table 8.1, #1).

Table 8.1 summarizes the publications that have been produced so far. The timeline of publications is shown in Figure 8.1. The complete results of the cross-case analysis and the model emerged from it were detailed in a paper that is under review to be submitted to an international journal.

Table 8.1 – Publications of this research

	Conference/Journal	Type	Content of publication
#1	SAC 2017- 32nd ACM Symposium on Applied Computing. April 03-07, Marrakech, Morocco. doi:http://dx.doi.org/10.1145/3019612.3019753	Conference	RSD Approach and partial results of the first evaluation
#2	QUATIC - 10th International Conference on the Quality of Information and Communications Technology. doi: 10.1109/QUATIC.2016.49 (MEDEIROS et al., 2016b)	Conference	First version of model
#3	ESELAW - 12th Workshop on Experimental Software Engineering. (MEDEIROS et al., 2015)	Conference	Systematic Mapping
#4	PRINCIPIA - ISSN 1517-0306 doi: http://dx.doi.org/10.18265/1517-03062015v1n28p11-24. (MEDEIROS et al., 2015c)	Journal	Systematic Mapping detailed
#5	NAIST - New Advances in Information Systems and Technologies, Springer, ISSN 2194-5357 doi: 10.1007/978-3-319-31232-3_35 (MEDEIROS et al., 2016)	Journal	Outline of RSD approach
#6	CIBSE – 18th Ibero-American Conference on Software Engineering. Doctoral Symposium. (MEDEIROS et al., 2015b)	Conference	Thesis Proposal
#7	CBSOFT – 5th Brazilian Conference on Software: Theory and Practice. Thesis and Dissertation Workshop. (MEDEIROS et al., 2014)	Conference	Thesis Proposal
#8	Thesis Qualification	Partial requirement for the Phd Degree	Thesis Proposal

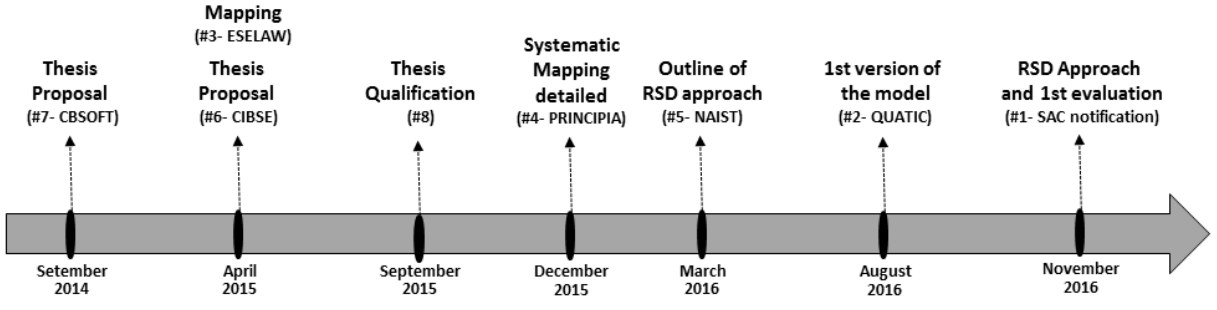

Thesis Proposal (#7- CBSOFT)
Setember 2014
Systematic Mapping (#3- ESELAW)
Thesis Proposal (#6- CIBSE)
April 2015
Thesis Qualification (#8)
September 2015
Systematic Mapping detailed (#4- PRINCIPIA)
December 2015
Outline of RSD approach (#5- NAIST)
March 2016
1st version of the model (#2- QUATIC)
August 2016
RSD Approach and 1st evaluation (#1- SAC notification)
November 2016

Figure 8.1: Timeline of publications

In Section 8.4, we pointed out other crucial questions attached to SRS in ASD, which are worthy of further investigation, serving, thus, as a basis to substantiate and organize new researches in this area.

8.2 Limitations

This work presents a series of limitations, regarding the following aspects:

- **Systematic Mapping**
  - The SM did not consider papers published in 2014 because the research was under way;
  - Approximately 6% of selected papers could not be analyzed because they were not available for download on the network of UFPE and there was no success in attempts to get the items directly from the authors. So, it is possible that some relevant paper has not been included for analysis;
  - It was not made a conceptual analysis to identify possible variations and interposition of the elicitation techniques and specification of the requirements identified by the first two research questions. For example, variations between the JAD and Brainstorm techniques of elicitation (see Section 3.3);
- **Model on quality factors of SRS in ASD**
  - The companies investigated did not provide historical data on quality and productivity (e.g., man-hours, non-conformities, among others). The outcomes described in the model reflect the results of the analysis that considered the opinion of software engineers, artifacts and the comments that were made during the period of observation in the development environment of the projects. We did not collect metrics to assess the extent of effectiveness of these outcomes;
- **RSD Approach**
  - The specification of mockups is one of the three practices used by the approach. The approach may have its benefits reduced when used in software projects that do not have a visual user interface, such as web services, or batch processing. The RSD

  approach has not yet been evaluated in the context of these types of projects;
- The evaluation of the approach aimed at identifying strengths and weaknesses. No quantitative evaluation was done to compare RSD with other approaches, although some questions have been asked about it during the interviews;
- The approach does not require the use of a specialized tool to support the elaboration of acceptance criteria, mockups, and conceptual model. However, reuse and traceability of requirements are compromised when there is no support of an appropriate tool.

8.3 Future Works

Regarding the opportunities to improve the current knowledge about the requirements specification activity in ASD, we expect to continue this research with the following actions:

- Publish the results of the second case study that evaluated the approach, comparing with the results of the first case study;
- Develop a tool to support the adoption of RSD approach in practice, providing the following features:
    - Automatic extraction of widgets from mockups, and the entities and fields from the conceptual model;
    - An editor to describe the AC+, as well as, to control their status (life cycle) illustrated in Figure 6.3;
    - Mechanisms to search the AC+ in the repository, encouraging reuse;
    - Automatic generation of traceability between requirements and AC+;
    - Automatic generation of the skeleton (source-code) of the business classes from the conceptual model.
    - Automatic generation of a roadmap (checklist) from AC+ to support the acceptance testing and the registering of results;

    - Provide a checklist with the quality factors of the *Content* and *Structure* categories so that the team can evaluate the RSD according to these quality factors;
    - Track the history of changes in requirements;
    - Integration with project management tools, such as *Redmine* in order to facilitate the adoption in the industry;
    - Integration with tools that assess the quality of textual requirements, such as the *RQA* tool (Requirements Query Analyzer) (GÉNOVA et al., 2011).
- Conduct further assessments of the RSD approach in other agile contexts (e.g., projects for the development of mobile applications and projects with distributed teams) to identify the points of convergence and divergence regarding the two empirical studies conducted and to improve the RSD approach.

8.4 Opportunities for new researches

The requirements specification activity in ASD still leaves many open questions and opportunities for new researches. So, we recommend that future researches should also evolve in the following directions:

- **Systematic Mapping**
    - Investigate in the industry the challenges related to the *Customer* and *Requirements Management* categories identified in SM. These categories were not investigated in the six industrial case studies carried out in this work;
- **Model on quality factors of SRS in ASD**
    - Based on the model, new studies can be performed to propose other practices to improve the requirements specification in ASD;
    - Conduct experiments to compare customer-oriented SRS versus developer-oriented SRS;
    - Define metrics to evaluate more objectively some of the quality factors defined in the model;

  - Conduct experiments to investigate whether there are statistically significant correlations between the quality factors and the outcomes appointed in the model. For example, to investigate if a complete SRS entails a total development effort (specification and code time) smaller than an incomplete SRS;
  - Conduct confirmatory case studies to test part of the model. For example, to test the factors of the *External* category (see Table 5.4);
  - Conduct other studies, similar to ours, in different contexts and development platforms. This extension to different contexts is likely to yield some common results, increasing our ability to infer that those results are applicable to a broader range of contexts. Similarly, it may also lead to some conflicting results with deep and rich explanations of localized contexts. It will give us a better understanding of reality, albeit a very small part of it;

- **RSD Approach**
  - Conduct a controlled experiment to evaluate the RSD quantitatively in comparison to other approaches regarding the team productivity and quality of SRS;
  - Investigate how the approach behaves in projects that require the requirements validation with the customer through the SRS.

