PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
COMPUTER SCIENCE GRADUATE PROGRAM

QUALITY ASPECTS OF AGILE REQUIREMENTS

GABRIEL P.A DE OLIVEIRA

Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fullfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Sabrina Marczak

Porto Alegre
2017

CONTENTS

1. INTRODUCTION

Agile software development methodologies have become popular research areas. However, the understanding of how requirements engineering (RE) traditional practices are applied on this context is still barely explored, with only a few focused studies aiming to understanding the differences between the new and traditional RE and the problems that comes with them (like cited by Cao [CR08] and Heikkila et. al [HDLP15]).

Cohn [Coh04] says that software requirements is a communication problem on witch those who want the new software (either to use or to sell) must communicate with those who will build the new software.

User stories are used on agile methodologies to communicate software requirements. Beck and Fowler[Ken00] say that a user story is a chunk of functionality (some people use the word feature) that is of value to the customer and that agile teams demonstrate progress by delivering tested, integrated code that implements a story. For Jeffries [JAH00] they're promises for conversation that will take place between the customer and the programmers over the life of the story. User stories only capture the essential elements of a requirement: *who* it is for, *what* it expects from the system, and, optionally, *why* it is important [LDVB15].

Still, there's also the problem of communication *how* the system should behave. Gartner [Gar12] says that hardest job in software is communicating clearly about what we want the system to do. Acceptance Test-Driven Development (ATDD) helps with the challenge, as the whole team collaborates to gain clarity and shared understanding before development begins by writing tests in some specific formats that could later be automated.

Lucassen et. al [LDVB15] says that, despite the fact that user stories are a widely used notation for formulating requirements in agile development projects, little to no academic work is available on assessing their quality and his work helps to fill that gap. Still, we believe that analyzing the user stories quality without also judging its details, documented with ATDD, will yield an incomplete picture of how well requirements are being written on agile methodologies.

Therefore, the current work seeks to explore how acceptance tests quality is being evaluated, what quality aspects or characteristics were reused from requirements engineering traditional requirements on acceptance tests and what formats acceptance tests can assume when used. We hope to find appealing data that we could use on future empirical researches that verify if practitioners are writing good enough acceptance tests.

1.1 Volume organization

The remainder of this paper is organized as follows:

- CHAPTER 1 – introduces the current work subject;
- CHAPTER 2 – summarizes the theoretical background on user stories, acceptance tests and requirements quality;
- CHAPTER 3 – describes how this work was planned and the timetable for it;
- CHAPTER 4 – details the current work research methodology;
- CHAPTER 5 – details how the execution of the research was made;
- CHAPTER 6 – reports the results found and the answers to our research questions;
- CHAPTER 7 – concludes the current work with a summary on what was learned.

2. BACKGROUND

This chapter summarizes the theoretical background on some concepts used throughout the current work, like user stories, acceptance tests and requirements quality.

TBD: Review references usages

2.1 Traditional Requirements

A requirement is either a condition or capacity necessary to solve a problem or reach a goal for an interested party or some characteristic that a solution or component should possess or acquire in order to fulfill some form of contract [IIB09]. Additionally, a requirement is also a written representation of this condition or capacity, or a usable representation of a need focused on understanding what kind of value could be delivered to a client if a requirement is fulfilled [IIB15].

When classified according to their purpose, they can be called **business requirements**, **stakeholder requirements** and **solution requirements**. The first are statements of goals, objectives, and outcomes that describe why a change has been initiated, while the second are the needs of stakeholders that must be met in order to fulfill business requirements. The later describe the capabilities and qualities of a solution and provide the appropriate level of details to allow the proper implementation of a solution and can be divided into functional requirements (that describes the capabilities a solution must have in terms of the behaviour and information to manage) and non-functional requirements (that describes conditions under which a solution must remain effective).

2.1.1 Use Cases

Cockburn [Coc00] says that a use case captures a contract between the stakeholders of a system about its behavior and describes the system's behavior under various conditions by interacting with one of the stakeholders (the *primary actor*, who want to perform an action and achieve a certain goal). Different sequences of behavior, or scenarios, can unfold, and the use case collects together those different scenarios. They are used to express behavioral requirements for software systems and can be put into service to stimulate discussion within a team about an upcoming system. They might later use that the use case form to document the actual requirements. Besides the primary actor, that interacts with the system, a use case has other parts as well: the *scope* identifies the system that we are discussing, the *preconditions* and *guarantees* say what must be true before and after the use case runs, the *main success scenario* is a case in which nothing goes wrong and the *extensions section* describes what can happen differently during that scenario.

