PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

GABRIEL PIMENTEL AFFONSO DE OLIVEIRA

Question-based Checklist to Evaluate BDD Scenarios' Quality

Porto Alegre
Ano 2018

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

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

QUESTION-BASED CHECKLIST TO EVALUATE BDD SCENARIOS' QUALITY

GABRIEL P.A DE OLIVEIRA

Dissertation presented as partial requirement for obtaining the degree of Master in Computer Science at Pontifical Catholic University of Rio Grande do Sul.

Advisor: Prof. Sabrina Marczak, PhD

Porto Alegre
2018

MINUTES NO. 527

On Mar 14, 2018, room 506.2 of Building 32, at Pontifícia Universidade Católica do Rio Grande do Sul's Main Campus, was home to the 527th Masters Dissertation defense of the Graduate Program in Computer Science. Student Gabriel Pimentel Affonso de Oliveira defended the dissertation entitled "QUESTION-BASED CHECKLIST TO EVALUATE BDD SCENARIOS QUALITY". The Examining Committee comprised the following professors: Dra. Maya Daneva (University of Twente), Dra. Milene Selbach Silveira (PPGCC/PUCRS) and Dra. Sabrina Dos Santos Marczak (PPGCC/PUCRS), as the advisor and chair of this Committee. The Examining Committee _____ the student concerning this requirement in view of the title of Master of of Computer Science.

There is nothing else to report as I have entered this record in the capacity of Chair of this Committee, which was signed by me and the other members as well.

SABRINA DOS SANTOS MARCZAK

MAYA DANEVA

MILENE SELBACH SILVEIRA

Ficha Catalográfica

O48q Oliveira, Gabriel Pimentel Affonso de

Question-based Checklist to Evaluate BDD Scenarios' Quality / Gabriel Pimentel Affonso de Oliveira . – 2018.
82 f.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientadora: Profa. Dra. Sabrina dos Santos Marczak.

1. Behavior-Driven Development. 2. requirements quality. 3. quality inspection. 4. reading technique. I. Marczak, Sabrina dos Santos. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecário responsável: Marcelo Votto Texeira CRB-10/1974

"The difference between something good and something great is attention to detail."
(Charles R. Swindoll)

ACKNOWLEDGEMENTS

I would like to thank all of those who have helped to manage the struggles of this long adventure.

Above all, I thank my wife, my parents, and my sister for all their immense patience, understanding, and love throughout this journey. Without your support, it would not be possible to navigate in the troubled waters of research.

To my advisor, Sabrina, for the flexible and broad orientation that allowed timetables, timezone and geographic differences to be greatly reduced.

To all my research colleagues, for the discussions, ideas and moral support during all the period of my Master's degree.

To the students and interview participants in the course of the research, by the time dedicated and contribution.

To PUCRS and the Faculty of the Computer Science School, for all the infrastructure offered during this work.

LISTA DE VERIFICAÇÃO BASEADA EM QUESTÕES PARA AVALIAR A QUALIDADE DE CENÁRIOS BDD

RESUMO

Tradicionalmente, a engenharia de requisitos se baseia na execução sequencial de atividades. Por outro lado, a engenharia de requisitos em metodologias ágeis é informal. Projetos ágeis são bem sucedidos "sem requisitos" graças ao fato de que casos de teste são comumentemente vistos como requisitos e de que requisitos são detalhados como casos de teste que servem tambem para validar e aceitar cada funcionalidade. Um dos formato destes testes de aceitação são cenários criados a partir da técnica de desenvolvimento orientado a comportamento (do inglês, behavior-driven development, BDD). Estes cenários ajudam a evitar problemas de comunicação entre especialistas de domínio e programadores, já que estes cenários são escritos numa linguagem comum a esses dois grupos, permitindo um caminho menos ambíguo dos requisitos de negócio para a especificação do comportamento do um software. Entretanto, aqueles que formalizam cenários BDD não possuem um conjunto padrão de regras para se familiarizarem com o conceito de um "bom" cenário, o que pode permitir que cenários BDD sofram de problemas conhecidos pela engenharia de requisitos, tais como requisitos incompletos, mal especificados ou inconsistentes. Portanto, para preencher essa lacuna, nessa pesquisa foram coletados dados de entrevistas semi-estruturadas com praticantes de BDD para propormos uma lista de verificação baseada em questões com 12 perguntas associadas a 8 atributos de qualidade. Esse instrumento deve prover aos praticantes de BDD orientações padronizadas para o refinamento de seus cenários.

Palavras Chave: Behavior-Driven Development, qualidade de requisitos, técnica de leitura.

QUESTION-BASED CHECKLIST TO EVALUATE BDD SCENARIOS' QUALITY

ABSTRACT

Traditional requirements engineering approaches are based on a sequential execution of activities. In the other hand, requirements engineering in agile development is informal. Agile projects succeed "without requirements" due to the fact that test cases are commonly viewed as requirements and detailed requirements are documented as test cases that also validate and accept each feature. One format of those acceptance test cases is Behavior-Driven Development scenarios. Those scenarios help to avoid communication problems between the domain experts and programmers on the team, as they are defined using a common language that allows for an easy, less ambiguous path from end-user business requirements to the specification of how the software should behave. However, those who formalize BDD scenarios do not have a standard set of rules to educate themselves on what a "good" BDD scenario is, which can allow BDD scenarios to suffer from other known problems in requirement engineering such as incomplete, underspecified and inconsistent requirements. Therefore, to fill that gap, this research gathered data from semi-structures interviews performed with BDD practitioners to propose a question-based checklist based on 8 newly defined quality attributes. This question-based checklist provides practitioners with an standard guideline for BDD scenarios' refinement.

Keywords: Behavior-Driven Development, requirements quality, quality inspection, reading technique.

CONTENTS

1 INTRODUCTION 11
1.1 RESEARCH GOAL 12
1.2 RESEARCH METHOD OVERVIEW 12
1.3 MAIN CONTRIBUTIONS 13
1.4 DOCUMENT OUTLINE 13

2 BACKGROUND 14
2.1 TRADITIONAL REQUIREMENTS 14
2.1.1 TRADITIONAL REQUIREMENTS QUALITY 14
2.1.2 REVIEWS AND INSPECTIONS 16
2.1.3 READING TECHNIQUES 16
2.2 AGILE REQUIREMENTS 18
2.2.1 USER STORIES 19
2.2.2 ACCEPTANCE TESTS 20
2.2.3 BEHAVIOR-DRIVEN DEVELOPMENT 22
2.2.4 AGILE REQUIREMENTS QUALITY 26
2.3 CONCLUSION 26

3 RESEARCH METHODOLOGY 28
3.1 RESEARCH OBJECTIVES 28
3.2 RESEARCH DESIGN 28
3.2.1 PILOT STUDY 30
3.2.2 INTERVIEW'S PARTICIPANTS SELECTION 31
3.2.3 INTERVIEW DESIGN 32
3.2.4 DATA ANALYSIS 34
3.2.5 THREATS TO VALIDITY 36

4 INTERVIEW RESULTS 37
4.1 PARTICIPANTS' PROFILES 37
4.2 QUALITY ATTRIBUTES' INTERPRETATION 38
4.2.1 CONCISE 39
4.2.2 SMALL 42
4.2.3 TESTABLE 44
4.2.4 UNDERSTANDABLE 46
4.2.5 UNAMBIGUOUS 48
4.2.6 VALUABLE 50

4.2.7 REMOVED ATTRIBUTES 52
4.2.8 ADDITIONAL ATTRIBUTES 53
4.3 PARTICIPANT'S CRITERIA 56
4.3.1 LANGUAGE 57
4.3.2 STEPS 59
4.3.3 TITLES 61
4.3.4 ADDITIONAL CRITERIA 63
4.4 NEWLY-DEFINED QUALITY ATTRIBUTES 66
4.4.1 ESSENTIAL 68
4.4.2 FOCUSED 69
4.4.3 SINGULAR 69
4.4.4 CLEAR 69
4.4.5 COMPLETE 70
4.4.6 UNIQUE 70
4.4.7 UBIQUITOUS 70
4.4.8 INTEGROUS 71

5 PROPOSED QUESTION-BASED CHECKLIST 72

6 FINAL CONSIDERATIONS 75
6.1 SUMMARY OF RESULTS 75
6.2 LESSONS LEARNED 75
6.3 PUBLICATIONS 76
6.4 LIMITATIONS 76
6.5 FUTURE WORK 77

REFERENCES 78

LIST OF FIGURES

Figure 2.1 – Use case example. Source: [36]. 15
Figure 2.2 – Cockburn's use case questionnaire. Source: [4] 18
Figure 2.3 – Scenario example 23
Figure 2.4 – Feature file example 23
Figure 2.5 – Typical stack of a BDD tool such as Cucumber. Source [38] 24
Figure 2.6 – Typical stages of a feature on a BDD process. Source [35] 25
Figure 3.1 – Research Design 29
Figure 3.2 – Survey Answers' Statistics 32
Figure 3.3 – Thematic synthesis process. Source [7] 34
Figure 3.4 – Levels of interpretation in thematic synthesis. Source [7] 36
Figure 4.1 – Overview of the attributes to analyze BDD scenarios 39
Figure 4.2 – Concise attribute interpretations 39
Figure 4.3 – Scenario with a more concise version 40
Figure 4.4 – Partial mentions feature and a more concise version 41
Figure 4.5 – Scenario's more concise version using declarative language 41
Figure 4.6 – Small attribute interpretations 42
Figure 4.7 – Not a small scenario due to the lengthy line 8 43
Figure 4.8 – Scenario that tests two things and a more atomic version 43
Figure 4.9 – Testable attribute interpretations 44
Figure 4.10 – Testable scenario due to the ability to follow the steps 45
Figure 4.11 – Understandable attribute interpretations 46
Figure 4.12 – Not understandable scenario due to the CSS element 47
Figure 4.13 – Unambiguous attribute interpretations 48
Figure 4.14 – Valuable attribute interpretations 50
Figure 4.15 – New attributes 54
Figure 4.16 – Profile photos feature 55
Figure 4.17 – Overview of the criteria to analyze BDD scenarios 57
Figure 4.18 – Language criteria to analyze BDD scenarios 57
Figure 4.19 – Steps criteria to analyze BDD scenarios 60
Figure 4.20 – Scenario alternating When/Then steps 60
Figure 4.21 – Title criteria to analyze BDD scenarios 61
Figure 4.22 – Example of a Feature description on comments.feature 61
Figure 4.23 – Breaking hovercards scenarios to allow better titles 63
Figure 4.24 – Additional criteria to analyze BDD scenarios 63

LIST OF TABLES

Table 3.1 – Cross-activity design of pilot study 31
Table 3.2 – Interview Questions 33
Table 4.1 – Participants' profiles 37
Table 4.2 – How participants use BDD scenarios 38
Table 4.3 – Summary of removed attributes 52
Table 4.4 – Newly-Redefined attributes mapping to characteristics 67
Table 4.5 – Summary of the newly defined quality attributes 68
Table 5.1 – Question-based checklist for BDD scenarios 73

1. INTRODUCTION

Traditional requirements engineering (RE) approaches, sometimes known as up-front RE approaches, are based on a sequential execution of elicitation, analysis, specification, verification and validation activities [16]. On the other hand, requirements engineering in agile development is informal and based on the skills and tacit knowledge of individuals [16]. Based on observations of 3 companies, agile development projects often manage well without extensive requirements due to the fact that test cases are commonly viewed as requirements and detailed requirements are documented as test cases [3]. One format of test cases as requirements identified on Bjarnason et. al [3] study is Behavior-Driven Development scenarios.

Behavior-Driven Development (BDD) is an umbrella term to describe a set of practices that uses scenarios as an ubiquitous language to describe and model a system [34]. BDD scenarios, a known format of acceptance tests [11], are expressed in a format known as Gherkin, which is designed to be both easily understandable for business stakeholders and easy to automate using dedicated tools [34]. Smart [34] states that bringing business and technical parties together to talk about the same document helps to build the right software (the one that meets customer needs), a thought reinforced by Wynne and Hellesoy [38] when saying that acceptance tests ensure a team to *build the right thing*.

Wynne and Hellesoy [38] understand that many software projects suffer from low-quality communication between the domain experts and programmers on the team, a known RE problem [10]. BDD scenarios help to avoid this problem by building scenarios in a common language that allows for an easy, less ambiguous path from end-user business requirements to scenarios that specify how the software should behave and guide the developers in building a working software with features that really matter to the business [34].

To the best of our knowledge, writers of BDD scenarios, those who formalize BDD scenarios on software development teams, do not have a standard set of rules to educate themselves on what a "good" BDD scenario is. They can only compare their work with a few guidelines and examples of "good" and "bad" scenarios found on Smart's book [34], Wynne and Hellesoy's book [38], and other informal internet references. This comparison can be misguided, as the writer's application context may not be comparable to the books' examples context, and incomplete, as the few guidelines are neither a set of enforcement rules [30] nor a questionnaire, as the one existent for use cases [4].

Due to that fact, we fear that BDD scenarios mitigation of RE communication problems, during the discovery and definition of features [35], may be lost due to unguided formalization of those features in the form of BDD scenarios, which may lead to other known problems in requirement engineering such as incomplete, underspecified and inconsistent requirements [10].

We believe that structuring the tacit knowledge of BDD practitioners could enhance the already existing guidelines from other sources (e.g., [34][38]) and the practitioners' ability to evaluate their own BDD scenarios' quality.

1.1 Research Goal

To avoid BDD scenarios' unguided formalization to suffer from problems such as incomplete, underspecified and inconsistent requirements [10], the main goal of this research is to provide a set of criteria suited to evaluate BDD scenarios and the proper guidance to use those criteria effectively to generate the input of scenarios' refinement sessions. To achieve that, we first identified the quality attributes that describe a good BDD scenario and later arranged them into a guideline accessible to be used by members of a software development team. Our proposed question-based checklist is similar to the one used by Cockburn on use cases [4].

1.2 Research Method Overview

To accomplish our goal, an empirical qualitative-based research was conducted, using thematic analysis to derive insights from a field study using semi-structured interviews.

To avoid our findings to suffer from the effect of practitioners' plurality of terms when describing similar good or bad practices, we guided our interviews with a sub-set of literature-informed quality attributes with the goal to to motivate participants to think about those unified terms, criticize them, suggest better alternatives, and map their own criteria to those terms in the hope that those literature quality attributes are judged appropriated to evaluate BDD scenarios.

Our initial list of quality attributes came from a literature review, which stated that traditional requirements' quality attributes [17][18] and the INVEST[5] acronym were used with agile requirements. In order to acquire some knowledge about how evaluators judge the quality of BDD scenarios using those attributes, we organized a pilot study with 15 graduate students. The output of this study [26] was the sub-set of literature-informed quality attributes that we used to guide the interviews and acquire 18 practitioners' opinions.