REFERENCES

ABDULLAH, B.N.N., HONIDEN, S., SHARP, H., NUSEIBEH, B., NOTKIN, D. **Communication patterns of agile requirements engineering**. In Proceedings of the 1st Workshop on Agile Requirements Engineering (AREW). ACM, New York, USA, 2011.

AGILE ALLIANCE. **Subway Map to Agile Practices**. Available at: https://www.agilealliance.org/agile101/subway-map-to-agile-practices/. Accessed: 18/12/2016.

AGILE MANIFESTO. **Manifesto for Agile Software Development**. Agile Aliance, Available at: http://www.agilemanifesto.org/. Accessed: 18/12/2016. 2001.

ALVES, D.C.P. **Engenharia de Requisitos em Projetos Ágeis: Um mapeamento sistemático baseado em evidências da indústria**. Masters Dissertation, Universidade Federal de Pernambuco, Centro de Informática. July, 2015.

AMBLER, S. W. **Agile Requirements Best Practices**. Available at: http://agilemodeling.com/essays/agileRequirementsBestPractices.htm. Accessed: 18/12/2016.

BECK, K. **Extreme Programming Explained: Embrace Change**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA,1999.

BECK, K. **Test Driven Development: By Example**. Ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

BATOOL, A., MOTLA, Y.H., HAMID, B., ASGHAR, S., RIAZ, M., MUKHTAR, M., AHMED, M. **Comparative study of traditional requirement engineering and Agile requirement engineering**. Advanced Communication Technology (ICACT), 15th International Conference, pp.1006-1014, 2013.

BJARNASON, E., WNUK, K., REGNELL, B. **Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering**. Inf. Softw. Technol. 54, 10, 1107-1124. DOI=10.1016/j.infsof.2012.04.006, 2012.

BJARNASON, E., WNUK, K., REGNELL, B. **A case study on benefits and side-effects of agile practices in large-scale requirements engineering**. In Proceedings of the 1st Workshop on Agile Requirements Engineering (AREW). ACM, New York, NY, USA, Article 3, 2011.

BROWN, B.B. **Assurance of Software Quality**. SEI-CM-7, 1987.

CMMI. **CMMI for Development.** Version 1.3 (CMU/SEI-2010-TR-033). Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661, 2010.

COHN, M. **User stories applied for Agile Software Development**. Addison-Wesley, 2004.

CRUZES, D.S., DYBA, T. **Recommended Steps for Thematic Synthesis in Software Engineering. International Symposium on Empirical Software Engineering and Measurement**. Banff, AB, 2011, pp. 275-284. doi: 10.1109/ESEM.2011.36, 2011.

DANEVA, M., VAN DER VEEN, E., AMRIT, C., GHAISAS, S., SIKKEL, K., KUMAR, R., AJMERI, N., WIERINGA, R. **Agile requirements prioritization in large-scale outsourced system projects: An empirical study**. J. Syst. Soft. 86, 5, 1333-1353, 2013.

DYBA, T., DINGSOYR, T. **Empirical studies of agile software development: A systematic review**. Information and Software Technology, doi:10.1016/j.infsof.2008.01.006, USA, 2008.

EASTERBROOK, S., SINGER, J., STOREY, M-A., DAMIAN, D. **Selecting Empirical Methods for Software Engineering Research**, In: Shull F et al (ed) Guide to advanced empirical software engineering, Chapter 11. Springer-Verlag, London, 2008.

EISENHARDT, K. M. **Building theories from case study research**. Academy of Management Review, 14: 532–550, 1989.

FERREIRA, J., NOBLE, J., BIDDLE, R. **Agile Development Iterations and UI Design**. Agile Conference (AGILE), Washington, DC, pp. 50-58. doi: 10.1109/AGILE.2007.8, 2007.

FLYVBJERG, B. **Five Misunderstandings About Case-Study Research**. Qualitative Inquiry, v. 12, n. 2, p. 219-245, 2006.

FRANÇA, A. C. C., SILVA, F. Q. B., FELIX, A.L. C., CARNEIRO, D. E. S. **Motivation in software engineering industrial practice: A cross-case analysis of two software organisations**. Inf. Softw. Technol. 56, 1, 79-101. DOI=http://dx.doi.org/10.1016/j.infsof.2013.06.006, 2014.

FRIGG, R., HARTMANN, S. **Models in Science**. The Stanford Encyclopedia of Philosophy, Edward N. Zalta (ed.), http://plato.stanford.edu/archives/fall2012/entries/models-science/, 2012.

GEBHART, M., GIESSLER, P., BURKHARDT, P., ABECK, S. **Quality-Oriented Requirements Engineering for Agile Development of RESTful Participation Service**. ICSEA: The Ninth International Conference on Software Engineering Advances, 2014.

GÉNOVA, G., FUENTES, J., LLORENS, J., HURTADO, O., MORENO, V. **A framework to measure and improve the quality of textual requirements**. Requirements Engineering, vol. 18, no. 1, pp. 25–41, 2013.

GONZALEZ-PEREZ, C., HENDERSON-SELLERS, B. **Metamodelling for Software Engineering**. John Wiley & Sons, Ltd, ISBN 978-0-470-03036-3, 2008.

HAUGSET, B., STALHANE, T. **Automated Acceptance Testing as an Agile Requirements Engineering Practice**. System Science (HICSS), 45th Hawaii International Conference on, Maui, HI, pp. 5289-5298, doi: 10.1109/HICSS.2012.127, 2012.

HECK, P., ZAIDMAN, A. **A Quality Framework for Agile Requirements: A Practitioner's Perspective**. arXiv preprint arXiv:1406.4692, 2014.

HECK, P., ZAIDMAN, A. **A systematic literature review on quality criteria for agile requirements specifications**. Software Qual J. Springer Science+Business Media New York. DOI 10.1007/s11219-016-9336-4, 2016.

HEIKKILÄ, V. T., DAMIAN, D., LASSENIUS, C., PAASIVAARA, M. **A Mapping Study on Requirements Engineering in Agile Software Development**. 41st Euromicro Conference on Software Engineering and Advanced Applications, Funchal, pp. 199-207. doi: 10.1109/SEAA.2015.70, 2015

HIRSCHHEIM, R., KLEIN, K., LYYTINEN, K. **Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations**. Cambridge University Press, New York, NY, USA, 1995.

INAYAT, I., SALIM, S.S., MARCZAK, S., DANEVA, M., SHAMSHIRBAND, S. **A systematic literature review on agile requirements engineering practices and challenges**. Comput. Hum. Behav. 51, PB, October, 915-929. DOI: http://dx.doi.org/10.1016/j.chb.2014.10.046, 2015.

ISO-IEEE 830-1998 - **IEEE Recommended Practice for Software Requirements Specifications**, IEEE, 1998.

ISO-IEEE 29148:2011 - **Systems and software engineering - Life cycle processes - Requirements engineering**, IEEE, 2011.

JACOBSON. I., CHRISTERSON, M., JONSSON , P., OEVERGAARD, G. **Object Oriented Software Engineering: a Use Case Driven Approach**. Addison-Wesley, 1992.