2.1.2 Traditional Requirements Quality

Requirements validation is a phase on traditional requirements engineering process that is known to support the three other activities (requirements elicitation, requirements analysis and requirements specification) by identifying and correcting errors in the requirements [HDLP15]. Génova [GFL+13] says that quality indicators must not provide numerical evaluations only, but first of all they must point out concrete defects and provide suggestions for improvement, just like a spell and grammar checker can help to improve the quality of a text. For Davis et. al [DOJ+93] a perfect software requirements specification is impossible, as some qualities may be achieved only on the expense of others. The author also implies that one must be careful to recognize that although quality is attainable, perfection is not.

The Business Analyst Body of Knowledge (BABOK) [IIB15] says that, while quality is ultimately determined by the needs of the stakeholders who will use the requirements or the designs, acceptable quality requirements exhibit many characteristics. The second edition [IIB09] describes eight characteristics a requirement must have in order to be a quality one, as follows: cohesion, completeness, consistency, correction, viability, adaptability, unambiguity and testability. The third edition [IIB15] bring nine: atomic, complete, consistent, concise, feasible, unambiguous, testable, prioritized and understandable. Both editions [IIB09] [IIB15] define what each characteristic means, but does not provide any measurement guidance.

Génova [GFL+13] lists other 11 properties along with analytical metrics that would later help the authors to build a quality framework and implement it on the requirements quality analyzer tool. The characteristics are as follows: atomicity, precision, completeness, consistency, understandability, unambiguity, traceability, abstraction, validability, verifiability, modifiability. The measurable indicators are: size, readability, punctuation, acronyms and abbreviations, connective terms, imprecise terms, design terms, imperative verbs, conditional verbs, passive voice, domain terms, versions, nesting, dependencies and overlappings.

The measurement formalism concern is shared by Davis et. al [DOJ+93], to whom a quality software requirements specification is one that contributes to successfully, cost-effective creation of software that solver real user needs and exhibits 24 quality attributes: unambiguous, complete , correct, understandable, verifiable, internally consistent, externally consistent, achievable, concise, design independent, traceable, modifiable, electronically stored, executable (interpretable), annotated by relative importance, annotated by relative stability, annotated by version, not redundant, at right level of detail, precise, reusable, traced, organized and cross-referenced. His work also define each attribute, provide ideas on measuring them, provide a recommendation of weight relative to other attributes and describe types of activities that can be used to optimize the present of each.

Use cases quality is discussed in details by Phalp et. al [PAJ+11], that summarizes prior works on that area (such as the rules found by Cockburn [Coc00]) and proposes refined rules based on discourse process theory, such as avoiding the use of pronouns, use active voice over passive one, achieve simplicity trough avoiding to use negative forms, adjectives and adverbs and use of discourse cues and the effect of readers background and goals. Also, desirable quality attributes of use cases are listed, that may be suited for certain project phases but not others as follows: standard format, completeness, conciseness, accuracy, logic, coherence, appropriate level of detail, consistent level of abstraction, readability, use of natural language and embellishment. The authors work create rules, that should be enforced and must be obeyed, and guidelines, which indicate an ideal that cannot always be followed, that could best produce those attributes.

ElAttar and Miller [EM12] present another list of use cases qualities attributes - this time, applied to use case diagrams. They are: consistency, correctness and completeness, fault-free, analytical and understandability. Along with the attributes, the authors had performed a systematic literature review and identified 61 unique guidelines, heuristics and rules for these format of requirements, that were synthesized and compiled into a set of 21 anti-patterns, a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences.

2.2 Agile Requirements

Agile software development methods take a different approach to requirements engineering and communication than the traditional requirement engineering approaches that is mostly based on formal documents and defined phases [HDLP15]. Trough the mapping of a systematic literature reviews on the subject, the authors have pointed out some benefits and challenges for requirement engineering on agile contexts.

Studies ([MAV+15], [PEM03], [HDLP15] and [ISM+15]) have shown that the requirements processes activities are executed in a continuous way, along with the product construction.