Additionally, to aid practitioners realize their own quality criteria, we asked them to evaluate real-life examples of BDD scenarios, taken from the Diaspora[1] open source project. If they were short on answers, we provoked their thoughts with a list of criteria taken from Smart's experiences [34] and Wynne and Hellesoy's experiences [38], the only informal references available we know of.

Those literature-informed quality attributes and criteria from Smart [34] and Wynne and Hellesoy [38] experiences were used as as the initial set of codes for our thematic analysis. We refined these codes with the practitioners' interpretations of the literature-informed quality attributes and their own personal criteria during our cyclical analysis, resulting in themes represented by the newly-labeled quality attributes that compose our proposed question-based checklist as presented in this thesis work.

[1] https://diasporafoundation.org/

1.3 Main Contributions

This thesis proposes a question-based checklist (main contribution), organized into three scope levels (feature, scenario and steps level), presented along with a proof of concept to demonstrate its feasibility in Chapter 5. The questions from that question-based checklist are based on newly-defined quality attributes (second main contribution) identified in the practitioner's interviews, presented in Chapter 4.

In addition, this thesis also contributes to the literature as follows:

- *On the Empirical Evaluation of BDD Scenarios Quality: Preliminary Findings of an Empirical Study* paper reported in the Workshop on Empirical Requirements Engineering in conjunction with the 2017 International Requirements Engineering Conference [26];
- *On the Understanding of BDD Scenarios' Quality: Preliminary Practitioners' Opinions* paper reported in the 2018 International Working Conference on Requirements Engineering [27].

1.4 Document Outline

The remainder of this document is organized as follows:

- **Chapter 2 - Background:** presents the theoretical background on traditional and agile requirements and how their quality can be evaluated;
- **Chapter 3 - Research Methodology:** presents the research methodology, describing the adopted research, data collection, and analysis methods;
- **Chapter 4 - Interview Results:** describes the key findings from the interviews with practitioners in the form of newly-redefined quality attributes;
- **Chapter 5 - Proposed Question-Based Checklist**: presents the question-based checklist organized by distinct levels of abstraction (feature, scenario and steps) and by the newly redefined quality attributes. A proof of concept is used to exemplify the feasibility of use of the proposed checklist;
- **Chapter 6 - Final Considerations**: explores the major contributions, inherent limitations, and outlines possible future research in the area.

2. BACKGROUND

Before delving into different ways of writing "good" BDD scenarios, this chapter presents some concepts such as the criteria to evaluate traditional requirements quality, how that evaluation is performed, what are agile requirements, how BDD is connected to it, and what a BDD scenario is.

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 [17]. In our research, we focus on the later, solution requirements, which describes the capabilities of a solution and provide the appropriate level of details to allow the proper implementation of that solution. More precisely, we focus on functional requirements, that describes the capabilities a solution must have in terms of system behaviors.

One common format to represent solution functional requirements is through use cases. Cockburn [4] states 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 wants to perform an action and achieve a certain goal). In addition to the primary actor, who interacts with the system, a use case consists of: the *scope*, that identifies the system that under construction; the *preconditions*, that state what must be true before and after the use case runs; the *main success scenario*, where a usage scenario where nothing goes wrong is described; and the *extensions section*, that describes what can happen differently during that scenario.

A single use case example can be found in Figure 2.1, provided by Williams et. al [36]. Note that some of the elements described by Cockburn [4] are missing – namely the scope and the main actor. Having missing elements is accepted by Cockburn, who states that one should select a format that the writers and the readers can all be comfortable with and not worry too much to adhering to an standard model.

2.1.1 Traditional Requirements Quality

Requirements validation is a phase on traditional RE process that is known to support requirements elicitation, requirements analysis and requirements specification activities by identifying and correcting errors in the requirements [16]. Génova [12] states 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 [8], a perfect software requirements specification is impossible as some qualities

```
Use Case: Withdraw Money
Precondition: ATM Customer must be logged onto system.
1.  ATM Customer selects Withdraw.
2.  System requests amount.
3.  ATM Customer enters amount.
4.  System dispenses cash.
5.  System debits amount from the account balance.
Exceptions:
3.1 [amount greater than account balance]
  3.1.1  System displays balance exceeded warning
  3.1.2  If ATM Customer chooses "continue" rejoin at 3
  3.1.3  Else terminate use case
Postconditions:
    Successful withdrawal:
   account.balance = account.balance@pre - amount
    Unsuccessful withdrawal:
   account.balance is unchanged.
```

Figure 2.1 – Use case example. Source: [36].

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) [18] states 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. BABOK's second edition [17] describes eight characteristics a requirement must have in order to be a quality one, as follows: adaptability, cohesion, completeness, consistency, correction, testability, unambiguity, and viability. BABOK's third edition [18] brings nine: atomic, complete, consistent, concise, feasible, prioritized, testable, unambiguous, and understandable. Both editions [17][18] define what each characteristic means, but do not provide any measurement guidance.

Use cases quality is discussed in details by Phalp et. al [30]. Their study summarizes prior work on use cases quality (such as the rules found by Cockburn [4]) 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, 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 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.

El-Attar and Miller [9] present another list of use cases qualities attributes - this time, applied to use case diagrams. The list contains the following attributes: 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.

2.1.2 Reviews and Inspections

BABOK's 3rd edition [18] argues that one way to validate quality characteristics is through a review, as they can help identify defects early in the work product life cycle, eliminating the need for expensive removal of defects discovered later in the life cycle.

Melo et. al [25] state that software review activities can be applied at many points during the software development process and can be used to discover defects in any type of deliverables or internal work products. Thus, software review activities have the "purification effect" of software artifacts as an objective. Inspection, a form of review, is a rigorous defect detection process. The advantages of this inspection review process are: there is an effective mistake detection mechanism; qualitative and quantitative project feedback is given earlier; and a record of the inspection is kept, allowing the evaluation of the task performed [25].

Laitenberger [23] expects that inspection results depend on inspection participants themselves and their strategies for understanding the inspected artifact. Therefore, supporting inspection participants (inspectors) with particular techniques that help them detect defects in software products may increase the effectiveness of an inspection team. Such techniques are referred as reading techniques.

Zhu [39] describes software inspection as a formalized peer review process applicable to any software artifact. According to him, it is widely established that software inspection is effective (it finds many defects), efficient (low cost per defect), and practical (easy to carry out). The immediate benefit of software review is to detect and fix errors or problems in software artifacts, which become more readable and maintainable [39].

Zhu [39] states that software review is primarily an individual effort and the kinds of reading techniques the individual uses are paramount to the outcome and effectiveness of software review. This thought is reinforced by Shull and colleagues [33], as these authors state that, while typical inspections require individuals to review a particular artifact to next meet as a team to discuss and record defects, empirical evidence has questioned the importance of team meetings by showing that meetings do not contribute to finding a significant number of new defects that were not already found by individual reviewers.

2.1.3 Reading Techniques

A software reading technique is a series of steps for the individual analysis of a textual software product to achieve the understanding needed for a particular task - thus demonstrating that reading has a purpose and is an individual activity [39] [33]. For Melo et. al [25], those techniques

attempt to capture knowledge about best practices for defect detection into a procedure that can be followed and also to achieve the understanding needed for a particular task. In a sense, a reading technique can be regarded as a mechanism or strategy for the individual inspector to detect defects in the inspected product [23].

Shull et. al [33] declare that software reading techniques can be applied to improve the effectiveness of software inspections by improving the effectiveness of individual reviewers. Zhu [39] further defines that a series of steps is a set of instructions that guide the reader how to read the artifacts, what areas to focus on and what problems to look for, while Shull et. al [33] define them as two components: (a) a concrete procedure that can be followed to focus on only the information in the review document that is important for the quality aspects of interest, and (b) questions that explicitly ask the reader to think about the information just uncovered in order to find defects. For the purpose of our research, we use Shull et. al's definition [33] to motivate the organization of our proposed question-based checklist.

The most widely used format of reading technique are checklists [39]. Zhu defines it as a list of questions to provide reviewers with hints and recommendations for finding defects during the examination of software artifacts. Since a question can be rephrased as an imperative sentence, the checklist does not have to be composed of questions only - matching his flexible definition for reading techniques. One example of checklist is the questionnaire used by Cockburn [4] to evaluate the quality of Use Cases, shown in Figure 2.2.

Laitenberger [23] found that ad-hoc reading and checklist-based reading are the most popular reading techniques used today for defect detection during inspection. Ad-hoc reading offers very little reading support at all since a software product is simply given to inspectors without any direction or guidelines on how to proceed through it and what to look for. It does not, however, mean that inspection participants do not scrutinize the inspected product systematically. The word "ad-hoc" refers to the fact that no technique is used to support the reviewers on the problem of how to detect defects in a software artifact – defect detection fully depends on the skill, the knowledge, and the experience of the reviewer. Training sessions may help subjects develop some of these capabilities to alleviate the lack of reading support.

Checklists offer stronger support in the form of questions. Although reading support in the form of a list of questions is better than none, Laitenberger [23] considers several weaknesses on checklist-based reading, as follows: the questions are often general and not sufficiently tailored to a particular development environment; concrete instructions on how to use a checklist are often missing, that is, it is often unclear when and based on what information an inspector is to answer a particular checklist question; and questions are often limited to the detection of defects that are based on past defect information.

For our purposes, we decided that having an instrument similar to Cockburn's checklist [4] in the form of question-based checklist to evaluate the quality of use cases is a reasonable first step to bring reading techniques to BDD scenarios.

Field	Question
Use case title.	1 Is the name an active-verb goal phrase, the goal of the primary actor?
	2 Can the system deliver that goal?
Scope and Level:	3 Are the scope and level fields filled in?
Scope.	4 Does the use case treat the system mentioned in the Scope as a black box? (The answer may be 'No' if the use case is a white-box business use case, 'Yes' if it is a system requirements document).
	5 If the Scope is the actual system being designed, do the designers have to design everything in the Scope, and nothing outside it?
Level.	6 Does the use case content match the goal level stated in Level?
	7 Is the goal really at the level mentioned?
Primary actor.	8 Does the named primary actor have behavior?
	9 Does it have a goal against the SuD that is a service promise of theSuD?
Preconditions.	10 Are they mandatory, and can they be put in place by the SuD?
	11 Is it true that they are never checked in the use case?
Stakeholders and interests.	12 Are they mentioned? (Usage varies by formality and tolerance) Must the system satisfy their interests as stated?
Minimal guarantees.	13 If present, are all the stakeholders' interests protected?
Success guarantees.	14 Are all stakeholders interests satisfied?
Main success scenario.	15 Does it run from trigger to delivery of the success guarantee?
	16 Is the sequence of steps right (does it permit the right variation in sequence)?
	17 Does it have 3 - 9 steps?
Each step in any scenario.	18 Is it phrased as an goal that succeeds?
	19 Does the process move distinctly forward after successful completion of the step?
	20 Is it clear which actor is operating the goal (who is "kicking the ball?)
	21 Is the intent of the actor clear?
	22 Is the goal level of the step lower than the goal level of the overall use case? Is it, preferably, just a bit below the use case goal level?
	23 Are you sure the step does not describe the user interface design of the system?
	24 Is it clear what information is being passed?
	25 Does the step "validate", as opposed to "checking" a condition?
Extension condition.	26 Can and must the system detect it, and handle it?
	27 Is it phrased as what the system actually detects?
Technology or Data Variation List.	28 Are you sure this is not an ordinary behavioral extension to the main success scenario?
Overall use case content.	29 To the sponsors and users: "Is this what you want?"
	30 To the sponsors and users: "Will you be able to tell, upon delivery, whether you got this?"
	31 To the developers: "Can you implement this?"

Figure 2.2 – Cockburn's use case questionnaire. Source: [4]

2.2 Agile Requirements

Agile software development methods take a different approach to RE and communication than the traditional RE approaches that are mostly based on formal documents and defined phases

[16]. Through the mapping of a systematic literature reviews on the subject, Heikkila and colleagues [16] have pointed out some benefits and challenges for RE on agile contexts.

Two important challenges to note in the Heikkila and colleagues study [16] are the insufficiency of the user story format and the reliance on tacit requirements knowledge. The first challenge comes from the fact that user stories do not convey enough information for software design and separate systems and subsystem are required to fill that hole. The second challenge is due to the fact that requirements knowledge is mostly tacit. Section 2.2.1 will describe user stories to help on the understanding of those problems, while Section 2.2.2 will describe acceptance tests, a way to express requirements details that do not fit the user story model.

The need to better describe those tests and how they bound together with user stories is reinforced by Bjarnason et. al [3], who described how tests are used as requirements on agile contexts. One of the companies on their study uses a BDD approach, where requirements are documented upfront as automated acceptance test cases as part of the elicitation process and are expressed in a structured format in a way that the specification can be executable. This approach is explained in details in Section 2.2.3.

2.2.1 User Stories

Traditional RE activities have become too abstract and moved away from how people ordinarily learn and communicate - too far away from storytelling, something that everyone understands intuitively [32]. By using storytelling, the process of gathering information and structuring the requirements document would be immediately improved, as the author understands that one instinctively transform abstract knowledge into a logical structure when telling a story.

Agile methodologies are sympathetic with this thought and represent requirements using user stories. This form of requirements representation have been created along with the Extreme Programming (XP) methodology, where each story describes one thing that the system needs to do as described by Jeffries et. al [20]. They are written by a customer (or a customer team) to a development team, who have the responsibility to understand the story and design, build and test a software that implement it. User Stories bring together the rights of customers, who have the right to get the most possible value out of every programming moment, by asking for small atomic bits of functionality, and developers, who have the right to know what is needed. They are a short description of the behavior of the system from the point of view of the user of the system and are backed up with conversation and perhaps with some related detailed information.

Furthermore, a user story represents a feature a customer wants in the software, a story she would like to be able to tell her friends about this great system she is using [22]. A user story is a specific thing that would make the system easy to use, a chunk of functionality that is of value to the customer. As a unit of functionality, the team demonstrates progress by delivering tested, integrated code that implements a story. The customer must write the story in cards, and it is up to the developers to estimate it trough collaboration with the customer.

For Cohn [5], a user story describes functionality that will be valuable to either a user or purchaser of a system or software. Each story must be written in the language of the business, not in technical jargon, so that the customer team can prioritize the stories for inclusion into iterations and releases. The author provides some reasons on why user stories are used, as follows: they emphasize verbal rather than written communication; are comprehensible by both the customer and the developers; and encourage deferring detail until the team, customers and developers, have the best understanding they are going to have about what they really need. While Jeffries et. al [20] list only two aspects of user stories – the short description of it and the conversations around it, Cohn [5] composes a User Story using 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; and
- tests that convey and document details and that can be used to determine when a story is complete.