KASSAB, M. **The changing landscape of requirements engineering practices over the past decade**. In: Proceedings of the IEEE international workshop on empirical requirements engineering (EmpiRE). IEEE, pp 1–8, 2015.

KITCHENHAM, B.A., PICKARD, L.M. and PFLEEGER, S.L. **Case Studies for Method and Tool Evaluation**. IEEE Software, 12(4):52–62, 1995.

KITCHENHAM, B., BUDGEN, D., BRERETON, P., LINKMAN, S. H. **Realising evidence-based software engineering**. In Proceedings of the 2005 workshop on Realising evidence-based software engineering (REBSE '05). ACM, New York, NY, USA, 1-3. DOI=http://dx.doi.org/10.1145/1082983.1083175. 2005.

KITCHENHAM, B., CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. Vol 2.3, EBSE-2007-01, Keele, UK, 2007.

LOSADA, B., URRETAVIZCAYA, M., FERNÁNDEZ-CASTRO, I. **A guide to agile development of interactive software with a User Objectives-driven methodology**. Science of Computer Programming, Volume 78, Issue 11, 1. Pages 2268-2281, 2013.

LOUCOPOULOS, P., ZICARI, R. **Conceptual Modeling, Databases and CASE: An Integrated View of Information System Development**. John Wiley & Sons, New York, USA, 1992.

LUCASSEN, G., DALPIAZ, F., WERF, J.M.E.M., BRINKKEMPER, S. **Improving agile requirements: the Quality User Story framework and tool**. DOI 10.1007/s00766-016-0250-x. pp 1-21, 2016.

LUCIA, A. D., ABDALLAH, O. **Requirements engineering in agile software development**. Journal of Emerging Technologies in Web Intelligence, vol. 3, pp.212-220, 2010.

MARCONI, M., LAKATOS, E. M. **Fundamentos de metodologia científica**. 5. ed. - São Paulo: Atlas, 2003.

MARCONI, M., LAKATOS, E. M. **Metodologia Científica**. 6. ed. São Paulo: Atlas, 2008.

MAMOLI, S. **On Acceptance Criteria for User Stories**. Available: http://nomad8.com/acceptance_criteria/. Acessed: 18/12/2016.

MEDEIROS, F., MEDEIROS, J.D.R.V, AYRES, F., VIANA, C., ROCHA, J., VIEGAS, V., MENDES, E., SANTOS, A.. **An Agile Approach to Developing an Information System for Anti-doping Control in Brazil**. NAIST - New Advances in Information Systems and Technologies. 1ed.: Springer International Publishing, ISSN 2194-5357, doi: 10.1007/978-3-319-31232-3_35, p. 369-378, 2016.

MEDEIROS, J.D.R.V, GOULÃO, M., VASCONCELOS, A., SILVA, C. 2016. **Towards a model about quality of software requirements specification in agile projects**. 10th International Conference on the Quality of Information and Communications Technology (QUATIC). doi: 10.1109/QUATIC.2016.49, Lisbon, Portugal, 2016b.

MEDEIROS, J.D.R.V., ALVES, D.C., WANDERLY, E., VASCONCELOS, A.M.L. and SILVA, C. **Requirements Engineering in Agile Projects: A Systematic Mapping based in Evidences of Industry**. 12th Workshop on Experimental Software Engineering (ESELAW) in XVIII CIBSE Ibero-American Conference on Software Engineering. 2015.

MEDEIROS, J.D.R.V., VASCONCELOS, A.M.L., SILVA, C. **Integration of Agile Practices: An approach to improve the quality of software specifications**. XVIII CIBSE Ibero-American Conference on Software Engineering– Doctoral Symposium. 2015b.

MEDEIROS, J.D.R.V., ALVES, D.C., WANDERLY, E., VASCONCELOS, A.M.L., SILVA, C. **Engenharia de requisitos em projetos ágeis: uma revisão sistemática da literatura**. ISSN 1517-0306, Principia Journal, http://dx.doi.org/10.18265/1517-03062015v1n28p11-24. 2015c.

MEDEIROS, J.D.R.V., VASCONCELOS, A.M.L., SILVA, C. **Integração de Práticas Ágeis: Uma abordagem para melhorar a qualidade de especificações de**

software em projetos mobile. IV Thesis and Dissertation Workshop (WTDSoft) of CBSoft (Brazilian Conference on Software), 2014.

MERRIAM, S.B. **Qualitative Research: A Guide to Design and Implementation**. ISBN: 978-1-118-94701-2. Jossey-Bass, 2009.

MPS.BR. **Melhoria de Processo de Software Brasileiro**. SOFTEX. Available at: http://www.softex.br/mpsbr. Accessed: 18/12/2016, 2012.

NAWROCKI, J., JASINSKI, M., WALTER, B., WOJCIECHOWSKI, A. **Extreme programming modified: embrace requirements engineering practices**. IEEE Joint International Conference, pp. 303-310, 2002.

NEBE, K., SNIGDHA, B. **Agile Human-Centred Design: A Conformance Check-list**. 18th International Conference, HCI International, Toronto, Canada, July 17-22, Part I, pp 442-453. doi=10.1007/978-3-319-40349-6_42, 2016.

NOBLIT, G.W., HARE, R. D. **Meta-Ethnography: Synthesizing Qualitative Studies (Qualitative Research Methods)**, Sage Publications Inc., 1988.

NORTHCUTT, N., and MCCOY, D. **Interactive qualitative analysis: a systems method for qualitative research**. Thousand Oaks, CA: SAGE Publications, 2004.

OLIVÉ, A. **Conceptual Modeling of Information Systems**. Springer Verlag, ISBN 978-3-540-39389-7. doi 10.1007/978-3-540-39390-0, 2007.

PAETSH, F., EBERLEIN, A., MAURER, F. **Requirements Engineering and Agile Software Development**. In: 12th IEEE International WETICE 03, IEEE CS Press, 2003.

PANDIT, N.R. **The Creation of Theory: A Recent Application of the Grounded Theory Method**. The Qualitative Report, Volume 2, Number 4, December, 1996.

PERRY, D. E.; SIM, S. E.; EASTERBROOK, S. M. **Case Studies for Software Engineers**. Proceedings of the 26th International Conference on Software Engineering. [S.l.]: [s.n.], 2004.

PETRONIO, F., MEDEIROS, J.D.R.V, AYRES, F., VIANA, C., ROCHA, J., VIEGAS, V., MENDES, E., SANTOS, A. **An Information System to Support the Anti-doping Process**. Lecture Notes in Electrical Engineering. 1ed.: Springer Singapore, v. 1, p. 97-107. DOI = 10.1007/978-981-10-0557-2_10, 2016.

POVILAITIS, S. **Acceptance Criteria**. Available at: http://www.leadingagile.com/2014/09/acceptance-criteria/. Accessed: 18/12/2016.

READ, A., BRIGGS, R.O. **The Many Lives of an Agile Story: Design Processes, Design Products, and Understandings in a Large-Scale Agile Development Project**. System Science (HICSS), 45th Hawaii International Conference on, Maui, HI, pp. 5319-5328. doi: 10.1109/HICSS.2012.684, 2012.

RICCA, F., SCANNIELLO, G., TORCHIANO, M., REGGIO, G., ASTESIANO, E. **Assessing the Effect of Screen Mockups on the Comprehension of Functional Requirements**. ACM Trans. Softw. Eng. Methodol. 24, 1, Article 1, October, 38 pages. DOI: http://dx.doi.org/10.1145/2629457, 2014.

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**. Information and Software Technology, Volume 56, Issue 6, June Pages 670-687, ISSN 0950-5849, http://dx.doi.org/10.1016/j.infsof.2014.01.011, 2014.