Even with the lack of a formal and well accepted definition for agile RE, Heikkila et. al [HDLP15] propose the following one:

"In agile RE, the requirements are elicited, analysed and specified in an ongoing and close collaboration with a customer or customer representative in order to achieve high reactivity to changes in the requirements and in the environment. Continuous requirements re-evaluation is vital for the success of the solution system, and the close

collaboration with the customer or customer representative is the essential method of requirements and system validation."

Paetsch et. al [PEM03] show that requirements validation on agile contexts focuses on frequent review meetings and acceptance tests. On the same paper, the lack of documentation is mentioned to potentially cause long-term problems for agile teams, such as improvement of knowledge loss when team members become unavailable and lack of training material to new members. We believe that the "good" quality of the minimal documentation generated by agile methodologies could mitigate those problems.

2.2.1 User Stories

Rizler [Rin09] says that traditional requirements methods have become too abstract and moved away from how people ordinarily learn and communicate - too far away from storytelling. Storytelling is something everyone understands intuitively, immediately improving the process of gathering information and structuring the requirements document. When you tell a story, you instinctively transform abstract knowledge into a logical structure. In his book, Rizler highlights that stories are the best way to communicate requirements and approaches the requirements writing process as a storytelling process - much in the way that agile methodologies use user stories.

Alexander [AM04] also describes how the stories have come to be used in software development. The author says that, some decades ago, business people, who couldn't understand technical computer details, could only communicate with computer technicians trough processes with a strong emphasis on contracts—rigid and precisely defined requirements strictly adhered to, that essentially provide little more than a means for pointing fingers when the development project inevitably failed. With stories came a necessary a common currency that tends to keep both parties honest and on their toes. If a system has users, the author continues, those users must communicate their expectations and needs and they will do so by telling you stories.

The user story format as a requirement's representation have been created along with the extreme programming (XP) methodology. Each user story is a short description of the behavior of the system, from the point of view of the user of the system [JAH00]. Beck and Fowler [Ken00] say that a story represents a feature customers want in the software, a story they would like to be able to tell their friends about this great system they are using. Alexander [AM04] says that user stories are expressed and documented using natural language prose: on the story card, in response to questions, and as part of the ongoing dialog with developers. The stories are descriptive and expressive of human desires and contain *what* is desired and *why* it's needed by the client. Due to the fact that stories focus on user needs rather than specific behavior details, we could compare that to business or stakeholders requirements from both versions of the BABOK [IIB09] [IIB15].

The main format of a user story, popularized by Cohn [Coh04], is shown below:

I as a (role) want (function) so that (business value)

Cohn [Coh04] expands the vision of user stories on his book. According to the author, a user story describes functionality that will be valuable to either a user or purchaser of a system or software and are composed of three aspects:

- a written description of the story used for planning and as a reminder
- conversations about the story that serve to flesh out the details of the story
- tests that convey and document details and that can be used to determine when a story is complete

Those aspects have come from Jeffries et. al [Jef01] terms Card, Conversation and Confirmation. On that work, the author described the **Card** represent customer requirements rather than document them, has just enough text to identify the requirement, and to remind everyone what the story is. The **Conversation** is an exchange of thoughts, opinions, and feelings. It is largely verbal, but can be supplemented with documents. The best supplements are examples; the best examples are executable, We call these examples confirmation. Last, the Confirmation, is told to the developers by the customer by telling them how she will confirm that they've done what is needed. She defines the acceptance tests that will be used to show that the story has been implemented correctly.

2.2.2 Acceptance Tests

According to Cao [CR08], acceptance tests are one of the most used agile requirements engineering practice. They document the conversations made between customer and the development team and details that have emerged from those conversations, in a similar way that solution requirements express the behaviors to fulfill stakeholders requirements.

For Cohn [Coh04], acceptance testing is defined as the process of verifying that stories were developed such that each works exactly the way the customer team expected it to work. Writing tests early is extremely helpful because more of the customer team's assumptions and expectations are communicated earlier to the developers.

Gartner [Gar12] says that the hardest job in software is communicating clearly about what we want the system to do and driving the development effort with acceptance tests helps with the challenge. Two core practices are described on his book, as follows: (a) before implementing each feature, team members collaborate to create concrete examples of the feature in action and (b) then the team translates these examples into automated acceptance tests, that become a prominent part of the team's shared, precise description of "done" for each feature. Teams that follow those practices are working on a Acceptance Test-Driven Development (**ATDD**) way.