Those aspects have come from the Card, Conversation and Confirmation terms defined by Jeffries et. al [19]. Cohn described that the Card represents 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 and the best examples should be executable. They are representations of the Confirmation, a way to customers tell developers how they will confirm that they have done what is needed in the form of acceptance tests. That confirmation, provided by those examples, 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.

The main format of a user story, popularized by Cohn's book [5], is: I as a *role* want *function* so that *business value*. Lucassen et. al [24] summarize that 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. As user stories express *what* is desired and *why* it is needed by the client, this requirement format is better compared to business or stakeholders requirements [17][18].

2.2.2 Acceptance Tests

Acceptance testing is the process of verifying that user stories were developed such that each works exactly the way the customer team, the ones who write user stories on XP methodology, expected it to work [5]. Acceptance tests are specified as soon as the iteration begins and, depending on the customer team's technical expertise, can be put into an automated testing tool. Those tests help the customer team to communicate assumptions and expectations to the developers, by expressing the details that result from the conversations between customers and developers, and also

by validating that a story has been developed with the functionality the team members had in mind when they wrote the story. New tests should be added as long as they add value and clarification to the story. Their scope should focus on clarifying the intent of the story to the developers instead of covering all the low level possible validations, like testing invalid dates range.

Jeffries et. al [20] said that acceptance tests allow the customer to know when the system works, and tell the programmers what needs to be done. In the XP methodology, programmers have the right to know what is needed and customers have the right to see progress in a running system, proven to work by specified automated test. These tests provide confidence that the system really does what it needs to do.

For Beck and Fowler [22], acceptance tests execution will determine whether the user stories have been successfully implemented. Also, once acceptance tests are running, the story card can be discarded, since they are encoded in far more detail and accuracy in the acceptance tests, so no information will be lost if the cards are destroyed.

Gartner [11] reported that the hardest job in software development is communicating clearly about what one wants the system to do. Guiding the development effort with acceptance tests helps with this challenge. Two core practices are described in his book to address such a challenge, 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 and precise description of "done" for each feature. Teams that follow those practices are working on the Acceptance Test-Driven Development (ATDD) way.

Haugset and Stalhane [14] describe ATDD as a mixture of the traditional documentation-centric RE and the communication-focused iterative agile RE, carrying its own set of benefits and challenges. The benefits stem both from the inherent strengths that lie in discussing requirements using the acceptance tests as a mediator and its abilities for automation of the same tests. On the case study described by the authors, the ATDD technique was used to help communicate requirements (externalization) in the specification phase, helping to uncover, understand and describe requirements. In the study's verification phase, ATDD was used to automatically verify that the code adhered to the customer requirement on an acceptance level, accompanied by manual testing for certain issues. The authors judge that ATDD saves effort in the long run, despite having a higher upfront cost due to the need of even more interaction with customers than agile development does, since the tests act as a safety harness for making changes and acts as documentation of code.

In short, as acceptance tests describe the expectations that a system must show in order to demonstrate that the application is acceptable by the customer [5] and can fill the documentation role of functional requirements [17][18]. Yet, due to the fact that they are often automated test and often written by customers with their own words, they can also fill the living documentation vision proposed by Adzic's Specification by Example book [1].

2.2.3 Behavior-Driven Development

One way to represent acceptance tests are scenarios. Alexander and Maiden [2] describe that the activity of building requirements scenarios encourages imagination and exploration while setting the stage for discovering unconscious and undreamed requirements. That is important because stories are the primary (perhaps only) means of communicating needs and desires, and providing critical feedback to developers.

Kaner [21] describes scenarios as a hypothetical story used to help a person think through a complex problem or system. Scenarios also help in the product learning, as people do not learn well by following checklists or material that is organized for them – people often 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.

Scenarios can also represent the key examples that describe the expected functionality, a concept introduced by Adzic [1]. The living documentation presented by the author helps to validate if the system works in the same way reflected by the documentation that represents it. Example tables are also used on the mentioned book, where one represents the system's inputs and outputs using tables where each row represents a given example. Specialized tools, like the framework for integrated tests shown by Gartner [11], can read those tables and use them on the software under test.

Behavior-Driven Development (BDD) is a set of practices that uses scenarios as an business-readable language to describe and model a system [34]. Scenarios are expressed in a format known as Gherkin, that is designed to be both easily understandable for business stakeholders and easy to automate using dedicated tools. According to the author, bringing business and technical parties together to talk about the same document helps to build the right software (the one that meets customer needs) and to build it right (without buggy code). Those scenarios are referred as structured examples, in a similar way as Adzic [1] calls them key examples. Many ideas are similar on both publications. As explained by Smart [34], using conversation and examples to specify how one expects a system to behave is a core part of BDD.

This idea of using conversation and examples to specify a system behavior is reinforced by Wynne and Hellesoy [38] discourse on their book about Cucumber, a library that implements BDD scenarios for the Ruby programming language. The authors state that software teams work best when the developers and business stakeholders are communicating clearly with one another – and a great way to do that is to collaboratively specify the work that is about to be done using automated acceptance tests. Additionally, when the acceptance tests are written as examples, they stimulate people's imaginations and help them see other scenarios they had not previously considered. Most important is the fact that, when the team members write their acceptance tests collaboratively,

```
Given Western line trains from Emu Plains leave Parramatta
for Town Hall at 7:58, 8:00, 8:02, 8:11
When I want to travel from Parramatta to Town Hall at 8:00
Then I should be told to take the 8:02 train
```

Figure 2.3 – Scenario example. Source [34].

```
Feature: Sign up

  Sign up should be quick and friendly.

  Scenario: Successful sign up

    New users should get a confirmation email and be greeted
    personally by the site once signed in.

    Given I have chosen to sign up
    When I sign up with valid details
    Then I should receive a confirmation email
    And I should see a personalized greeting message

  Scenario: Duplicate email

    Where someone tries to create an account for an email address
    that already exists.

    Given I have chosen to sign up
    But I enter an email address that has already registered
    Then I should be told that the email is already registered
    And I should be offered the option to recover my password
```

Figure 2.4 – Feature file example. Source [38]

they can develop their own ubiquitous language to talk about their problem domain – a common language that helps them avoid misunderstandings.

BDD practitioners collaborate with the user to find concrete examples to illustrate features requested to the development team. Quite often, they also automate the execution of those scenarios. Since they are written to express the details and help the customer accept a given feature, they can be used as acceptance tests. Also, the collaboration among development team and customers that is needed to write those scenarios can be mapped as the conversations about a story that Cohn [5] and Jeffries [19] mentioned.

In Gherkin, the scenarios 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 set of keywords: *Given* describes the preconditions for the scenario and prepares the test environment; *When* describes the action under test; *Then* describes the expected outcomes. One example of a scenario for a train management application that should have a feature to help a commuter to find the optimal itinerary between stations on the same line defined by Smart [34] is presented in Figure 2.3.

Multiple scenarios can be grouped together into feature files, such as the one in Figure 2.4 which groups together two scenarios related to sign-in feature – the first represents a successful login, while the second represents a duplicate e-mail situation. Each of those scenarios has a title

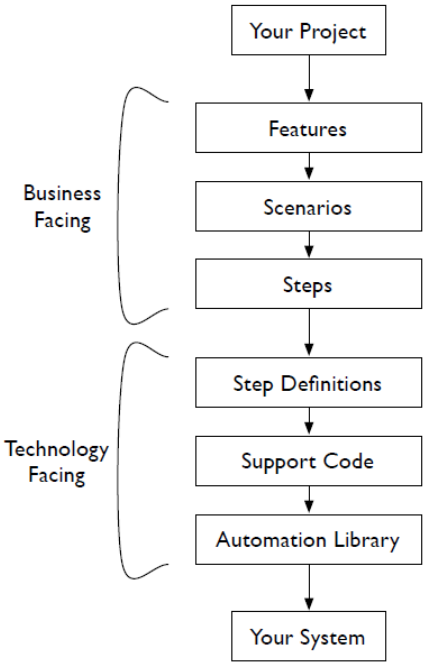

Figure 2.5 – Typical stack of a BDD tool such as Cucumber. Source [38]

and a short description before the steps are described. Also, the beginning of the feature file has a description of the feature being tested by those scenarios.

Along with the features, one should provide to specialized tools, such as Cucumber, a set of step definitions, which map the business-readable language of each step into a code written in a certain programming language to carry out whatever action is being described by the step. This hierarchy, from features down to automation library, is illustrated in Figure 2.5. In this thesis, we are only concerned with the business facing aspects of a BDD scenario, represented by the feature, the scenario and the step boxes in Figure 2.5.

Regarding the process used to specify those scenarios, Smart [35] describes each stage of a feature worked by a team following a BDD process, as shown in Figure 2.6, as a set of cyclical stages, described as follows:

- **Discover stage**: during this early stage, high level techniques, like Story Mapping [29] and Impact Mapping [13], can help get an overall picture of what one is trying to achieve [35]. Additionally, Smart [35] points out that *at this stage, the emphasis is on breadth, rather than detail*;

- **Define stage:** in this stage the team starts to have more concrete conversations around more specific business rules and examples of how the user would interact with the system. Often referred simply as "conversations", in this stage the team should not get bogged down in the Gherkin syntax.

- **Formalize stage**: Smart [35] states that, once key business rules and examples have been identified they can be formalized into Gherkin by an engineer in test, often pairing with a Business Analyst or manual tester. Additionally, once they are ready, the Gherkin scenarios

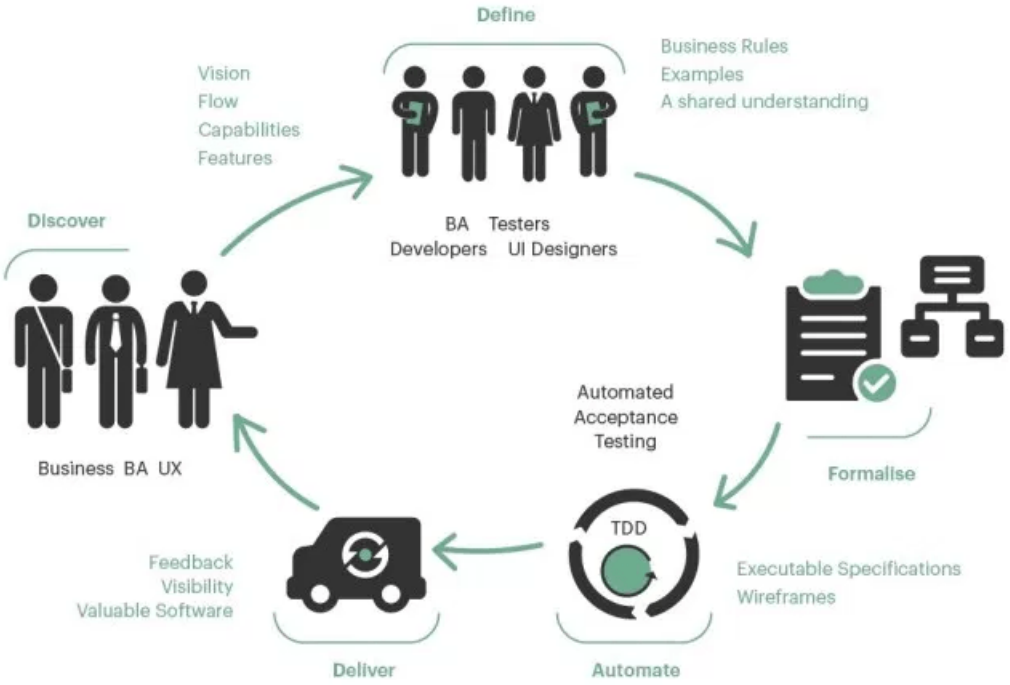

Figure 2.6 – Typical stages of a feature on a BDD process. Source [35]

should be shared and discussed with the broader team, highlighting the need to acquire team's feedback on scenarios, a common behavior we saw on many participants. This is the stage where our research is focused on, as we seek to provide a quality evaluation instrument to aid in the assessment of scenario's quality;

Automate and Deliver stages: this is the stage where the team will create the missing piece of functionality and these Gherkin scenarios will be automated. It is outside of this thesis scope to evaluate the source code that transforms each Gherkin step into an executable script.

BDD scenarios are similar to use cases scenarios as they both describe a system behavior under certain precondition (expressed on the *Given* clauses of a BDD scenario) to achieve a certain goal (expressed on the *Then* clauses of a BDD scenario). As they are a format to express acceptance tests, described in Section 2.2.2, they can also fill the documentation role of functional requirements [17][18]. Finally, due to the fact that they are often automated test and are written on a ubiquitous language that should be understood by everyone in the project, they can also fill the living documentation vision proposed by Adzic [1].

2.2.4 Agile Requirements Quality

Lucassen et. al [24] report that the number of methods to assess and improve user story quality is limited. Existing approaches to user story quality employ highly qualitative metrics, such as the heuristics of the INVEST (Independent-Negotiable-Valuable-Estimable-Scalable-Testable) framework described by Cohn [5]. Due to that fact, Lucassen et. al [24] define additional criteria to evaluate user stories on their QUS Framework, as follows: atomic, minimal, well-formed, conflict-free, conceptually sound, problem-oriented, unambiguous, complete, explicit dependencies, full sentence, independent, scalable, uniform, and unique.

For acceptance tests, our understanding is that this kind of tests should meet the quality attributes that all requirements from both BABOK versions [17][18] meet, as we compare acceptance tests with functional requirements, and the ones for use cases found by Cohn [4] and Phalp et. al [30]. However, little work has been found to support this understanding.

Generic scenarios quality evaluation seems to be based upon subjective characteristics, like the ones found by Kaner [21], or on generic guidelines, like what Alexander and Maiden [2] had highlighted. Kaner [21] 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. Nevertheless, empirical evaluation of existing acceptance test cases expressed as scenarios according to his characteristics were not found. Alexander and Maiden [2] define a guideline that should be used for use cases and scenarios alike - their view seems to indicate that acceptance tests fill the same role on requirements documentation.

To the best of our knowledge, BDD scenarios are only being evaluated based on characteristics, such as those taken from Smart [34] and Wynne and Hellesoy [38] experiences: scenarios steps expressiveness; focus the steps on what goal the user want to accomplish and not on implementation details or on screen interactions (writing it in a declarative way and not on a imperative way); 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. Even though the authors [34][38] specify a few scenarios as examples to demonstrate those described characteristics, we feel that BDD scenarios could benefit from a question-based checklist defined from the collective knowledge of other practitioners, similar to the one defined by Cockburn for use cases [4].

2.3 Conclusion

The quality characteristics shown on both versions of the BABOK [17][18] have taught us that traditional requirements writers have many guides to help them evaluate their work like the ones described by Laitenberger [23] and Melo et. al [25]. Also, Phalp et. al [30] have shown that use cases quality criteria and guidelines are also mature enough to help practitioners writing them.