RUDORFER, A., STENZEL, T., HEROLD, G. **A Business Case for Feature-Oriented Requirements Engineering**. IEEE Software, vol. 29, no. 5, pp. 54-59, Sept.-Oct. doi: 10.1109/MS.2012.106, 2012.

RUNESON, P., MARTIN, H. **Guidelines for conducting and reporting case study research in software engineering**. Empirical Software. 14, 2, 131-164. http://dx.doi.org/10.1007/s10664-008-9102-8, 2009.

SAAVEDRA, R., BALLEJOS, L., ALE, M. **Software Requirements Quality Evaluation: State of the art and research challenges**. 14th Argentine Symposium on Software Engineering, ASSE, 2013.

SAITO, S., TAKEUCHI, M., HIRAOKA, M., KITANI, T., AOYAMA, M. **Requirements clinic: Third party inspection methodology and practice for improving the quality of software requirements specifications**. Requirements Engineering Conference (RE), 21st IEEE International, pp.290-295. doi: 10.1109/RE.2013.6636732, 2013.

SCHWABER, K., BEEDLE, M. **Agile Software Development with Scrum**. Prentice Hall PTR, NJ, USA, 2001.

SEAMAN, C.B., Qualitative Methods, in: F. Shull, J. Singer, D.I.K. Sjøberg (Eds.), **Guide to Advanced Empirical Software Engineering**, Springer, pp. 35–62. Chapter 2, 2008.

SHARP, H., ROBINSON, H., PETRE, M. **The role of physical artefacts in agile software development: Two complementary perspectives**. Interact. Comput. 21, 1-2, 108-116. doi=http://dx.doi.org/10.1016/j.intcom.2008.10.006, 2009.

SCHÖN, E-M, THOMASCHEWSKI, J, ESCALONA, M. J. **Agile Requirements Engineering: A systematic literature review**, Computer Standards & Interfaces, Volume 49, January 2017, Pages 79-91, ISSN 0920-5489, http://dx.doi.org/10.1016/j.csi.2016.08.011.

SJØBERG, D.I.K., DYBÅ, T., JORGENSEN, M. **The Future of Empirical Methods in Software Engineering Research**. Proceedings of the Future of Software Engineering (FOSE '07). [S.I.]: IEEE, 2007.

SJØBERG, D.I.K., DYBÅ, T., ANDA, B.C.D., HANNAY, J.E. **Building theories in software engineering**, in: F. Shull, J. Singer, D.I.K. Sjøberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer, pp. 312–336. Chapter 12, 2008.

TECHREPUBLIC. **Tech companies have highest turnover rate**. Available: http://www.techrepublic.com/blog/career-management/tech-companies-have-highest-turnover-rate/. Accessed in: 19/02/2015.

TERZAKIS, J. **Tutorial writing higher quality software requirements**. ICCGI, http://www.iaria.org/conferences2010/filesICCGI10/ICCGI_Software_Requirements_Tutorial.pdf. pp 37-38. Accessed in: 18/12/2016.

THAYER, R.H., DORFMAN, M. **Software Requirements Engineering**. IEEE Computer Society Press, 2d ed., 1997.

THOMSON, C., HOLCOME, M., COWLING, T., SIMONS, T., MICHAELIDES, G. **A pilot study of comparative customer comprehension between extreme x-machine and uml models**. Empirical software engineering and measurement. ESEM'08, ACM, New York, pp. 270–272 http://dx.doi.org/10.1145/1414004.1414048, 2008.

TRAVASSOS, G., BIOLCHINI J. **Revisões Sistemáticas Aplicadas a Engenharia de Software**. In: XXI SBES - Brazilian Symposium on Software Engineering, João Pessoa, PB, Brazil, 2007.

VERSIONONE, **10th Annual State of Agile Survey**. Available at: https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf. Accessed in: 18/12/2016.

WANDERLEY, F., SILVA, A., ARAUJO, J., SILVEIRA, D. S. **SnapMind: A framework to support consistency and validation of model-based requirements in agile development**. IEEE 4th MoDRE, Karlskrona, Sweden. 2014.

WHICHARD, G. **Definition of Done vs. Acceptance Criteria**. Available in http://www.governmentciomagazine.com/2014/08/definition-done-vs-acceptance-criteria. Accessed in: 18/12/2016.

WOHLIN, C., HÖST, M., HENNINGSSON, K. **Empirical Research Methods in Software Engineering**. Lecture notes in Computer Science, 2765 7-23. 2003.

INAYAT, I., SALIM, S. S., Marczak, S., Daneva, M., Shamshirband, S., **A systematic literature review on agile requirements engineering practices and challenges**, Computers in Human Behavior, Volume 51, Part B, October, Pages 915-929, ISSN 0747-5632, http://dx.doi.org/10.1016/j.chb.2014.10.046, 2015.

YIN, R. K. **Case Study Research: Design and Methods**. 4th. ed. Thousand Oaks, California: SAGE Publications, v. Applied Social Research Methods Series, Volume 5, ISBN 978-1-4129-6099-1, 2009.

APPENDICES

Appendix A - Systematic Mapping Protocol

This appendix contains the protocol of the Systematic Mapping performed in this research, as described in Chapter 3.

1. Objectives and Research Questions

The general goal of this study is to investigate how the requirements engineering has been conducted in projects that adopt agile methods. To achieve the objectives, the following Principal Research Question (PRQ) was defined:

- How is the requirements engineering has been conducted in projects that adopt agile methods?

The following Specific Research Questions (SRQ) were defined to guide the extraction, analysis and synthesis of results:

- SRQ1: What approaches are being used to elicit requirements in projects that adopt agile methods?

- SRQ2: What approaches are being used to specify requirements in projects that adopt agile methods?

- SRQ3: What are the implications for the software industry and academia, reported in the current studies involving the requirements engineering in agile projects?

- SRQ4: What are the implications for the software industry and academia, reported in the current studies involving the Requirements Engineering in Agile projects?

2. Search Strategy

According to Kitchenham (2007), a strategy must be used for the selection of primary studies from the definition of keywords, digital libraries, journals, and conferences. The search terms have been identified from the structure of research questions. As recommended by Dyba and Dingsoyr (2008), the terms used in the construction of the string was as inclusive as possible in order to return a greater number of papers and to avoid the loss of relevant studies, for this reason, the PICO method is not used. The wide-ranging search aims to eliminate the bias in the selection of the primary studies for this research. The search terms, synonyms or related words are presented in Table A.1.

Table A.1 – SM Protocol - Search terms, synonyms or related words

Search terms	Synonyms or related words
Software	Software, information system development, information system engineering.
Agile methods	Agile, agility, scrum, extreme programming, xp, dynamic system development, dsdm, crystal methodologies, crystal clear, crystal orange, crystal red, crystal blue, feature driven development, fdd, lean software development, adaptive software development, test driven development.
Requirements	Requirements, use case, user stories.

According to Kitchenham (2007), the search strings must be derived from the research questions and search terms. Thus, the search string was generated from the combination of search terms, their synonyms or related words, concatenating them through the Boolean operators "OR" and "AND". The following String Search was defined:

(("requirements" OR "use case" OR "use cases" OR "user stories") AND ("agile" OR "agility") AND ("scrum" OR "extreme programming" OR "xp" OR "dynamic system development" OR "dsdm" OR "crystal methodologies" OR "crystal clear" OR "crystal orange" OR "crystal red" OR "crystal blue" OR "feature driven development" OR "fdd" OR "lean software development" OR "adaptive software development" OR "test driven development" OR "tdd") AND ("software" OR "information system development" OR "information system engineering"))

3. Data Sources

The search will be performed using both automatic and manual approach. The following search engines will be used: IEEExplore Library, ACM Library, ScienceDirect, SpringerLink, and Scopus. The manual search will be done in the Proceedings of International Requirements Engineering Conference and Agile Development Conference covering the period from 2009 to 2013.