The importance of acceptance tests is highlighted by Jeffries et. al [Jef01], who says that the confirmation provided by them is what makes possible the simple approach of card and conversation. When the conversation about a card gets down to the details of the acceptance test, the customer and programmer settle the final details of what needs to be done.

That importance is also mentioned by Cohn [Coh04] when he says that agreements between client and the development team are documented by tests that demonstrate that a story has been developed correctly. Those tests seems to make the user story more complete, as suggested by a quote from James Grenning where he noted that the text on a story card "plus acceptance tests are basically the same thing as a use case".

In addition to ATDD, there are other terms that also describes acceptance tests practices. There are teams that understand that executing acceptance testing manually will in most cases be tedious, expensive and time consuming and that automation of acceptance tests may thus seem as a promising initiative to ease and improve this process. In order to do that, it's necessary to document requirements and desired outcome in a format that can be automatically and repeatedly tested. The name of Automated Acceptance Tests (AAT) [HH08] is given to that idea, but we also found other different ones, like Executable Acceptance Test-Driven Development [MM07] and Story Test Driven Development [PM10].

2.2.3 Tabular format for Acceptance Tests

One of the most popular format to document acceptance tests described by Gartner [Gar12] is a table of examples like the ones found on the Framework for Integrated Tests (Fit). Haugset and Hanssen [HH11] say that Fit is an open source framework used to express acceptance test cases and a methodological tool for improving the communication between analysts and developers. Fit lets analysts write acceptance tests in the form of simple tables (named Fit tables) using HTML or even spreadsheets. A well-known implementation of Fit is FitNesse, substantially a Wiki where analysts/customers can upload requirements and related Fit tables. A Fit table specifies the inputs and expected outputs for the test.

2.2.4 Scenarios for Acceptance Tests

Another format to represent acceptance tests are scenarios, according to Gartner [Gar12]. Alexander [AM04] describes that the activity of building requirements scenarios encourages imagination and exploration and sets the stage for discovering unconscious and undreamed of requirements and that it is important because stories are the primary (perhaps only) means of communicating needs and desires, and providing critical feedback, to developers.

Scenarios are also known as key examples that describe the expected functionality [Adz11]. The living documentation presented by Adzic helps to validate if the system works in the same way reflected by the documentation that represents it.

Karner [Kan03] describes scenarios as a hypothetical story used to help a person think through a complex problem or system. Scenarios also helps to learn the product, as people don't learn well by following checklists or material that is organized for them - they learn by doing tasks that require them to investigate the product for themselves. The author also brings the idea that a scenario is an instantiation of a use case—take a specific path through the model, assigning specific values to each variable. Finally, as the scenario is a story about someone trying to accomplish something with the product under test, it can help to expose failures to deliver desired benefits.

Still, scenarios can be used as an ubiquitous language that business people can understand to describe and model a system [Sma14]. The Behavior-Driven Development (**BDD**) practice uses it with this intent, by expressing them in a format known as Gherkin, that is designed to be both easily understandable for business stakeholders and easy to automate using dedicated tools. Those scenarios are referred as structured examples, in a similar way as Adzic [Adz11] call them key examples.

In Gherkin, the requirements related to a particular feature are grouped into a single text file called a feature file. A feature file contains a short description of the feature, followed by a number of scenarios, or formalized examples of how a feature works. Each scenario is made up of a number of steps, where each step starts with one of a small number of keywords. The natural order of a scenario is Given ... When ... Then, such as shown below:

Given describes the preconditions for the scenario and prepares the execution environment;

When describes and performs an action;

Then describes the expected outcomes and performs the verifications to validate them.

2.3 Traditional Requirements Quality Characteristics

Requirements validation is a phase on traditional requirements engineering process that is known to support the three other activities (requirements elicitation, requirements analysis and requirements specification) by identifying and correcting errors in the requirements [HDLP15]. While quality is ultimately determined by the needs of the stakeholders who will use the requirements or the designs, acceptable quality requirements exhibit many of the following characteristics described on the the Business Analyst Body of Knowledge. The second edition describes eight characteristics a requirement must have in order to be a quality one, as follows: cohesion, completeness, consistency, correction, viability, adaptability, unambiguity and testability. The third edition [IIB15] bring nine: atomic, complete, consistent, concise, feasible, unambiguous, testable, prioritized and understandable.