Finally, user stories writers have informal guidelines (such as the INVEST acronym used by Cohn [5]) and an ongoing study on quality characteristics (from Lucassen et. al [24]) to help on their efforts. On the other hand, the quality of acceptance tests is still open to debate, with only a few characteristics (found by Kaner [21], Smart [34] and Wynne and Hellesoy [38]) or generic guidelines (like the one created by Alexander and Maiden [2]) to use, all without empirical evaluation to certify their usefulness. No reading technique were found to address acceptance tests as well.

On agile contexts, requirements are represented mainly as user stories according to Lucassen et. al [24]. Due to that fact, it makes sense that the authors focus their quality framework on the user story card representation from Cohn [5] only, letting the details that are expressed as acceptance tests out of it. However, due to the growing importance of requirements expressed as tests shown by Bjarnason et. al [3], we believe this gap needs to be filled.

BDD is a format to represent acceptance tests, as stated by Gartner [11], that was used on one of the companies interviewed by Bjarnason et. al [3] study and to which only informal characteristics exists to evaluate them, either from Smart's experiences [34] or Wynne and Hellesoy's [38]. Since BDD scenarios represent the intended behavior of a system, the quality attributes for use cases and functional requirements could be re-used – but no study was found on that area. Also, the framework from Lucassen et. al [24] could be expanded to also cover BDD scenarios – but again, no efforts on that direction have been found. Finally, having a set of attributes to evaluate a written work without a strategy or a way to use them properly may not be enough to help a writer be more effective. Thus, there is also the need to provide a way to guide the software development team members on how to proper use those attributes in a review process.

3. RESEARCH METHODOLOGY

The main goal of our research is two-folded: to identify a set of criteria suited to evaluate BDD scenarios based on the knowledge of practitioners and to propose a question-based checklist to evaluate BDD scenarios. The output of those questions are meant to help the evaluator decide how good her BDD scenarios are and, if not satisfactory, may be the input of formal or informal scenarios' refinement sessions and other formats of conversations around those scenarios.

Research Question 1 (RQ1):

What are the criteria suited to evaluate a good BDD scenario by the view of a software development team member?

Research Question 2 (RQ2):

How does a software development team member evaluates BDD scenarios with those attributes?

3.1 Research Objectives

This research have the following minor objectives:

- **Objective 1:** To summarize the evaluation characteristics that the written form of BDD scenarios may have, according to the experience of its practitioners;
- **Objective 2:** To summarize the quality attributes applicable to BDD scenarios format;
- **Objective 3:** To link these evaluation characteristics with the quality attributes applicable to BDD scenarios format;
- **Objective 4:** To propose a question-based-checklist, that would guide the reader during the quality evaluation of BDD scenarios.

3.2 Research Design

Since there is no standard on what is considered a "good" scenario due to the few views on this matter, this is a concept that still needs to be fully understood. A qualitative approach is suitable for situations like this [6]. Also, we believe the opinions from industry practitioners working on multiple companies and contexts would be a natural input to construct a quality concept for BDD scenarios. Therefore, we judge that the use of an empirical approach would suit our research goal and questions. Finally, we decided to collect data using semi-structured interviews, as we value multiple different opinions taken from different contexts.

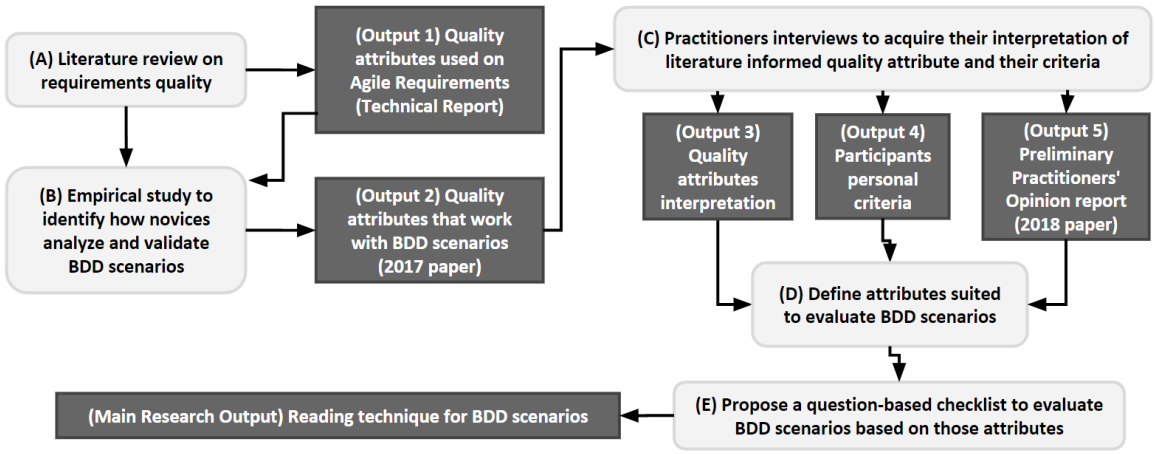

Figure 3.1 – Research Design

To achieve our research goal, we proposed a multiple-steps research design as presented in Figure 3.1, where the light gray boxes are the studies/actions that achieved the outputs shown in the dark gray boxes. This design was based upon the understanding that interviews with open-ended questions alone would not provide enough data to formulate a proper question-based checklist due to the participants' plurality of terms and opinions. Therefore, we judged necessary to have our interviews guided by a set of quality attributes that had previously been used to evaluate BDD scenarios.

First, we had to discover what are those literature quality criteria. A literature review (Figure 3.1, Study A) has confirmed that traditional attributes [17][18] and the INVEST[5] acronym were used with agile requirements. That literature review was reported as a technical report [28].

Additionally, we acquired some knowledge about how evaluators judge the quality of BDD scenarios using those attributes. To that end, we conducted a pilot study (Figure 3.1, Study B) with 15 graduate students to understand how they evaluate BDD scenarios. The output of this pilot study was the sub-set of literature attributes that were used to guide the debate with practitioners – shown as Output 2 in Figure 3.1 and reported in the 2017 Workshop on Empirical Requirements Engineering (EmpiRE'17) in conjunction with the International Requirements Engineering Conference [26].

In addition to those literature quality attributes, we used real-life examples of BDD scenarios during our interviews to aid practitioners realize their own quality criteria. To avoid our own bias towards what would be a good or bad BDD scenario, we decided to not create the examples ourselves. Instead, we handed them real BDD scenarios, taken from an open source project that employ BDD scenarios to detail their applications' behavior.

To find those real BDD scenarios, we searched Github's[1] repository database. Due to the fact that BDD scenarios are automated, they are often carried along with the application source code as the tests to that system. Due to the popularization of tools like Cucumber [38], BDD's feature files are saved with the extension ".gherkin". Github's search feature allowed us to filter repositories by the types of files they used, an advanced functionality missing in other known repositories of

[1] https://github.com/

source code. Therefore, we filtered our search for the gherkin extension. That search resulted in Diaspora[2], a decentralized social network with a list of feature files mapping the behavior of the different application screens. To the best of our knowledge, Diaspora is Github's open source project with the most feature files available.

Besides the literature-informed quality attributes and the real-world example of BDD scenarios, we sometimes provoked participant's thoughts using a list of experience-based criteria taken from Smart's book [34] and Wynne and Hellesoy's book [38], such as: steps too long/too short; too few/many steps; business language usage; title description; keywords (Given/When/Then) usage and order; repetition of steps. That list was kept hidden from the participant's – certain items were used whenever we judged necessary, as additional questions to refine their analysis of Diaspora's scenarios.

Our semi-structured interviews (Figure 3.1, Study C) produced practitioners' interpretations of literature-informed quality attributes (Output 3, reported in Section 4.2) and their own personal evaluation criteria (Output 4, reported in Section 4.3). With that data at hand, we were able to define quality attributes to suited to evaluate BDD scenarios (Action D, reported in Section 4.4) and answer our RQ1. Composing those newly-defined attributes into a question-based checklist (Action E, reported in Chapter 5), similar to the questionnaire Cockburn [4] uses for Use Cases, we reached the answer to our RQ2. A preliminary report of our semi-structured interview (with 8 practitioners) results is reported in the 2018 International Working Conference on Requirements Engineering (REFSQ'18) paper [27].

Action D's initial purpose was to refine the sub-set of literature-informed quality attributes. However, as we could not decide for ourselves what interpretation of each attribute is more suited than others without enforcing our own bias, we used them all as characteristics, regardless of what literature attribute generated it, and composes them into newly-labeled quality attributes.

3.2.1 Pilot Study

Our pilot study (Figure 3.1, Study B) aimed to provide us some knowledge about how evaluators judge the quality of BDD scenarios using quality attributes found on the literature review (Figure 3.1, Study A). Therefore, we organized a study with 15 graduate students, all with previous industry experience as developers, technical leads, or managers. They also have declared to know well how to write use cases, and to a lesser extent how to write user stories or BDD scenarios. The literature quality attributes used were: atomic, complete, consistent, concise, estimable, feasible, independent, negotiable, prioritized, small, testable, understandable, unambiguous, and valuable.

During the study, the students were first asked to write requirements for two fictional products (PA and PB) in two different formats – use cases (UC) and user stories with BDD scenarios (US+BDD). Later, each student evaluated the work of 2 others colleagues using the literature quality attributes we have listed before. This cross-activity design exemplified in Table 3.1 helped us reduce

[2]https://diasporafoundation.org/

Table 3.1 – Cross-activity design of pilot study. Source: [26]

ID	Task 1	Task 2	Task 3	Task 4
S1	Write UC for PA	Write US+BDD for PB	Evaluate US+BDD for PA *(S2 task 1)*	Evaluate UC for PB *(S4 task 2)*
S2	Write US+BDD for PA	Write UC for PB	Evaluate UC for PA *(S1 task 1)*	Evaluate US+BDD for PB *(S3 task 2)*
S3	Write UC for PA	Write US+BDD for PB	Evaluate US+BDD for PA *(S4 task 1)*	Evaluate US+BDD for PB *(S2 task 2)*
S4	Write US+BDD for PA	Write UC for PB	Evaluate UC for PA *(S3 task 1)*	Evaluate US+BDD for PB *(S1 task 2)*

the risk of the evaluator lacking understanding of the application domain under analyses and of the evaluator's reading ability being driven by the same writer's bias.

From this pilot study, we have acquired the literature quality attributes that could potentially work with BDD scenarios, according to the opinion of the novice BDD evaluators, students with industry experience. Therefore, the literature-informed quality attributes used later in the semi-structured interviews are: concise, estimable, feasible, negotiable, prioritized, small, testable, understandable, unambiguous, and valuable.

3.2.2 Interview's Participants Selection

In order to identify the first participants for our interviews, we organized a survey to both acquire a grasp of people's opinion about BDD quality topic and their contact information. The survey, titled as *Do you believe BDD scenarios' quality matter?*, posed two statements and an entry to collect a respondent's contact information for later follow up. Answers to the statements ranged from 1 (Totally Disagree) to 5 (Totally Agree). The statements were:

1. Well written BDD scenarios directly impacts the final product quality.
2. It is important to have a standard set of criteria to evaluate written BDD scenarios' quality.

The survey was shared on the social network (such as Twitter[3] and Linkedin[4]) profiles of the thesis author and on BDD virtual communities such as Cucumber slack channel[5], BDD

[3]https://www.twitter.com
[4]https://www.linkedin.com
[5]https://cucumberbdd.slack.com/messages/C590XDQQH/

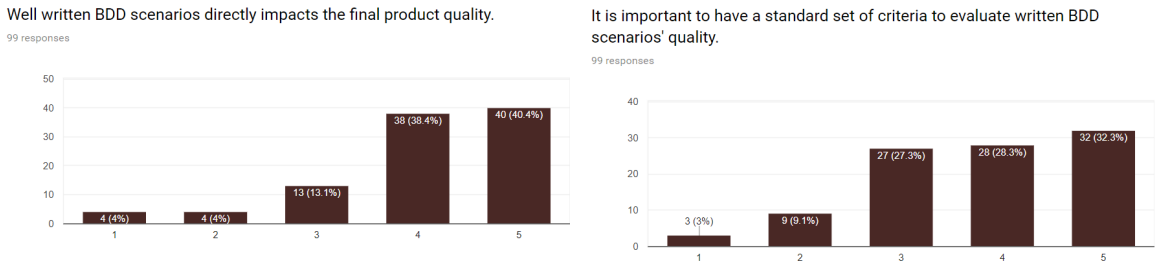

Figure 3.2 – Survey Answers' Statistics

London slack Channel[6], Ministry of Testing forum[7] and slack channel[8]. We got 99 answers to the survey, detailed in Figure 3.2, with the large majority of the respondents agreeing with the previously presented statements. However, contrary to our beliefs, the follow up e-mails inviting people to a larger interview session were not answered accordingly.

Therefore, we changed our focus to direct searches on this thesis' author Linkedin social-network. Our belief was that a more personal touch, showing the author's profile (someone who works in industry with test automation and has experience with BDD for over 5 years) to potential participants, would improve our chances. The search feature on the website[9] was heavily used, returning data from the inside of the thesis' author network but also from the outside – up to 3 connections of distance. Using the terms "BDD" and "Behavior-Drive-Development", every search returned experienced BDD practitioners. Within every practitioner profile, we double-checked the existence of BDD experience and proceeded to invite the person to connect and dedicate their time in an 1hr slot interview. By the time of the writing of this thesis, 79 invitations were accepted and 34 invitations were not answered. From those who accepted to connect, only 18 accepted to participate in the interviews. All those 18 participants' profiles are reported in Section 4.1.

3.2.3 Interview Design

Taking inspiration on the questions used by Heck and Zaidman [15] on their interviews with practitioners to understand how user stories' quality can be understood, we list in Table 3.2 the set of questions to capture practitioners views and opinions on our interviews, to disclose how they use BDD scenarios to perform their tasks during a software development cycle, and to identify how they evaluate if those scenarios are ready to be used.

The authors' original interview script [15] was organized into two parts – one performed with minimal introduction from their side and without showing the participants their framework and another in which they asked the participants to use their framework, transformed into a checklist, on some examples taken from open source projects.

[6]https://bddlondon.slack.com/messages/C1BMRM2HX/details/
[7]https://club.ministryoftesting.com/t/do-you-believe-bdd-scenarios-quality-matter/1449
[8]https://ministryoftesting.slack.com/messages/C0RUQE6V8/
[9]https://www.linkedin.com/help/linkedin/answer/302/searching-on-linkedin

Table 3.2 – Interview Questions

ID	Question
1	What is your role on the project?
2	What is your main task on the project?
3	For how long do you use BDD?
4	How does your project use BDD scenarios?
5	What do you pay attention to when evaluating BDD scenarios?
6	On Diaspora, evaluate a feature file according to your criteria.
7	Do quality attributes help evaluating BDD scenarios?
8	What is the meaning of each attribute on BDD scenarios?
9	On Diaspora, evaluate a feature file according to the attributes.
10	Do you miss any other attribute?
11	What quality attributes did you find difficult to use?
12	To what extent the attributes helped you evaluate a scenario?
13	How your criteria maps to those attributes?