4. Selection Criteria

The studies should be selected according to the inclusion and exclusion criteria, as described in Table A.2. A study should be included if it fulfills all the inclusion criteria. A paper must be excluded if at least one of the exclusion criteria is met.

Table A.2 – SM Protocol - Inclusion and Exclusion Criteria

Inclusion Criteria	IC1. Studies addressing requirements on software projects using agile methodologies IC2. Studies validated in the industry IC3. Qualitative or quantitative research IC4. Primary or secondary studies
Exclusion Criteria	EC1. Studies written in a language other than English EC2. Duplicated study report, with no extra information EC3. Studies that do not address on elicitation, specification or modeling software requirements EC4. Incomplete studies, prefaces, slides or summaries EC5. Tertiary studies EC6. Studies that address only the teaching of agile or requirements EC7. Studies that do not address at least an agile methodology EC8. Papers not available for download in institutional environments UFPE or IFPB. EC9. Studies that no present empirical data

5. Study Selection Procedures

This research will be conducted by at least two researchers. The selection process should be performed following the six steps below.

5.1 Automated and manual search

A researcher shall perform the search using the Reviewer[21] tool to execute the string simultaneously in all search engines. The result will be exported to an Excel spreadsheet which should be used in the next steps. Then the manual search will be conducted in order to add studies to the list obtained from the automated search.

5.2 Selection by Title and Abstract

Each study selected in the previous step should be examined by a pair of researchers. The titles and abstracts should be read, and the inclusion and exclusion criteria should be applied. If, after examination of the researchers, doubts exist about the relevance of the study, it must be included. The papers selected in this stage should be recorded in a list of potentially relevant articles in order to be analyzed in the following steps.

5.3 Selection by Introduction and Conclusion:

Each study selected in the previous step should be analyzed by a pair of researchers, preferably different from the previous step. The criteria should be applied

[21] https://github.com/bfsc/reviewer

based on the reading of the introduction and conclusion of the studies resulting from the previous phase. If necessary, the remaining sections of the papers can be read to help in judging whether the criteria should be applied. In the case of disagreement among researchers about the inclusion or exclusion of a study, a consensus meeting should be made, and if the conflict persists should consider the opinion of a third investigator.

5.4 Quality assessment

The assessment of the quality of primary studies should be performed after the application of the criteria (inclusion and exclusion), and two researchers should read all sections of the papers. The evaluation should be done using a questionnaire adapted from Dyba (2008). The applied questions are presented in Table A.2. A three-point scale of Likert should be used to evaluate the papers: 0 (Nothing in the paper that meets the criteria evaluated); 0.5 (The paper does not make clear whether or not meet the criteria) and 1 (Paper meets the criteria evaluated).

Table A.3 – SM Protocol - Questions for Quality Assessment

1. It is a research paper?
2. Is there a clear statement of the aims of the research?
3. Is there a description of the context in which the research was carried out?
4. Was the research design appropriate to address the aims of the research?
5. Was the recruitment strategy appropriate to the aims of the research?
6. Was the data collected in a way that addressed the research questions?
7. Was the data analysis sufficiently rigorous?
8. Has the relationship between researcher and participants been considered?
9. Is there a clear statement of findings?
10. Is the study of value for research of practice?

The answers to the questionnaires should be tabulated so that the researchers can compare, discuss and find a consensus. The scores of all the questions should be summarized and the papers should be classified into four quality groups: low (score < = 2), medium (score> = 8.5), high (8 <= score> = 6) or very high (score> = 8.5). The papers with a low quality (score < = 2) should be discarded.

5.5 Extraction

The papers resulting from the previous stage must be divided among the researchers to perform the extraction. Each researcher must analyze the data extracted by the other. This review is necessary to enhance the quality of the data collected. The extraction should follow the guidelines of Cruzes (2011) who argue that the extraction of data is a key part in systematic reviews. To extract the data from the relevant primary studies in a structured way should be used in the form below:

Table A.4 – SM Protocol - Form for Data Collection

Form for Data Collection								
ID:		Title:					Field:	
General Overview:								
Collaboration with Customer:								
Characteristics of the Development Team:								
Evidence related to the Research Questions								
SRQ1 - What approaches are being used to elicit requirements in projects that adopt agile methods?								
SRQ2 - What approaches are being used to specify requirements in projects that adopt agile methods?								
SRQ3 - What are the implications for the software industry and academia, reported in the current studies involving the requirements engineering in agile projects?								
Evidence related to the Context								
Publication	**Context Data**							
Year and source	**Study Type**	**Research Method**	**Data Collect**	**Data Analysis**	**Sample**	**Agile Methods**	**Country**	

5.6 Synthesis

The analysis and synthesis of the data will be conducted following a qualitative approach to answering the research questions about the requirements engineering in agile software development. A thematic synthesis and data analysis will be carried out according to the process recommended by Cruzes and Dyba (2011), which can be seen in Figure A.

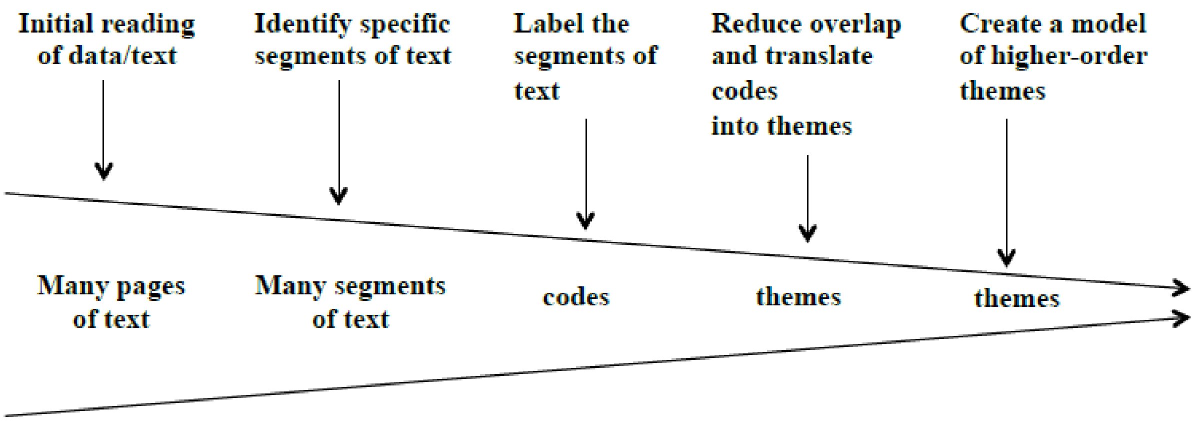

Initial reading of data/text
Identify specific segments of text
Label the segments of text
Reduce overlap and translate codes into themes
Create a model of higher-order themes
Many pages of text
Many segments of text
codes
themes
themes

Figure A.1 Thematic synthesis process, adapted from Cruzes and Dyba (2011)

Briefly, the process starts with an initial reading of the texts, then extract specific segments of text which are labeled with codes. These codes are translated into themes, and then identified the themes of higher order (categories) to create the model to explain the phenomenon or research questions. This process will conduct by a researcher and then reviewed by another researcher.

Appendix B - Selected Primary Studies

This appendix contains the Primary Studies (PS) that it was selected and analyzed in Systematic Mapping.