2.3.1 User Stories Quality Characteristics

Jeffries et. al [JAH00] highlight many characteristics that made up a good user story. They must be understandable by the team and valuable to the customer. Since it represents a concept and a promise to conversations, it should be as small and short as possible, not a full detailed specification. It's also noted that they should be small enough, so a work cycle can deliver a few of them and also to help on the estimation of the effort. Other desired characteristics are that they should be as independent as possible, so the team can be more flexible to deliver them in any order, and that they should be testable, so the team can easily demonstrate they were integrated in the system in a correct way.

For Alexander [AM04], it also matters that they're estimable and testable. Besides, the author says they should be discrete and describe a single bit of functionality, a single feature, or an expectation the customer has of the system. In addition to those characteristics, it should also be prioritized, as the customer must be able to determine which stories are more important, more likely to generate an immediate business benefit, or otherwise more valuable to the business.

For Duncan [Dun01], the quality of requirements in the user story format is evaluated on XP methodology perspective. This informal evaluation took into consideration the other 24 quality attributes found on traditional requirements formats. However, it seems that no empirical evaluation of his opinions have been made.

On the other hand, Lucassen et. al [LDVB15] created a framework to evaluate user stories and have performed some kind of empirical evaluation. However, they are focused on only the first aspect of user stories described by Cohn [Coh04]: the written description of it, the Card. This gap of analysis is one of the reason that have motivated the current study.

Lastly, Heck and Zaidman [HZ15] have created another requirements quality framework, more focused on agile requirements used on open source projects. One future work raised by him is the need to evaluate the quality of requirements expressed as a set of example, that also motivated the current work.

2.3.2 Acceptance Tests Quality Characteristics

We have found only a few publications that cover what is needed to write a good acceptance test.

Glinz [Gli00] suggest that scenarios have the potential for improving the quality of requirements and that they have a strong positive impact on the qualities of adequacy, partial completeness, modifiability and verifiability – as long as the scenarios are used properly. However, no empirical studies or metrics were found to verify his thoughts on how formal scenarios need to be in order to effectively improve the mentioned characteristics.

Kaner [Kan03] describes that a test based in a scenario has five characteristics, as follows: it is (a) a story that is (b) motivating, (c) credible, (d) complex, and (e) easy to evaluate. Empirical evaluation of existing acceptance test cases expressed as scenarios according to his characteristics weren't found.

Other characteristics are raised by Smart [Sma14], as follows: the scenarios steps expressiveness, focused on what goal the user want to accomplish and not on implementation details on on screen interactions; the use of preconditions on the past tense, to make it transparent that those are actions that have already occurred in order to begin that test; the reuse of information to avoid unnecessary repetition of words; and the scenarios independence. The author specify examples of good and bad scenarios in order to demonstrate those characteristics.

Melnik [MRM04] haven't highlighted characteristics of good quality requirements on tabular format. Instead, they focus were on problems that those requirements didn't possessed - as fewer problems appears on this format, they are judged better suited for developers to work with. However, the authors conclusion about the lack of problems shown on FIT tables is not based on the documentation analysis, but in other indirect factors (easy to learn, lack of problems on development, etc).

3. PLANNING

According to [KC07], a systematic literature review is a means of identifying, evaluating, and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest. The most common reasons for undertaking it are described below:

1. Summarize the existing evidence concerning a treatment or technology;
2. Identify any gaps in current research in order to suggest areas for further investigation;
3. Provide a framework/background in order to appropriately position new research activities.

The current work is motivated by the first item, as it aims to summarize the *state of the art* on how acceptance tests are measured regarding their quality.