In a similar way, the open ended questions in our interview script in Table 3.2 were crafted to acquire the participants views before presenting their own and help us answer our **RQ1**, while the act of actively using literature-informed quality attributes to evaluate Diaspora's examples aid on our **RQ2**. The four starting questions were used to understand the participants' profile and their answers are presented in Section 4.1.

Questions 6 and 9 asked the participant to actively use examples of BDD scenarios taken from Diaspora project and evaluate them, either with their own personal criteria or the literature-informed quality attributes. From the 47 feature files available to cover Diaspora's desktop version, the participants were asked to choose one to evaluate as part of question 6 and another, different one, to evaluate as part of question 9. The interviewer did not influenced their decision in any way.

Questions 5 and 6 were useful to acquire participant's criteria without any bias of our list of literature-based quality attributes and their answers are reported in Section 4.3. If only a few answers emerged for those questions, we provoked their thoughts using the experience-based criteria from Smart [34] and Wynne and Hellesoy [38] books, as explained in the Chapter 2.2.4.

Questions 7 to 11 were useful to acquire participant's interpretations of our literature-informed quality attributes and are analyzed in Section 4.2.

Taking the RQs lens, Questions 5 and 8 helped reveal the meaning practitioners see on quality attributes and on their own criteria, thus allowing the newly defined quality attributes that answer our RQ1 to emerge. In addition, Questions 6 and 9 gave us insights on how those attributes and personal criteria were used in practice, providing important information to answer RQ2 and define our question-based checklist.

Question 12 was useful to assess how useful such a list was and tease the interviewee to suggest better ways to evaluate BDD scenarios. Question 13 asks the participant to link those literature quality attributes with their own criteria, tying those attributes with practical details of BDD scenarios and motivating us to threat interpretations and criteria altogether in our checklist.

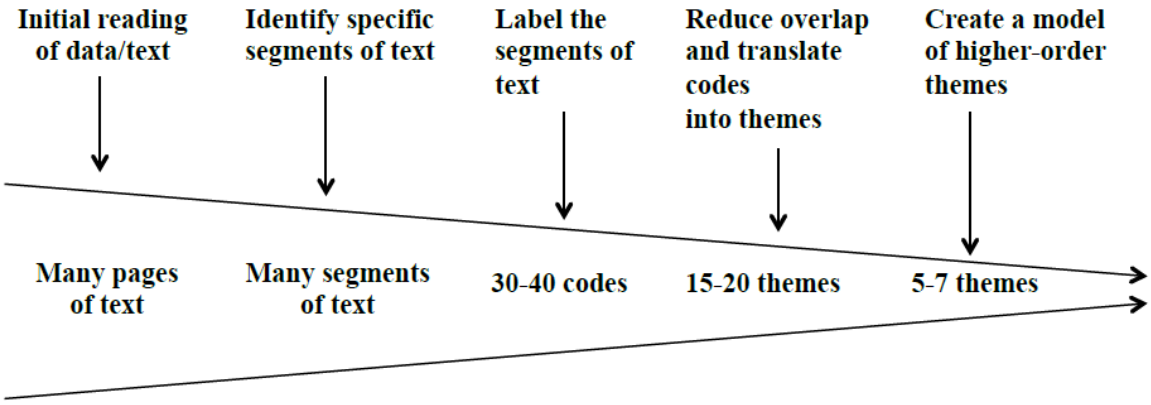

Figure 3.3 – Thematic synthesis process. Source [7]

That question was a final check in case the conversation have not linked participant's criteria and interpretations to attributes. Our initial thought was that this mapping would be useful to join good and bad practices, that emerged from practitioner's personal criteria, with literature-based quality attributes. However, this intention was not successful, due to the different interpretations of each attribute. As explained before, we decided to use all interpretation of each literature-based attribute as characteristics to avoid choosing which of those interpretations was best suited for each literature quality attribute. Additionally, the participants' personal evaluation criteria were also treated as characteristics. Those characteristics were later grouped together into newly-redefined quality attributes, that compose our proposed question-based checklist as presented in the following sections.

Before starting those interviews, the script was validated by a doctoral student with 4 years of previous experience in industry. Additionally, a pilot interview was conducted with a software tester who has about 6 years of experience with BDD usage. He reinforced our idea of using Smart's [34] and Wynne and Hellesoy's [38] experience-based criteria to provoke the discussions in Questions 5 and 6 from Table 3.2.

3.2.4 Data Analysis

In total, we interviewed 18 practitioners. Each interview lasted in average 77 minutes, being the longest 1 hour and 51 minutes and the shortest 62 minutes long – a total of 1390 minutes were recorded. The interviews were conducted via Skype[10] tool and were voice-recorded (with documented permission). The interviews were performed either in English or Portuguese during a period of 3 months (from September to November in 2017) and then manually transcribed to the participant's native language. The quotes were translated to English whenever a certain reference to it was needed. We realized we had reached saturation when no new practitioner quality criteria was found. That is when we decided to stop the interview process and start the findings compilation.

[10]https://www.skype.com

Cruzes and Dyba state [7] that a number of different methods have been proposed for the synthesis of qualitative and mixed-methods findings such as the ones we typically find in software engineering (SE), which lead them to conceptualize thematic synthesis in SE. Thematic synthesis, whose process is presented in Figure 3.3, draws on the principles of thematic analysis and other established methods in primary qualitative research to identify the recurring themes or issues in the primary sources of data, analyzes these themes, and draws conclusions from them.

Coding is a method that enables the researcher to organize and group similar data into categories because they share some characteristic – the beginning of themes [7]. All our text was coded in English, even tough our interviews' data were both in English and Portuguese – a process made possible by the fact that both the main researcher, who interviewed the participants and coded the data, and the advisor, who reviewed the codes, reads and speaks both languages. Coding can be performed using one of the following approaches:

1. **Deductive approach:** starting with a start list of preconceived codes, data is analyzed and coded accordingly. Cruzes and Dyba [7] state that great care must be taken to avoid forcing data into these categories because a code exists for them;

2. **Inductive approach:** Cruzes and Dyba [7] state that the inductive approach reviews data line by line in detail and, as a concept becomes apparent, a code is assigned. Upon further review of data, the analyst continues to assign codes that reflect the emerging concepts, highlighting and coding lines, paragraphs, or segments that illustrate the chosen concept;

3. **Integrated approach:** the integrated approach is meant to mix both the deductive and the inductive approach, creating a general accounting scheme for codes that is not content specific, but points to general domains in which codes can be developed inductively [7].

Knowing that we specifically decided to guide our interviews with literature-informed quality attributes (Questions 7 to 11 from Table 3.2) and with author's experience-based criteria [34][38] (Questions 5 and 6 from Table 3.2), we used those as initial thematic analysis codes only, taking a integrated approach and assuming they would be refined along with the interviews responses.

Themes are the outcome of coding, categorization, and analytic reflection, but also a way of grouping the initial codes into a smaller number of sets [7]. We refined the initial thematic analysis codes into themes during our cyclical analysis, resulting in the newly-redefined quality attributes (Action D from Figure 3.1, reported in Section 4.4) that composes our proposed question-based checklist (Action E from Figure 3.1, reported in Chapter 5)

The connection between these different levels of abstractions, from codes (literature-informed quality attributes and author's experience-based criteria) to themes (the newly-defined quality attributes) until a model is formed (the proposed question-based checklist), can be seen in Figure 3.4. Codes were grouped together in a mind map, constructed in the XMind[11] software. The mapping between our codes and themes is found in Table 4.4, where the characteristics (codes) are mapped to themes (newly-defined attributes).

[11] www.xmind.net/

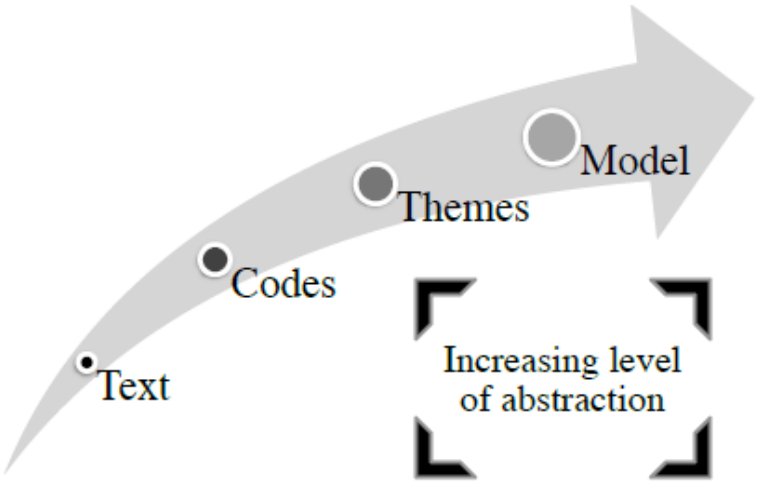

Figure 3.4 – Levels of interpretation in thematic synthesis. Source [7]

3.2.5 Threats to Validity

The fact that only the author of this thesis was involved into the coding process may have impacted the themes creation in an unforeseen way, even with the careful review of the advisor.

Additionally, we have not taken into consideration the gender, role, location nor the type of industry a participant works into to reflect upon the data taken from the interviews. We understand that different life experiences may have brought different quality criteria and opinions, so we tried to mitigate this effect interviewing people from many different areas and companies.

The choice of using BDD scenarios from the Diaspora open source real-life project might also be a threat to the interview results. Choosing different sets of BDD scenarios might have yield different results than ours. We discussed this matter during the interview validation and pilot interview, and both professionals judged Diaspora's feature files as good representatives of real world BDD scenarios.

Finally, we could have phrased the items in our question-based checklist in many different ways, so we have to acknowledge that our own language bias may have driven us to write them in that way presented in Chapter 5. A specialist review of that question-based checklist is listed as a future work to avoid this threat.

4. INTERVIEW RESULTS

This chapter explore the participants responses to the questions in Table 3.2 by presenting participants' interpretations of the literature-informed quality attributes, taken as result of our pilot study [27], and their personal evaluation criteria, tied to BDD scenario's characteristics and similar to the author's experience-based criteria [34][38].

The multiple interpretations of the literature-informed quality attributes had forced us to question if words such as "concise" and "understandable" would be suited to represent the characteristics of good BDD scenarios. Therefore, we decided to deal with each interpretation of each attribute in separation and threat them all as characteristics. Additionally, the participant's personal evaluation criteria were also treated as characteristics. In Section 4.4 we group those characteristics into 8 newly-labeled attributes, that we believe are more suited to evaluate BDD scenario's quality than literature quality attributes. Those newly-labeled attributes answer our RQ1 and provide us the insights needed to answer the RQ2 in Chapter 5.

4.1 Participants' Profiles

The summary of the 18 participants' profiles can be found in Table 4.1. All of them reported being involved with BDD scenarios by writing or reviewing them. The majority of the participants were marked with the "Test" role on the table due to their testing background - they

Table 4.1 – Participants' profiles

Participant	Role	BDD experience	Location
P1	Tester	3 years	England
P2	Tester	3 years	Netherlands
P3	Developer	3 years	Netherlands
P4	Coach	3 months	Denmark
P5	Tester	9 months	Netherlands
P6	Tester	2 months	Netherlands
P7	Tester	3 years	Netherlands
P8	Dev/Coach	10 years	Hungary
P9	Coach	3 years	England
P10	Developer	6 years	Sweden
P11	Tester	4 years	Australia
P12	Tester	3 years	Brazil
P13	Developer	3 years	Brazil
P14	Tester	1 year	Brazil
P15	Coach	5 years	Australia
P16	Developer	1 year	Brazil
P17	Tester	4 years	United States
P18	Tester	4 years	England

Table 4.2 – How participants use BDD scenarios

	Participants
Conversations leads to scenarios	1,4,10,11,15,17,18
Team scenarios are validated by PO	2,3,5,6,7,8,9,11,14
Tester writes test cases into scenarios' format	12
Scenarios received by the Team as upfront requirements	13, 16

worked both on the scenario's Gherkin textual descriptions and on the automation of those steps. The ones marked with the "Developer" role worked as developers who used BDD scenarios as inputs for their development tasks. Finally, the ones marked as "Coaches" were mainly external consultants who help teams get training on BDD - all of them helped the team reviewing their BDD scenarios Gherkin textual descriptions.

Regarding their location, the majority of participants came from the Netherlands (P2, P3, P5, P6, P7). That majority can be explained by the fact that we had used the Linkedin[1] social network to find people to interview and the Netherlands is the home location of one of the participants – therefore, intuitively it would make sense to have that social network suggesting people around one's localization. Other participants came from Brazil (P12, P13, P14, and P16), England (P1, P9, and P18), Australia (P11 and P15), the United States (P17) and from other European countries (P4, P8, and P10).

Another common trait is the way participants use BDD scenarios. The majority of them enhance the textual descriptions on BDD scenarios with conversations, be it a prior conversation with different project roles (such as an example mapping session [37]) or a post conversation to validate if the written scenarios made sense to other team members. However, we have also found cases with BDD scenarios being used without conversations. For one participant (P12), BDD scenarios were used only as inputs to their test automation tool. For other two (P13 and P16), BDD scenarios were requirement's documentation, send to the team by the client as upfront documentation made to represent his needs.

4.2 Quality Attributes' Interpretation

Participants' interpretations of each literature-informed quality attribute that could potentially be used with BDD scenarios [26] help us understand what words would help to evaluate BDD scenarios. To that end, the answers of Questions 8 and 9 from Table 3.2 were analyzed. Additionally, the answers of Question 10 in the same Table 3.2 helped us to evaluate if something was missing in our list, while the answers of Question 7 (Table 3.2) helped us to eliminate some attributes, judged as not useful for the participants.