ID	YEAR	SOURCE	REFERENCE
PS01	2012	IEEE	Rudorfer, A., Stenzel, T., Herold, G. A Business Case for Feature-Oriented Requirements Engineering. Software, vol.29, no.5, pp.54,59, Sept.-Oct. doi: 10.1109/MS.2012.106.
PS02	2011	ACM	Bjarnason, E., Wnuk, K., Regnell, B. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. 1st Workshop on Agile Requirements Engineering (AREW '11). New York, USA, Article 3, 5 pages. doi=10.1145/2068783.2068786.
PS03	2008	ACM	Thomson, C., Holcome, M., Cowling, T., Simons, T., Michaelides, G. A pilot study of comparative customer comprehension between extreme x-machine and uml models. Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '08). New York, USA, 270-272. doi=10.1145/1414004.1414048.
PS04	2008	SCOPUS	Cao, L.A., Ramesh, B. B. Agile requirements engineering practices: An empirical study. IEEE, Vol.25, no.1, pp.60,67. doi: 10.1109/MS.2008.1.
PS05	2007	ACM	Bang, T.J. An agile approach to requirement specification. 8th international conference on Agile processes in software engineering and extreme programming (XP'07. Springer-Verlag, Berlin, Heidelberg, 193-197.
PS06	2012	SCIENCE DIRECT	Bjarnason, E., Wnuk, K., Regnell, B. Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering. Inf. Software Technol. 54, 10, 1107-1124. doi=10.1016/j.infsof.2012.04.006.
PS07	2012	IEEE	Haugset, B., Stalhane, T. Automated Acceptance Testing as an Agile Requirements Engineering Practice. System Science (HICSS), 2012 45th Hawaii International Conference on pp.5289,5298. doi: 10.1109/HICSS.2012.127.
PS08	2012	ACM	Abdullah, B.N.N., Honiden, S., Sharp, H., Nuseibeh, B., Notkin, D. Communication patterns of agile requirements engineering. In Proceedings of the 1st Workshop on Agile Requirements Engineering (AREW '11). New York, USA, Article 1, 4 pages. doi=10.1145/2068783.2068784.
PS09	2013	SCOPUS	Batool, A., Motla, Y.H., Hamid, B., Asghar, S., Riaz, M., Mukhtar, M., Ahmed, M. Comparative study of traditional requirement engineering and Agile requirement engineering. Advanced Communication Technology (ICACT), 15th International Conference on pp.1006,1014, 27-30.
PS10	2011	ACM	Lee,J.C, Judge,T.K, McCrickard, D.S. Evaluating eXtreme scenario-based design in a distributed agile team. In CHI '11 Extended Abstracts on Human Factors in Computing Systems (CHI EA '11). New York, USA, 863-877. doi=10.1145/1979742.1979681.
PS11	2006	IEEE	Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J. High-Speed Software Development Practices: What Works, What Doesn't. IT Professional , vol.8, no.4, pp.29,36, doi: 10.1109/MITP.2006.86.
PS12	2012	IEEE	Gregorio, D.D. How the Business Analyst supports and encourages collaboration on agile projects. Systems Conference (SysCon), pp.1,4. doi: 10.1109/SysCon.2012.6189437.
PS13	2013	IEEE	Lorber, A.A., Mish, K.D. How We Successfully Adapted Agile for a Research-Heavy Engineering Software Team. Systems Conference, pp.1,4. doi: 10.1109/SysCon.2012.6189437.
PS14	2011	IEEE	Mahmud, I., Veneziano, V.: Mind-mapping: An effective technique to facilitate requirements engineering in agile software development
PS15	2012	SCOPUS	Farid, W.M., Mitropoulos, F.J.: Novel lightweight engineering artifacts for modeling non-functional requirements in agile processes

PS16	2013	ACM	Abdallah, A., Hassan, R., Azim, M.A.: Quantified extreme scenario based design approach
PS17	2008	ACM	Obendorf,H., Finck, M.: Scenario-based usability engineering techniques in agile development processes
PS18	2012	IEEE	Read, A., Briggs, R.O.: The Many Lives of an Agile Story: Design Processes, Design Products, and Understandings in a Large-Scale Agile Development Project
PS19	2009	SCIENCE DIRECT	Sharp,H., Robinson,H., Petre, M.: The role of physical artefacts in agile software development: Two complementary perspectives
PS20	2004	IEEE	Martin, A., Biddle, R., Noble, J.: The XP customer role in practice: three studies
PS21	2010	REC	Savolainen,J., Kuusela,J., Vilavaara, A.: Transition to Agile Development - Rediscovery of Important Requirements Engineering Practices
PS22	2013	Snowball-ing	Batool, A., Hafeez, Y., Asghar, S., Abbas, M.A., Hassan, M.S.: A Scrum Framework for Requirement Engineering Practices
PS23	2010	Snowball-ing	Sem A.M., Hemachandran, K.: Elicitation of Goals in Requirements Engineering using Agile Methods
PS24	2010	Snowball-ing	Hoda, R., Noble,J., Marshall, S.: Agile Undercover: When Customers Don't Collaborate

Appendix C - Interview Guide to investigate the phenomenon in practice

This appendix contains guides to interviewing the software engineers as data collection instrument for the six case studies described in Chapter 4.

PRESENTATION

- Greetings and introduction.
- Thank the participant.

INTRODUCTION

The purpose of our research is to investigate the phenomenon of the quality of SRS in different agile contexts and build an explanatory model providing a deeper description in order to point out the factors that should be considered to write more useful SRS in agile projects.

We would be very grateful for your input to this research. This would include participating in this interview. As a retribution for your participation, we will provide your company with an analysis of strengths and potential fragilities of the practices related to specifying of requirements in agile projects. This information may be useful for you and your team as you looks for opportunities to improve.

All your responses will be kept strictly confidential. Your participation is voluntary. You will not, in any way, be penalized if you choose not to participate in the study. Do you have any questions regarding this interview?

ABOUT THE ANSWERS

There are no rights or wrongs answers for most of the questions in this interview. My goal is to collect your impressions, opinions, and feelings about the various subjects addressed. So, please answer the questions as spontaneously and honestly as possible, knowing that your answers will not be, in any way, disclosed to other individuals inside or outside your company.

May we start?

Questions

S - Questions applied to software engineers

Team Member Background - TMB		
ID	**Questions**	**Type**
S1	Gender: **O** Male **O** Female	Profile
S2	What is your current role? **O** System Analyst **O** Tester **O** Developer **O** Project Manager **O** Other ___________	Profile
S3	What is your academic background? **O** University Graduate **O** Master **O** Ph.D. **O** Others	Background
S4	How many years of experience do you have in this role?	Experience

Now we will talk about the concept of quality of SRS in agile projects.

Concept of quality of SRS - CQUA		
ID	**Questions**	**Type**
S5	What do you consider a Software Requirements Specification (SRS) with quality? In other words, in order to code a feature, which attributes (characteristics) you would like to find in the SRS? *Probe: Why?*	Opinion
S6	What factors affect the quality of the SRS? *Probe: Why?*	Opinion
S7	In the current project, how do you assess the impact of the following factors in the quality of SRS and your performance? Compromise (impairs) or Not Affect? • Little team experience in requirements specification • Little team experience with agile development • Documentation structure used by the company • The lack of support tool to automate the activities of requirements • A more focused SRS for the client than to the development team • Changes in the data model *Probe: Why?*	Opinion
S8	Do you think an SRS divided in various artifacts facilitates the understanding or is it more productive for the programmer to obtain information in a consolidated manner from a few artifacts? *Probe: Why?*	Opinion

Now we will talk about the content used to specify requirements in your current project.