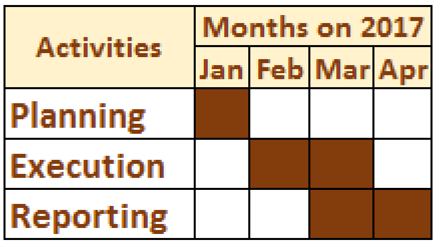
3.1 Review Process

We follow the three stages that [KC07] described to define the current work phases, as follows:

1. Planning stage, covered on Chapter 4
   (a) Specifying the research question(s)
   (b) Developing a review protocol
2. Execution stage, covered on Chapter 5
   (a) Identification of research
   (b) Selection of primary studies
   (c) Study quality assessment
   (d) Data extraction and monitoring
   (e) Data synthesis
3. Reporting stage, covered on Chapter 6
   (a) Formatting the main report
   (b) Evaluating the report

3.2 Review process timetable

Based on the before mentioned stages, a timetable was defined as follows below



Activities	Months on 2017			
	Jan	Feb	Mar	Apr
Planning	🟫			
Execution		🟫	🟫	
Reporting			🟫	🟫

Figure 3.1 – Review process timetable

4. SYSTEMATIC REVIEW PROTOCOL

4.1 Research Questions

[KC07] warn us that the critical issue in any systematic review is to ask the right question. In our context, the right question seems to move towards a direction that's important to both research team and practitioners: to understand how the academic publications evaluate acceptance tests may help to broaden our understanding of what takes to write a good one or what pitfalls to avoid in order to not create bad ones.

With that goal in mind, the research questions that we aim to answer are:

RQ1 What are the formats used to describe requirements on agile processes?

RQ2 What aspects are used to evaluate agile requirements quality?

4.2 Research Method

A review protocol specifies the methods that will be used to undertake a specific systematic review, in order to reduce the possibility of researcher bias and to avoid the selection of individual studies or the analysis based on researcher expectations [KC07].

4.2.1 Search Strategy

In order to focus our efforts on the analysis of the results found, we have used a automated searches on the digital repositories known to concentrate most of the requirements enginnering research - IEEExplore Library, ACM Library and Scopus. While the first two are well known sources of academic publication, Scopus [1] was included due to their claims to be the largest abstract and citation database of peer-reviewed literature. On all three search sources, the search was made on the title, the abstract, and the keywords of the publications.

4.2.2 Search criteria

Our research questions had put in evidence the words "quality" and "agile requirements". We took inspiration from [TG15] and [EM12] to break the word "quality" and from the work

[1]Scopus - https://www.scopus.com/

of [MAV+15] to break the word "agile requirements". However, the later haven't added "behaviour driven development" explicitly, so we took the liberty to add it as we believe this can be an important way to represent requirements (as highlighted on Chapter 2).

As our search was made on different sources, not all the terms were used all the time. ACM and Scopus allowed us to use them all, but IEEE restricted the number of search terms to be used. The ones in bold, below, were selected as the most representative ones and thus were used on all the three sources.

1. **Requirements group: "requirements engineering"**, **"use case"**, "story", **"user stories"**, "feature", "specifications", "textual descriptions", "templates", "models", **"framework for integrated tests"**, "fitnesse", "living documentation", "executable documentation", "scenario", "scenarios", **"behavior driven development"**, "bdd";
2. **Quality group:** "quality", **"validation"**, "criteria", **"heuristics"**, **"guidelines"**, **"anti-patterns"**, "patterns", **"rule"**, **"pitfalls"**, "classification", **"assessment"**, **"checklist"**;
3. **Agile group:** "agile methodology", "agile methodologies", **"agile"**, "scrum", "extreme programming", "feature driven development", "lean software development", "adaptive software development".

4.2.3 Study Inclusion Criteria

In order to provide an initial filter on the result obtained from the search query, the following inclusion criteria were adopted:

- Studies focused on describing requirements quality with qualitative or quantitative nature;
- Studies focused on evaluating requirements quality despite if focusing on teams or projects;
- Studies focused on listing aspects of requirements quality.

4.2.4 Study exclusion criteria

A paper was excluded from our results if at least one of the exclusion criteria was complied, as follows:

- Studies not related with requirements (that focus on product quality, work processes and methodologies for example);
- Studies not related with requirements quality (that focus on non-functional requirements, also called quality requirements, or on requirements views that are not quality related, as prioritization or estimation);

- Studies that do not define their context as agile development (projects or teams);
- Studies not written in English.

4.2.5 Control papers

Through exploratory manual searches, two control papers were elected as control papers (the ones that should appear on the searches to guarantee they're directed on the correct direction).

Table 4.1 – Control papers

Reference	Year	Title
[LDVB15]	2015	Forging high-quality User Stories: Towards a discipline for Agile Requirements
[HZ15]	2015	Quality criteria for just-in-time requirements: just enough, just-in-time?