The results presented in the next sections are based on a partial mind map summarizing the participants' opinions, shown collapsed in Figure 4.1, that will be break down in each next

[1] www.linkedin.com

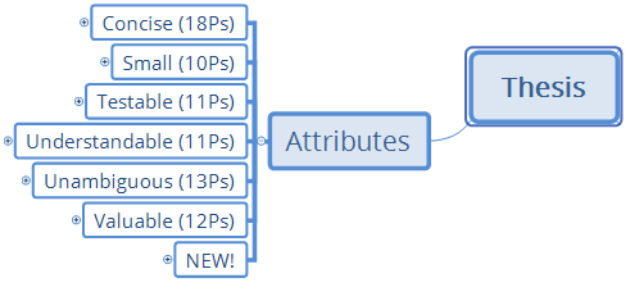

Figure 4.1 – Overview of the attributes to analyze BDD scenarios

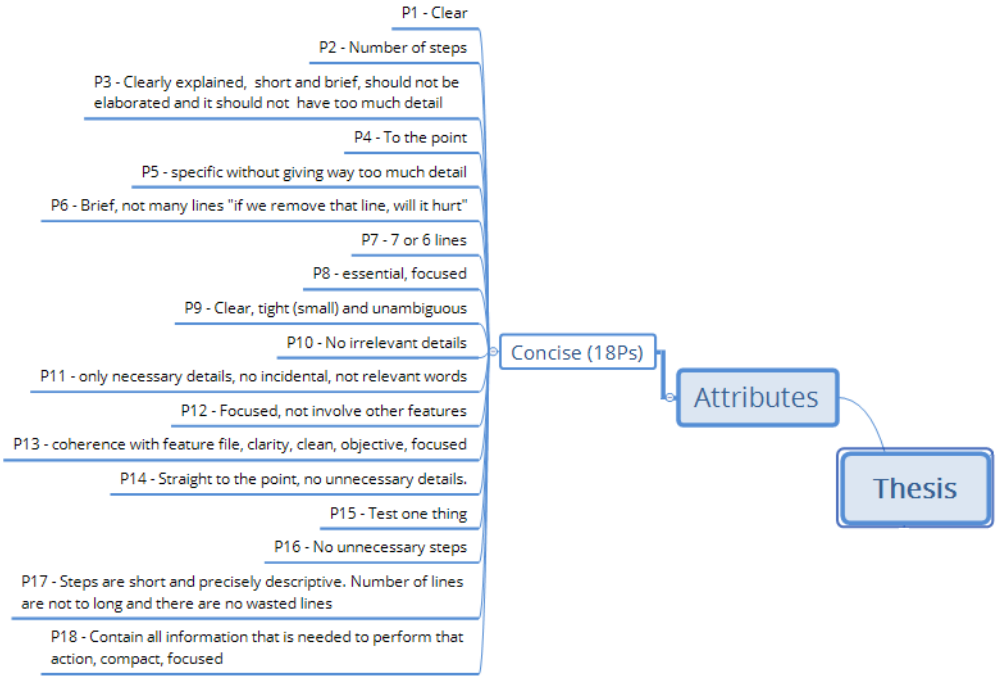

Figure 4.2 – Concise attribute interpretations

section topic. Each attribute node contains the number of participants that were able to interpret that attribute presence into BDD scenarios – for example, all 18 participants were able to interpret concise attribute presence into BDD scenarios, while only 12 were able to interpret the meaning of valuable on that context. Attributes not listed were judged as not useful for BDD scenarios and are discussed into Section 4.2.7. New attributes, together with their corresponding interpretation, can be found under Section 4.2.8.

4.2.1 Concise

Concise attribute was deemed important by all 18 participants – a summary of their interpretation can be found in Figure 4.2. A concise BDD scenario has no unnecessary details (P3, P5, P8, P10, P11, P16, P17), is focused (P4, P7, P10, P12, P13, P14, P18), clear (P1, P3), and has only a few (P2, P7) brief and comprehensive (P3, P6, P9, P11, P18) steps.

```
Scenario: user logs in
  Given a user with username "ohai" and password "secret"
  When I sign in manually as "ohai" with password "secret"
  Then I should be on the stream page
```

```
Given a user
  | username | password |
  | ohai     | secret   |
When I sign in with that user
Then I should be on the stream page
```

Figure 4.3 – Scenario from logs-in-and-out feature (left) and its more concise version (right)

(A) Not much details

P5 reports that a concise scenario *should be specific without giving way too much detail* and should have *no steps longer than they should*. P17 says that *the implication of the word concise is that it's upon the text itself and the length of the scenario, as when you say you have concise steps that means that the steps themselves are short in length and descriptive, and precisely descriptive, right, that are no wasted words and when you talk about a concise scenario that means to me that the number of lines are not to long and again there are no wasted lines, every line has a significant meaning and it says what it needs to say and it's short, sweet to the point fashion*. P9 says it is something clear, tight and unambiguous, while also suggesting to avoid *long step definitions, long steps, things that take a long time to read and understand*. P10 enhances that stating that concise means *a scenario without irrelevant details*, and continues by saying that *some details are important, some should be there, some are important, and if you remove them some things will be scarce anyway. So, if you give me something with lots of details but there's only one or two that is actually important, then you just confuse me* – and even summarizes his opinion with a metaphor, saying that *there are too many trees, I cannot see the wood*. P8 define scenarios without unnecessary things as essential and P11 define it as *crispy* and refine this stating that *you only have the necessary details on your steps, you do not have incidental words, which are not really relevant to that scenario*. As an example, P11 suggested the use of a data table to isolate data used by many steps and avoiding that same data appearing multiple times on the scenario, as can be seen in Figure 4.3.

(B) Not many steps

P2 said that he *does not like overly long scenarios as the longer they are the most likely they contain implementation details*. P7 also had the same opinion, saying that *a scenario should be small* and that a writer should not *use more than 2 Givens, only 1 When, and then 2 Thens, maybe even 1 because you want to check one thing*. As stated before, P17 is also in favor of few steps - as concise would mean no wasted words and no wasted lines.

P4 puts steps size under small word, but suggested that steps repetition is something to hurt conciseness. Therefore, to avoid steps repetition on successive scenarios, P4 proposed the use of Background section, as exemplified in Figure 4.4.

```
Scenario: A user mentions another user at the end of a post
  Given following users exist:
    | username     | email             |
    | Bob Jones    | bob@bob.bob       |
    | Alice Smith  | alice@alice.alice |
  And a user with email "bob@bob.bob" is connected with "alice@alice.alice"
  When I sign in as "alice@alice.alice"
  And I expand the publisher
  And I append "@Bob" to the publisher
  And I click on the first user in the mentions dropdown list
  And I press "Share"
  Then I should see "Bob Jones" within ".stream-element"
  When I follow "Bob Jones"
  Then I should see "Bob Jones"

Scenario: A user tries to mention another user multiple times
  Given following users exist:
    | username     | email             |
    | Bob Jones    | bob@bob.bob       |
    | Alice Smith  | alice@alice.alice |
  And a user with email "bob@bob.bob" is connected with "alice@alice.alice"
  When I sign in as "alice@alice.alice"
  And I expand the publisher
  And I append "@Bob" to the publisher
  Then I should see "Bob Jones" within ".tt-suggestion"
  When I click on the first user in the mentions dropdown list
  When I press the "A" key in the publisher
  And I append "@Bob" to the publisher
  Then I should not see the mentions dropdown list
  When I press "Share"
  Then I should see "Bob Jones" within ".stream-element"
```

```
Background:
  Given following users exist:
    | username     | email             |
    | Bob Jones    | bob@bob.bob       |
    | Alice Smith  | alice@alice.alice |
  And a user with email "bob@bob.bob" is connected with "alice@alice.alice"

Scenario: A user mentions another user at the end of a post
  When I sign in as "alice@alice.alice"
  And I expand the publisher
  And I append "@Bob" to the publisher
  And I click on the first user in the mentions dropdown list
  And I press "Share"
  Then I should see "Bob Jones" within ".stream-element"
  When I follow "Bob Jones"
  Then I should see "Bob Jones"

Scenario: A user tries to mention another user multiple times
  When I sign in as "alice@alice.alice"
  And I expand the publisher
  And I append "@Bob" to the publisher
  Then I should see "Bob Jones" within ".tt-suggestion"
  When I click on the first user in the mentions dropdown list
  When I press the "A" key in the publisher
  And I append "@Bob" to the publisher
  Then I should not see the mentions dropdown list
  When I press "Share"
  Then I should see "Bob Jones" within ".stream-element"
```

Figure 4.4 – Partial mentions feature (left) and a more concise version (right)

```
Scenario: Change my post default to public
  When I press the aspect dropdown
  And I toggle the aspect "Public"
  And I press "Change" within "#post-default-aspects"
  And I go to the stream page
  And I expand the publisher
  Then I should see "Public" within ".aspect-dropdown"
```

```
Scenario: Change my post default to public
  When I  change my post default to public
  Then I should see "Public" within ".aspect-dropdown"
```

Figure 4.5 – Scenario from change settings feature (left) and its more concise version using declarative language (right)

(C) Focused

That check-one-thing opinion enforces P4's opinion that concise is more *to the point*. P12 said it should have a specific focus and should not involve other features. P13 declares that concise means *clarity, clean, objective, focused*. P14 said it should be *straight to the point, without unnecessary details, one that does not stall, like those who are very long and want to explain many things, it does not need to be very long, trying to explain point-by-point, it can be more direct*. P15 described a scenario on change_settings feature as not concise *cause it's testing a whole bunch of different things, it's long, it's boring, I do not care* and suggested the change found in Figure 4.5.

(D) Clear

P3's opinions about the steps was that they should be clearly explained, short and brief, should not be elaborated and it should not have too much detail. The word clear also appeared

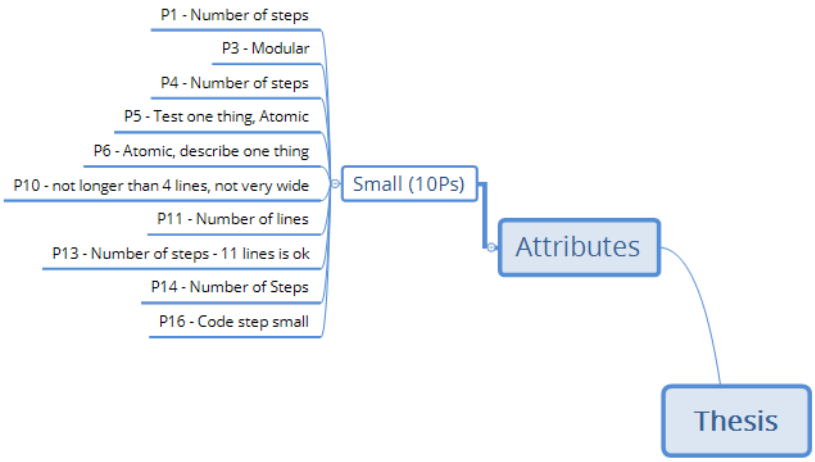

Figure 4.6 – Small attribute interpretations

on the report from P1, who said that *you need to write clear steps and you need to be sure that everybody has the same understanding so it's really clear and there's hard to read it differently*.

(E) Not Unnecessary Lines

P6 also referred to it to be *brief and comprehensive* but went ahead and question himself: *if I remove this line, will it actually change the implementation and the outcome? what can I remove without changing the meaning that would make it understandable?* Those questions are somewhat related with P16 opinion that *each step has value, each step is necessary*.

4.2.2 Small

Small attribute was considered important by only 10 participants – a summary of their interpretation can be found in Figure 4.6. A small scenario, often considered the same as a concise scenario (P2, P7, P8, P9, P15), is one with just a few steps (P1, P3, P4, P10, P11, P13, P14), which test only one thing (P5, P6) and for which the code to turn it into an executable step is small (P16). Additionally, some participants do not believe that scenarios have to be small (P12, P17, P18).

For P2, P7, P8, P9, and P15, a small BDD scenario is a concise BDD scenario, as both have the same meaning.

(A) Not Many Steps

The participants P1, P3, P4, P10, P11, P13, and P14 declared that small refers to the number of steps, a problem previously exemplified in Figure 4.5 and in Figure 4.4. P3 and P10 expand it to also cover the steps length, with P10 saying that *if [the scenario is] longer than 4 lines it's possible not small, if it's wider than the example I looked, some were very very wide, some I would break in some point. If I need a lot of words to express this specific example, it's certainly*

```
6     Scenario: Invalid client id to auth endpoint
7       When I register a new client
8       And I send a post request from that client to the code flow authorization endpoint using a invalid client id
9       And I sign in as "kent@kent.kent"
10      Then I should see a message containing "Invalid client id or redirect uri"
```

Figure 4.7 – Not a small scenario due to the lengthy line 8

```
Scenario: Setting not safe for work
  Given following users exist:
    | username    | email             |
    | pr0n king   | tommy@pr0n.xxx    |
  And I sign in as "tommy@pr0n.xxx"
  When I go to the edit profile page
  And I mark myself as not safe for work
  And I submit the form
  Then I should be on the edit profile page
  And the "profile[nsfw]" checkbox should be checked
  When I go to the edit profile page
  And I mark myself as safe for work
  And I submit the form
  Then I should be on the edit profile page
  And the "profile[nsfw]" checkbox should not be checked
```

```
Scenario: Setting not safe for work
  Given I'm marked as not safe to work
  When I go to edit profile page
  And I mark myself as safe for work
  Then I should be on edit profile page
  And the profile not safe for work checkbox should not be checked
```

Figure 4.8 – Scenario that tests two things (left) and a more atomic version of it (right)

not small. And if you need lots of lines to do the same thing, it's probably not small while pointing out to the example in Figure 4.7.

(B) Test One Thing

Two participants, P5 and P6, reduced the word small to atomic – the fact that a test describes one thing. P6 provided the example shown in Figure 4.8, while also suggesting a more atomic version of it. He described that example as not atomic because *it's testing two things, it's describing two things. These BDD scenarios are obviously written not to describe functionality or be testable – these are written as test scripts*.

(C) Code Step Small

P16 declares that *a scenario cannot be neither small or big, It has to have enough steps, and well described and it's good already*. He sees *small as in the implementation part of that step*.

(D) Not useful

For P12, small is not necessary to evaluate BDD scenarios, as *[a scenario] has to follow the concept of ready, definition of done, of that case. If it's small, nice, but if not it has to be big*. P17 also did not consider small, but for a different reason as *small could refer more broadly to the behavior itself, and that's why I was hesitant to use the word small, because behaviors themselves may not be small behaviors, they may be very big behaviors they may be very complex behaviors*.

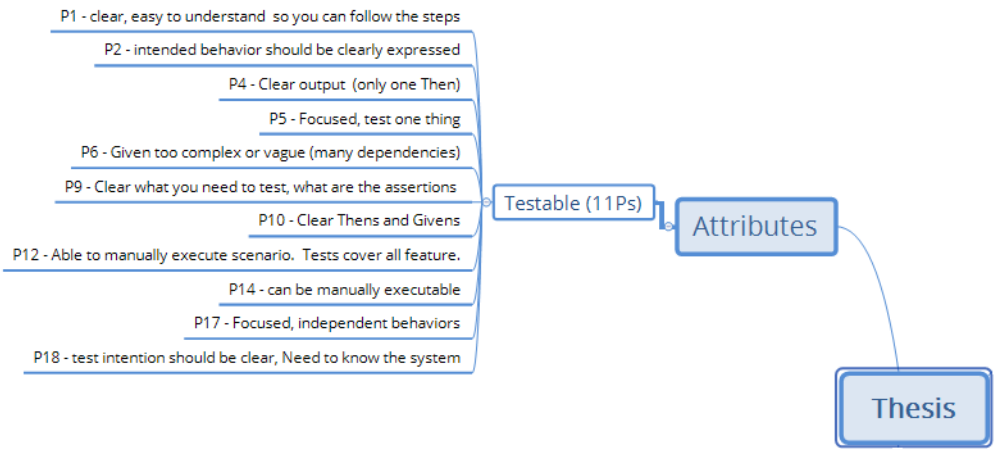

Figure 4.9 – Testable attribute interpretations

Finally, P18 also disregarded the small word, as *pretty much on all the agile teams I worked, [small] does not say so much about the requirement but about the deliverable, in other words, we pick up a piece of work that we can manage and not a big piece of work that is risky to do*.