Content of SRS - CON		
ID	**Questions**	**Type**
S9	In the current project, which artifacts are used by a programmer to code a feature? *Probe: Use Cases \ Scenarios, Features, User Stories, Goals, Interface Prototype, Diagram of Classes, Diagram of Activities, Data Model, Rules, Without SRS, …?*	Context
S10	What other information is not present in the SRS currently used, but are necessary to code a requirement? *Probe: Why?*	Context
S11	How these artifacts help the programmer to code a requirement? *Probe: Why?*	Opinion
S12	Is the SRS sufficient for the developer to produce code, or does the developer frequently need to consult other sources of information (e.g. people, documents, etc.) to answer questions? *Probe: If the SRS is not sufficient, what you think the developer is normally missing?*	Opinion
S13	Do you consider that the SRS has information unnecessary that is not useful	Opinion

	for the execution of your activities? *Probe: If you do, what kind of information you consider unnecessary?*	
S14	Do you consider that the SRS has redundant or duplicate information? *Probe: If you do, why do you think that the SRS is written in this way?*	Opinion
S15	What do you consider that contributes to making the SRS outdated? *Probe: Why?*	Opinion

The next questions are about how the SRS is described in your current project.

Quality of Specification - QUA		
ID	**Questions**	**Type**
S16	In the current project, is the SRS hard to read and understand? *Probe: If YES, why do you think that the SRS is written in this way?*	Opinion
S17	In the current project, do you consider that the SRS is ambiguous, that is, unclear and subject to more than one interpretation? *Probe: If YES, what do you think contributes to the SRS be written in this way?*	Opinion
S18	In the current project, do you consider that the SRS presents problems of consistency between the requirements or between artifacts? *Probe: If YES, why do you think this happens?*	Opinion

Now we will talk about the software process used in your current project.

Context of Development Process- CTX		
ID	**Questions**	**Type**
S19	What is the frequency of communication with the customer?	Context
S20	Do you consider that the communication with the customer is suitable or compromises the execution of your activities? *Probe: Why?*	Opinion
S21	What are the tools used to support the requirements activities? What are the benefits they provide?	Context and opinion
S22	Are the tests made directly from the SRS, or is it necessary to elaborate a specific document only for the tests? Do you consider that the tests are being done properly?	Context and opinion
S23	Does the project hold information about the dependency relationships between requirements? *Probe: If yes, how is this information updated throughout the development?*	Context

Now we will talk about your suggestions to improve the SRS.

Suggestions for Improvements - SUG		
ID	**Questions**	**Type**
S24	What good practices could be used in other projects because they improve the performance of your activities? *Probe: Why? How?*	Opinion
S25	What changes in the SRS and / or the requirements activities do you suggest in order to improve the development of future projects? *Probe: Why? How?*	Opinion

M - Questions applied to project managers

Profile of the Company - PCO		
ID	**Questions**	**Type**
M1	Currently, how many employees work in the company? How many of them exclusively work in software development (analysis, implementation, testing, quality, project management, etc.)?	Context
M2	What agile practices you consider that the company uses in its software de-	Context

	velopment process? How long these practices have been used in the company?	
M3	What documents (artifacts) the team uses to develop software? Are these available to customers?	Context
M4	Is the validation of requirements done by means of frequent software deliveries or by means of documentation? Are the validations conducted face-to-face with customer or remotely?	Context
M5	In general, how long is each sprint (development cycle)? Are the validations of software with customer done at the end of each sprint? *Probe: If not, how often?*	Context
M6	How is it done the impact analysis when changes are requested in the software requirements?	Context
M7	Does the project hold information about the dependency relationships between requirements? *Probe: If yes, how is this information updated throughout the development?*	Context
M8	In your opinion, what are the factors that affect the quality of the SRS? *Probe: Why?*	Context
M9	Which good practices could be used in projects to improve the team performance? *Probe: How?*	Context
M10	Are the payments for the provision of customer's services related to the delivered functionality or are these payments based on fixed values no matter what is available in each delivery?	Context

Appendix D - Cross-case analysis and synthesis

Table D.1 – Profile of the software engineers interviewed

Software Engineers	Gender	Years in the Profession	Education	Functional Role
(a) Interviewees in Case Study 1				
#1	Female	13	University Graduate	System Analyst
#2	Female	10	University Graduate	System Analyst
#3	Female	18	University Graduate	System Analyst
#4	Male	9	Master Degree	Developer
#5	Male	9	Master Degree	Developer
#6	Male	8	Master Degree	System Analyst
#7	Female	10	University Graduate	Developer
#8	Male	11	University Graduate	Developer
#9	Female	9	University Graduate	Developer
#10	Female	9	Master Degree	System Analyst
#11	Female	5	University Graduate	Tester
#12	Female	16	Master Degree	Developer
#13	Male	16	Master Degree	Developer
#14	Female	13	University Graduate	System Analyst
(b) Interviewees in Case Study 2				
#1	Male	9	University Graduate	Developer and System Analyst
#2	Male	7	University Graduate	Developer and System Analyst
#3	Male	1	University Student	Developer
#4	Male	7	Specialist Degree	Developer and System Analyst
#5	Male	10	University Graduate	Developer and System Analyst
#6	Male	1	University Graduate	Developer
#7	Male	20	University Graduate	Developer
#8	Male	6	University Graduate	Developer and System Analyst
#9	Male	6	University Graduate	Developer
#10	Male	5	Master Degree	Developer
#11	Male	3	University Graduate	Developer and System Analyst
#12	Male	5	University Graduate	Developer
#13	Male	6	University Graduate	Developer
#14	Female	2	University Graduate	Developer
(c) Interviewees in Case Study 3				
#1	Male	3	University Graduate	Developer
#2	Male	6	Master Degree	Developer
#3	Male	4	University Graduate	Developer and System Analyst
#4	Male	5	Specialist Degree	Tester
#5	Male	3	Specialist Degree	Developer and System Analyst
#6	Male	4	University Graduate	Developer and System Analyst
#7	Male	3	University Graduate	Developer
#8	Male	2	University Graduate	Developer
#9	Male	4	University Graduate	Developer and System Analyst
#10	Male	3	University Graduate	Tester
#11	Male	2	University Graduate	Developer
#12	Male	1.5	Specialist Degree	Tester
#13	Male	6	University Graduate	Developer and System Analyst
#14	Male	5	University Graduate	Developer and System Analyst
#15	Female	4	Master Student	Tester
(d) Interviewees in Case Study 4				
#1	Male	8	University Graduate	Developer and System Analyst
#2	Male	6	Specialist Degree	Developer and System Analyst
#3	Male	7	University Graduate	Developer and System Analyst
(e) Interviewees in Case Study 5				
#1	Female	12	Master Degree	System Analyst
#2	Male	3	University Graduate	Tester
#3	Male	3	University Graduate	Developer
#4	Male	10	University Graduate	Developer and System Analyst
#5	Male	5	Specialist Degree	Developer and System Analyst
#6	Female	2	University Graduate	Tester
#7	Male	3	University Graduate	Developer
#8	Male	3	University Graduate	Developer
#9	Male	2.5	University Graduate	Developer
(f) Interviewees in Case Study 6				
#1	Male	18	University Graduate	Developer
#2	Male	36	Specialist Degree	Developer
#3	Male	15	Master Degree	Tester
#4	Male	15	Master Degree	Solution Architect

Table D.2 – Translation strategies for first level concepts adapted from FRANÇA et al. (2014)

Type of translation	Situation	Strategy
Identical	The same label and definition were used for a concept in cases studies	We simply repeated the concept as the translation
Renaming	Different labels were used for a concept in each study, but the definitions were equivalent	We chose the label that better expressed the meaning, consulting the thesaurus, dictionaries, and the literature to support the choice
Generalization	Different concepts were found in each case, with different names and definitions, but one concept could be interpreted as a generalization, or abstraction, that included one or more concepts in the others studies	We used the more general concept as the translation whenever it expressed the findings of case studies
Localization	A concept was found in one study but not in the others	We kept the concept as the translation with a remark that it was context dependent and associated the concept to its context
Refutation*	A concept in one study contradicted a concept in another study	We tried to understand and explain the refutation based on contextual data and added the explanation to the translation
Not applicable	A concept was found in some studies but is not directly related to the phenomenon.	The concept was not kept in the model

* We did not identify refutation instances in cross-case analysis

Table D.3 – Translation strategies for propositions adapted from FRANÇA et al. (2014)

Type of translation	Situation	Strategy
Reciprocal (RTA)	Propositions related the concepts by similar or comparable causal relationships	We identified the common aspects between the cases studies and translated into a consistent proposition with them
Refutational (RFA)*	Propositions related the concepts in opposing or contradicting relationships	Contextual information would be used to explain the refutations enriching our understanding of the varying conditions under which the local propositions would hold or fail
Line-of-argument (LOA)	Propositions related a different set of concepts, with enough intersection and without refutation, allowing the construction of a line-of-argument that could explain the different situations	Higher degree of inductive inference and interpretation was used to build the translations than in the reciprocal case
Localization (LOC)	Proposition not identified in all cases studies where the concepts were presented. Thus it is dependent on the context.	The proposition was maintained with a remark that represents a relationship dependent-context.

* We did not identify RFA instances in cross-case analysis

Appendix E - Interview Guide to evaluate the RSD approach

This appendix contains guides to interviewing the software engineers as data collection instrument for the empirical studies described in Chapter 7.

PRESENTATION

- Greetings and introduction.
- Thank the participant.
- Could you please tell me your full name and email?

PARTICIPANT INFORMATION

Name	
Email	

INTRODUCTION

The long term objective of our research is to assess how the team used RSD, which the best practices, difficulties and limitations of the proposed approach. We believe that this study is essential to improve the adoption of the approach in practice.

We would be very grateful for your input to this research. This would include participating in this interview. As a retribution for your participation, we will provide your company with an analysis of strengths and potential fragilities of the practices related to specifying of requirements in agile projects using RSD. This information may be useful for you and your team as you looks for opportunities to improve.

All your responses will be kept strictly confidential. Your participation is voluntary. You will not, in any way, be penalized if you choose not to participate in the study. Do you have any questions regarding this interview?

ABOUT THE ANSWERS

There are no rights or wrongs answers for most of the questions in this interview. My goal is to collect your impressions, opinions, and feelings about the various subjects addressed. So, please answer the questions as spontaneously and honestly as possible, knowing that your answers will not be, in any way, disclosed to other individuals inside or outside your company.

May we start?

Questions

Team Member Background - BCK		
ID	**Questions**	**Type**
Q1	What roles did you play in the project? ☐System Analyst ☐Developer ☐ Scrum Master ☐ Others____	Background
Q2	How long have you been working in this role?	Experience
Q3	How do you describe your experience in this role?	Experience
Q4	How were the requirements specified in the projects you worked on previously? *Probe: What approaches? Use Cases\Scenarios, Features, User Stories, Goals, Interface Prototype, Diagram of Classes, Diagram of Activities, Business Rules, …?*	Background
Q5	Would you describe what you think is the ideal SRS for the developer? *Probe: Why?*	Opinion

Now we will talk about the process of learning the RSD approach.

Learning of Approach- LEA		
ID	**Questions**	**Type**
Q6	How was the process of learning the RSD approach?	Opinion
Q7	What helped you in this process of learning approach? *Probe: Why?*	Opinion
Q8	What hindered you in this process of learning approach? *Probe: Why?*	Opinion

The next questions are about the structure and content of the RSD.

Structure and content of RSD - STR		
ID	**Questions**	**Type**
Q9	How do you assess the RSD structured through mockups, concepts and acceptance criteria (the type of information)? *Probe: Why?*	Opinion
Q10	On a scale of one (inadequate) to five (very adequate), how do you assess the structure of RSD (the type of information)? *Probe: Why?*	Opinion
Q11	Using a discrete scale (compliance, partial compliance, non compliance), how do you evaluate the compliance of RSD with each quality factor? Team-oriented Simplicity Consolidated Information Functional Requirements Technical aspects Acceptance Criteria Non-Functional Requirements Automated Support Traceability Change History *Probe: Why?*	Opinion
Q12	Regarding the structure (type of information), what are the differences between RSD and other approaches that you used before?	Opinion
Q13	Using a discrete scale (worse, equal, better) in each quality factor, how do you evaluate RSD approach compared to others approaches you have used before? Team-oriented Simplicity Consolidated Information Functional Requirements	Opinion

	Technical aspects Acceptance Criteria Non-Functional Requirements Automated Support Traceability Change History Completeness Clearness Objectivity Readability *Probe: Why?*	
Q14	Was the quality of the RSD different from what you expected? *Probe: What?*	

Now we will talk about how the activities of the development team were carried out using RSD.

Effect of RSD approach on Team Works - ETW		
ID	**Questions**	**Type**
Q15	How do you assess the strategy adopted to perform the acceptance tests? *Probe: Why?*	Opinion
Q16	How do you assess the reuse of requirements in the project? *Probe: Why?*	Opinion
Q17	How do you assess the tools used to support the requirements activities? *Probe: Why?*	Opinion
Q18	How is the customer's participation in the project? *Probe: Frequency? Adequate?*	Opinion
Q19	How is the collaboration between the stakeholders of the development team? *Probe: Frequency? Adequate?*	Opinion

Now we will talk about the effort required to use the RSD approach.

Effort required to using RSD – EFF		
ID	**Questions**	**Type**
Q20	Regarding the effort required, what are the differences between the RSD and other approaches you have used before?	Opinion
Q21	What practices used in the project hinder your performance? *Probe: How?*	Opinion
Q22	What practices used in the project improve your performance? *Probe: How?*	Opinion
Q23	Using a discrete scale (much, reasonable, little), how do you evaluate the effort required to use the RSD approach? *Probe: Why?*	Opinion
Q24	Using a discrete scale (lower, equal, higher), how do you evaluate the effort required to specify using RSD approach compared to others approaches you have used before? *Probe: Why?*	Opinion
Q25	Using a discrete scale (lower, equal, higher), how do you evaluate the effort required to code from RSD compared to others approaches you have used before? *Probe: Why?*	Opinion
Q26	Using a discrete scale (lower, equal, higher), how do you evaluate the effort required to test from RSD compared to others approaches you have used before? *Probe: Why?*	Opinion
Q27	Is the effort required to use RSD different from what you expected? *Probe: Why?*	Opinion

Now we will talk about the changes in requirements.

Changes in requirements - CHA		
ID	**Questions**	**Type**
Q28	How do you assess the strategy adopted to analyze the change requests in the requirements? *Probe: Why?*	Opinion
Q29	How do changes affect your work?	Opinion
Q30	How do you assess the impact of these changes compared to other approaches that you used previously? *Probe: Why?*	Opinion

Finally, we will talk about opportunities to improve the adoption of RSD approach.

Opportunities for improvement - IMP		
ID	**Questions**	**Type**
Q31	What changes would you like to make in RSD approach? *Probe: Why?*	Opinion
Q32	What are the practices you do not recommend using in future projects? *Probe: Why?*	Opinion
Q33	What practices do you recommend to use in future projects? *Probe: Why?*	Opinion
Q34	What other difficulties did you have using RSD approach? *Probe: Why?*	Opinion

Thanks

Thank you so much. Your participation was very important to this research.