4.2.3 Testable

Testable attribute was elected important by 11 participants – a summary of their interpretation can be found in Figure 4.9. A testable scenario is one who allow the reader to follow its steps (P1, P12, P14, P18), has clear outcomes (P2, P4, P9, P15) and pre-conditions (P6), is independent (P16, P17), focused (P5), and cover all feature (P12), if we analyze all the scenarios together.

(A) Clear Outcomes and Verifications

For some participants testable means that there are clear outcomes in the scenarios. For P2, testable means that *scenario's intended behavior, or what you are trying to verify, should be clearly expressed in the scenario*. P4 said that a scenario is testable *if it has a clear output, if it has only one Then statement*. P9 also highlighted the need of clear verifications by calling it assertions while stating that a scenario is testable *if it's clear what you need to test, what are the assertions you need to make, it has to be a clear indication of what it is you are asserting*. P15 said that what turns a scenario testable is the fact that *Then step gotta be an outcome you can verify, so the Then step has to be something that can fail*. P18 said that *testable means every time I follow this steps and I'll verify the same behavior*.

(B) Follow the Steps

However, for P1, be a testable scenario means *it should be clear, something easy to understand so you can follow the steps*, like the scenario in Figure 4.10. That same idea is explored by P12 and P14, who said that it is important to be able to manually execute the scenarios. P18

```
Scenario: Change my preferred language
  When I select "polski" from "user_language"
  And I press "Change language"
  Then I should see "Język został zmieniony"
  And "polski" should be selected from "user_language"
```

Figure 4.10 – Testable scenario due to the ability to follow the steps

also said that *testable means every time I follow this steps and I'll verify the same behavior – that every time I follow, I setup this scenario and know I'll get the same results*, and concluded by saying that he *would need to know what the system is actually capable of doing*.

(C) Clear and Simple Givens

P6 stated that testable attribute is reflected on the Given steps, as *if your Given conditions are overly complex or very vague then stuff becomes hard to test, that's mostly because the setup for a test is overly complex. And if the setup is overly complex than it means your software architecture is overly complex.*

(D) Focused

The focused adjective, already mentioned in the concise attribute section and exemplified in Figure 4.5 was also associated to testable attribute. For P5, *the Given/When/Then [structure] helps, to give you focus, to test one thing. And also, not go Given, When, Then, When, Then, When, Then. Because then you are writing a novel, not a scenario. Given this and this, And this, And this, And this...then maybe you are doing something that is not easily testable because it have too many pre-conditions. So we would say Given this and this, And And And And And, When, And And And And, Then, And And And. What are we testing now? Or maybe we are trying to do too much at once?*.

(E) Independent

Independence of behaviors was also associated with testable attribute. P16 said that a testable scenario should be *as auto-sufficient as it can* in order to achieve independence of scenarios. P17, when asked what attribute would be impacted if one scenario depend on another, answered: *testable, because if you have interdependent scenarios, that means the problem on one scenario might affect the other. So run one scenario and that should be ok, run it right after the other and it may ruin everything. So, absolutely testable.*

(F) Completeness

Additionally, for P12, testable represents how complete the scenarios' coverage of the feature behavior is, saying that *if it's not testable I can't even measure the test's coverage*.

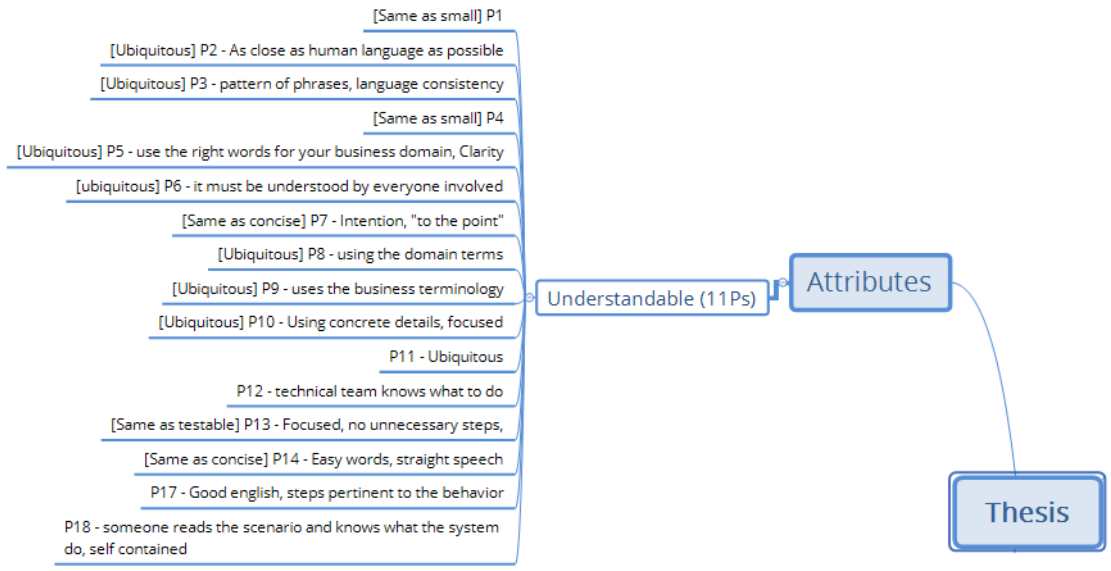

Figure 4.11 – Understandable attribute interpretations

(G) Not useful

Participants P3, P7, P8, P10, P11 and P13 said that testable is not related with scenarios, but with the product or system being tested.

4.2.4 Understandable

Understandable attribute was also deemed important by 11 participants – a summary of their interpretation can be found in Figure 4.11. For some participants, an understandable scenario uses an ubiquitous language (P2, P3, P5, P6, P8, P9, P10, P11, P15), is self-contained (P18), and is written in "good english" (P17). For others (P12, P16), understandable means only that technical people understand what to do. The remaining participants (P1, P4, P7, P13, P14) associated understandable as part of other attributes.

(A) Ubiquitous

P11 defined understandable as *quite related to a term we use which is called ubiquitous. What that means is that you basically come up with a terminology which is ubiquitous. What that mean is that all your project team members they are aware of that terminology, and you do not use that terminology where is too technical or business focused alone. It's a terminology that both your technical and business people understand and it is specific to your domain, So, if you do that, all of the scenarios they will be understandable, they will be easy to understand, for everyone, irrespective of their background*. P8 reinforced that idea by saying that *an understandable scenario for me is the one that's using the domain terms*. P9 declared that *it makes sense in terms of the system that is being built*. P10 used the term *concrete details* to refer to the same concept of using business terms on scenarios, by saying that *if you have concrete details you are able to use the same words*

```
Scenario: Someone likes my post
  When I like the post "I like unicorns" in the stream
  And I sign out
  And I sign in as "bob@bob.bob"
  Then I should see a ".likes" within "#main-stream .stream-element"
```

Figure 4.12 – Not understandable scenario due to the CSS element on Then step

as business use and turns out you get something that we all understand what you mean and we talk about the same thing. P15 also suggested the use of an ubiquitous language in the addition of the use of a glossary – he declared that *the terms in your glossary should be on your scenario*.

For P5, in order to have an understandable scenario a writer have to *make sure to use the right words in your business domain, so people in your domain understand it* – something also related to ubiquitous term seen before. P5 continued by saying to *make sure to, on the other hand, not use a particular jargon with HTTP code, that someone may not be familiar with* – the same advice to avoid technical details and make scenarios more understandable was given by P2. P2 also said that *they [the scenarios] should be as close as human language as possible*. In a similar way, P6 stated that *by understandable you imply that it must be understood by everyone involved. And ahn, when I look at the likes.feature [like the one in Figure 4.12], with the CSS elements, that's not understandable by everybody involved*. Finally, P5 compared understandability with clarity by saying that *if you are being too vague, if there's too much room for error, than it would mean something different than I do*. P3 also quoted clarity when saying that *cannot understand [a scenario] for the first few steps* and suggesting the use of a *pattern of phrases* to define language consistency. As all those declarations are related to the consistent use of business language, we also map it to ubiquitous term.

(B) Self Contained

For P18, an understandable scenario means *is when someone, who has never seen it before, reads the scenario and knows ok, this is what a user is able to do, so it is self contained, includes all the information needed*. That fact could be also covered by the ubiquitous terms, but the participant have not make use of that definition.

(C) Same as Others

For P7 and P14, an understandable scenario is straight and *to the point* – quite similar to their definition of the Concise word. P1 and P4 declared understandable as the same as small. P13 made it the same as testable.

(D) Good english

Participant P17 interpreted understandable as in the use of *good english*, the use of *proper grammar, proper tenses, proper point of view*. When referring to tenses, his views are: *the Given*

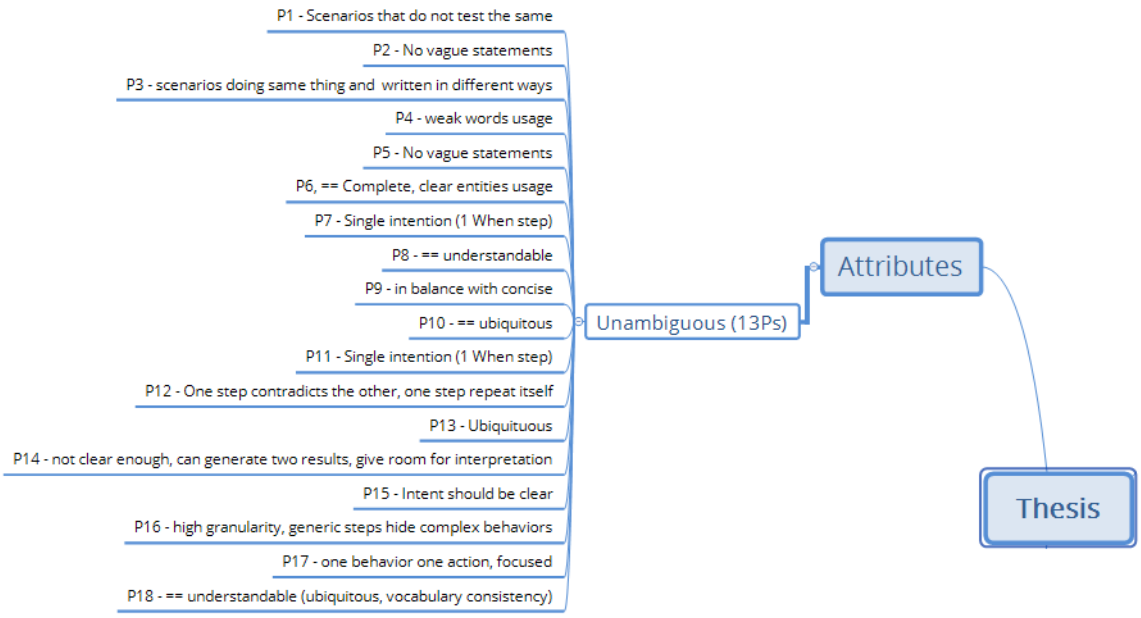

Figure 4.13 – Unambiguous attribute interpretations

steps should either be on past tense or present perfect tense, as it states things you already have; When steps should be active tense because it's something you are doing; Then steps should be present or future tense because that's an outcome that's expected. The same concern with steps tenses was briefly mentioned by P1, but not associated with any attribute. When referring to the *proper point of view*, P17 referred to the use of *1st person point of view versus 3rd person point of mine, are you saying I and mine or are you saying the user and the app*. For P17, *1st person is somewhat ambiguous*.

(E) Technical People Understand What to Do

For P12 and P16, understandable scenarios are the ones where technical people understand what to do. For P12, *I read and have no questions on what am I suppose to do. It should have a specific focus, it should take into consideration environment details and business details*, For P16, *it becomes understandable if it's technical, if it has a low granularity on an imperative format with specific steps*.

4.2.5 Unambiguous

Unambiguous attribute was reported as relevant by 13 participants – a summary of their interpretation can be found in Figure 4.13. Ambiguity on scenarios was interpreted as the use of vague statements, weak words and contradictions (P2, P3, P4, P5, P12, P14), and the lack of a single scenario's intention (P7, P11, P15, P17). Additionally, some concerns were mapped to this attribute, as the fact that scenarios with different descriptions test the same thing (P1, P3), the lack of enough test coverage (P6) and the steps' high granularity (P9, P16).

(A) Vague Statements

One of the sources of ambiguity is bad use of language by using vague, not clear, statements such as *Then the outcome is ok* or *Then the result is good* (both suggested by P2) or *When we see the change in the GUI* (suggested by P4), opening too much room for interpretation (P5, P14). Also, P12 raised the concern of having one step contradicting the other.

(B) Single Clear Intention

Another source of ambiguity is when the scenario intention is not clearly stated - quite often by the use of multiple When steps (P7, P11). P15 was also concerned with imperative steps, where the action was clear enough but the intention was not, like on the one in Figure 4.5. P17 agreed with that concern and said that an unambiguous scenario is one that has *one behavior, one scenario*.

(C) Different Scenarios Testing the Same

Despite the scenario's description, P1 and P3 reported that different scenarios may be doing the same thing, even if written differently (P3). P1 exemplified it saying that *if you need to test if the connection of the USB on your tablet is working you'd need to be sure the tablet is working, even if the connection is not. So the USB could be broken or the port of the USB could be broken - in the end, you are testing the same thing, because from the testing standpoint of view you are just checking if the tablet will behavior the same if you do not have this USB connection*.

(D) Completeness

For P6, scenario's coverage of the feature should also be mapped into unambiguous attribute, as well as the use of third person rather then first person statements – referring to *the admin user* rather than *I login as admin user*.

(E) Ubiquitous

P13 declared that unambiguous maps to the ubiquitous term, already mentioned for some other participants as an interpretation for understandable.

(F) High Granularity

The participant P16 declared the use of steps written with a declarative language as another source of ambiguity due to the fact that it can hide a lot of implementation details. P9 described the same concern as P16, saying that understandable should be in balance with concise *because the less ambiguous they are, the more complicated they might need to look*.

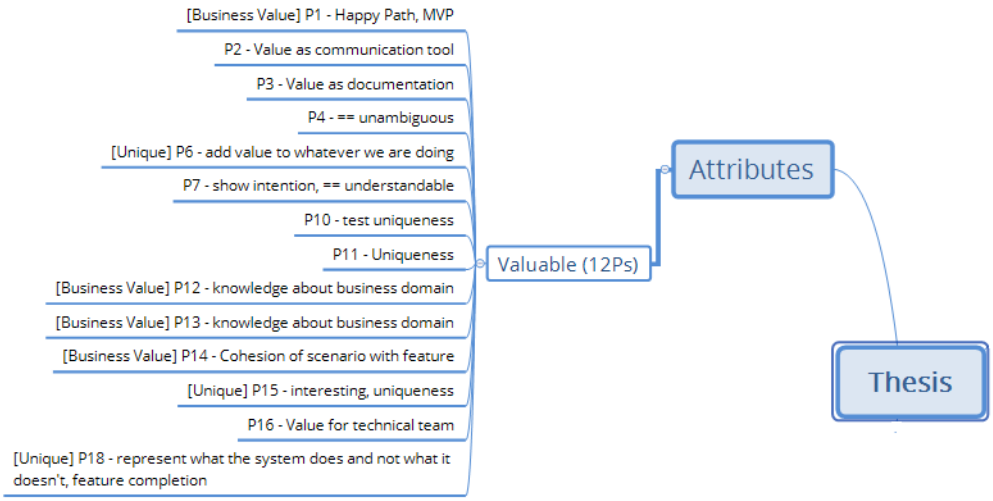

Figure 4.14 – Valuable attribute interpretations

(G) Same as Others

For the other participants (P8, P9, and P10) unambiguous attribute should be the same as understandable due to the fact that the ubiquitous term covers both attributes.

4.2.6 Valuable

Valuable attribute was considered important by 12 participants – a summary of their interpretation can be found in Figure 4.14. A valuable scenario is unique (P6, P10, P11, P15, P18) by validating some different and interesting behavior. Some participants could not identify how valuable a scenario is by reading their scenario description alone – that characteristic would have been discussed with other team members during formal or informal conversations (P5, P8, P9, P17). Some participants preferred to say a scenario should have value for the business (P1, P12, P13, P14), to the technical team (P16), as a documentation source (P3), or as a communication tool (P2).

(A) Unique

For P6, *valuable makes sense because you have to think of a scenario - is this scenario going to add value to whatever we are doing?*. P10 exemplified it further by saying that a scenario is *valuable if we are talking about something that is not covered previously. It's not valuable if we verify that we can do the same thing multiple times. We can search for different books, different titles, we can buy them if we are talking about an e-commerce site, but it's not so valuable to verify that we can buy multiple different titles, it's enough to verify if we can buy one or maybe two, but do the same thing again does not bring us value, it's just more of the same*. For P15, value *is interesting – is it testing something fundamentally different to the other scenarios*. P18 defined valuable by saying that scenarios *should not be repetitive, if you are having multiple scenarios and*

they are testing the same behavior, they are redundant – it's valuable if you have one, that cover it, and that's it. Finally, P16 declared that *[scenarios] should have a certain importance on their own*, indicating something similar to uniqueness as well.

(B) Business Value

P11 agreed that uniqueness should belong to valuable, but that attribute should also cover the importance of a scenario for business people. P1 said that her team *create the scenarios that has more value for the implementation part - like the happy path of a feature, as we need to be sure it's there as this is what is expected to work because this is our main deliverable*. Both P12 and P13 decided how valuable a scenario is by their knowledge about their business domain. P14 decided what are the important scenarios for the feature by the fact it employs important actors or *buying* customers – also based on his business knowledge and the value to the business.

(C) Covered on Conversations

For P8, valuable comes from discussions with the team, as they *only makes scenarios for things that are valuable*. For P9, valuable has *more to do with Scrum and Agile rather then BDD scenarios*. P5 defined it as *what is worthy testing and what's not worthy testing*, and continued by saying that *I might think that hey, this could go wrong, is that something that should be tested and either yeah, it could go wrong and the probability is low so never mind or that could never happen for technical reasons, so never mind*. For P17, *the value of the scenario is really on the value of the behavior, how important is this behavior for my team, how important is this behavior for the feature*.

(D) Other values

Some other meanings of valuable appeared: for P2, those scenarios are a communication tool, so the communication and acting up on the misunderstandings that's what's valuable; for P3, *what value it brings to me is the documentation*. Those opinions did not tied how valuable a scenario is to their scenario description, and therefore could not be used to evaluate a scenario's quality.

For P16, the scenarios should have value to the technical team alone by being described in a low details' granularity.

(E) Same as Others

P7 said business value is represented by showing the scenario intention, thus being understandable. For P4, *the value part would be unambiguous to me without additional elaboration on what's a valuable scenario description*.

Table 4.3 – Summary of removed attributes

Reason	Estimable	Negotiable	Feasible	Prioritized
Technical knowledge	P1, P5, P6, P10, P18		P1, P6, P8, P10, P17	P8
Domain knowledge	P3, P12, P13	P12	P3, P12, P13, P16	P8, P12, P14
Conversations	P11, P15	P5, P7, P11, P15, P17	P3, P7, P11, P14, P15	P3, P7, P11, P15
Only for Features	P14	P2, P14	P2, P15	P2, P3, P6
Not applied at all	P2, P4, P7, P8, P17	P1, P4, P6, P8, P10, P13, P18	P4, P5, P18	P4, P5, P10
Other	P16 (granular, imperative)	P16 (PO)	P16 (steps reuse), P17 (feasible to automate)	P1, P3, P14, P17, P16 (PO), P13 (happy path), P18 (Tags)

4.2.7 Removed Attributes

Prioritized, Negotiable, Feasible, and Estimable were largely regarded as not useful to evaluate BDD scenarios textual descriptions, as either they are useful only for conversations around scenarios, suited for feature file (sets of scenarios) only, dependent on steps' technical implementation knowledge, or are project's domain knowledge – as summarized in Table 4.3.

P11, who declared those attributes as useful for conversations only, explains his thoughts about those attributes as follows: *So I think the other would already have been considered by the time you reach that stage, because, so if you are remember I was talking about example mapping, which is a step which comes before feature files. So, the way you write your examples, so when you write your examples you need to make sure they are estimable for example. And obviously your scenarios would be derived from examples and therefore will automatically be estimable. So it's more relevant at the time of example mapping rather than feature file*. Similar declarations were heard from all the participants who either have conversations before writing scenarios or validate them after they are written as listed in Table 4.2.

(A) Granular and Imperative help on Estimable

Additionally, some participants (P12 and P16) declared that steps with a low granularity, written in an imperative way and carrying fine grained details, help the technical team to estimate the effort to implement them. P12 have developed a system to aid estimate his work with function points [31] by counting the amount of steps of inputs, outputs, integrations, and so on. P15 agreed with it, but declared that *it may be more estimable all the time, but coming back it loses all the*

value. As other participants declared that estimable was not useful for textual scenario quality evaluation, we removed that attribute from our list of potential ones.

(B) Prioritization

Prioritization was another characteristic that was more suited to conversations around features than to analyze textual BDD scenarios. P1 and P16 went as far as saying that the PO defines it, alongside the client. P13 suggested that happy path scenarios may have more priority - a characteristic that P1 had assigned to valuable attribute. P3, P14, P17, and P18 suggested the use of tags to tell a scenario priority, using a scale with "low", "medium" and "high" priority – a characteristic explored on later sections, which also helps on valuable and understandable attributes. Others (P1, P2, P4, P5, P6, P7, P8, P9, P10, P11, P12, P15, P16) have disregarded prioritization as interesting to evaluate textual scenarios, so we decided to also leave that attribute out of our list.

(C) Automation Characteristics

Finally, some comments were more directed to the test automation side of BDD scenarios. P17 is used to review textual scenarios alongside their steps code, so feasibility for him was described by how possible that scenario is to automate – a characteristic that was deemed unimportant for textual scenarios by others. P5 had show concerns with test execution time, but have not assigned it to any attribute – therefore, we judge this is not something she would use to evaluate the quality of BDD scenarios. P16 was concerned with the ability to have generic steps, easy to reuse and to create new test cases with different parameters, aligned with his opinion that a valuable scenario is the one more useful for the technical team – again, not associated only with the textual description of a BDD scenario, our research focus.

4.2.8 Additional Attributes

Some characteristics outside our list were quoted as important as well, like completeness (P1, P4, P11, P18), coherence (P3, P8), cohesion (P12, P16), integrity (P17), and declarative rather than imperative language (P10, P11, P18). As they were not our investigation focus, not all participants had the chance to provide their opinions about them, so we will just list them here for future reference. Each participant addition is shown in Figure 4.15.

(A) Complete

P1 defined a set of scenarios complete if, during a review, he detects a scenario is missing, what could means the product would be delivered with an unexpected bug. P4 defined it when the scenarios requirements are not entirely covered. P11 suggested that all the scenarios should cover the feature completely – therefore also adding completeness to the list. Finally, P18 assigned completeness to valuable when considering the feature file as a whole or a set of scenarios.

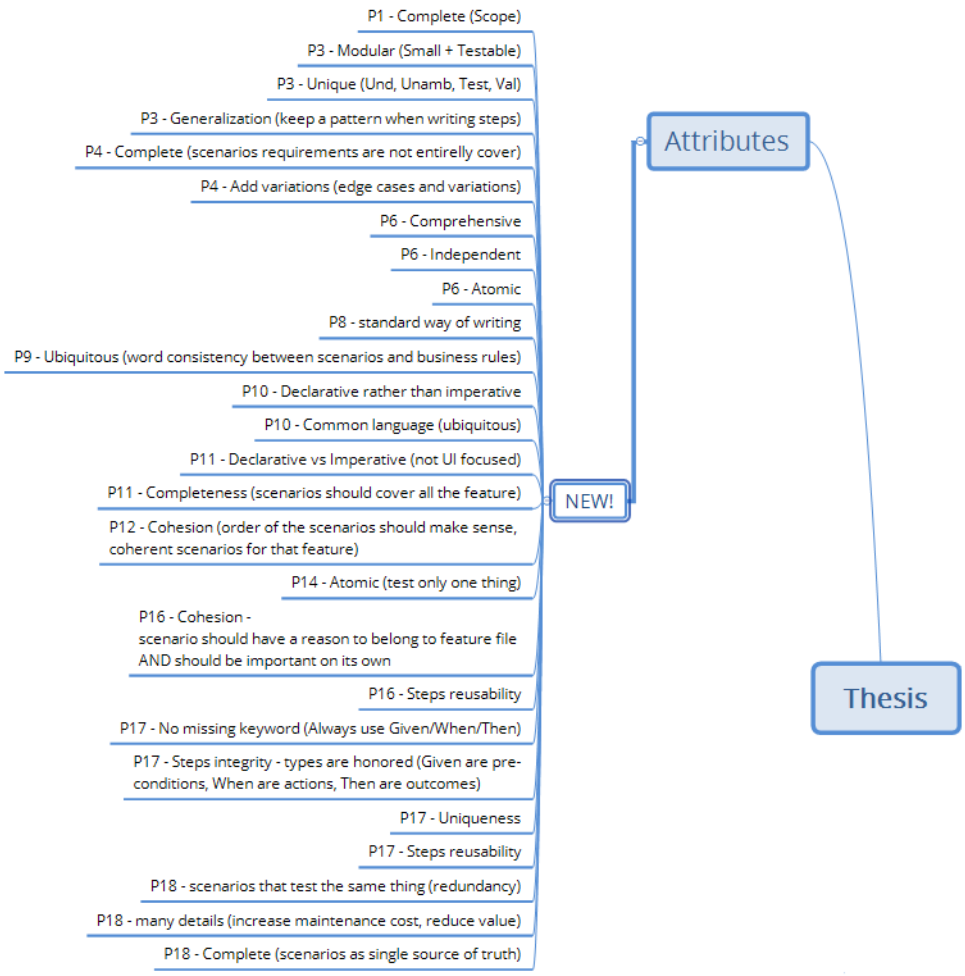

Figure 4.15 – New attributes

On a related way, P4 highlighted the importance of *adding variations* to scenarios, thus having only 1 or 2 scenarios in a feature make it *very thin*. When variations are added, one has to pay attention to missing scenarios as well. When analyzing the scenarios on the profile photos feature file in Figure 4.16, P4 declared it to *raises more questions or concerns*, as *there is no thing in this feature if Robert deletes a profile photo, so can Alice continue to see it? That is not really described, so that would raise a concern on me, something that says the feature is not complete*.

(B) Writing Patterns

Coherence was referred to P3 when talking about how his team *define the semantics on how you write the scenario [...] you follow the same pattern even if you have a different application*. According to him, *you can have 3 or 4 business annalists working in the same application, if it's really big, and everyone can come with their own way of explaining a scenario – so if you define a semantic like, first it should be, you know, the user or the something, the user or the entity for example and then it should be followed by a word then the context. These kinds of semantics really helps everyone to follow the same pattern. Or at least they would be written in similar pattern*. P8 referred to it as *consensus about the scenarios*, when stating that *BDD projects should be under*

```
@javascript
Feature: show photos

  Background:
    Given following users exist:
      | username      | email               |
      | Bob Jones     | bob@bob.bob         |
      | Alice Smith   | alice@alice.alice   |
      | Robert Grimm  | robert@grimm.grimm  |
    And "robert@grimm.grimm" has posted a status message with a photo
    And I sign in as "robert@grimm.grimm"

    Scenario: see my own photos
      When I am on "robert@grimm.grimm"'s page
      And I press the first "#photos_link"
      Then I should be on person_photos page

    Scenario: I cannot see photos of people who don't share with me
      When I sign in as "alice@alice.alice"
      And I am on "robert@grimm.grimm"'s page
      Then I should not see "Photos" within "#profile-horizontal-bar"

    Scenario: I can see public photos of people who share with me
      When "robert@grimm.grimm" has posted a public status message with a photo
      And I sign in as "alice@alice.alice"
      And I am on "robert@grimm.grimm"'s page
      Then I should see "Photos" within "#profile-horizontal-bar"
      When I press the first "#photos_link"
      Then I should be on "robert@grimm.grimm"'s photos page
      And I should see "Photos" within "#profile-horizontal-bar"

    Scenario: I delete a photo
      When I am on "robert@grimm.grimm"'s photos page
      Then I should see a ".thumbnail" within "#main-stream"
      When I confirm the alert after I delete a photo
      Then I should not see a ".thumbnail" within "#main-stream"
      When I am on "robert@grimm.grimm"'s page
      Then I should not see "Photos" within "#profile-horizontal-bar"
```

Figure 4.16 – Profile photos feature

consensus so if they are using this way then it should be used always that way within the same project. Therefore, coherence, or keeping the same standard way of writing, is another word that could be used to evaluate sets of scenarios.

(C) Cohesion

Within the context of component-level design for object-oriented systems, cohesion implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself [31]. Within the context of BDD scenarios, P16 said that *scenarios should have a reason to belong to a given feature file* and P12, while also agreeing, added that the order the scenarios are presented on a feature file should make sense.

(D) Integrity

P17 highlighted the importance of using the three BDD keywords, Given/When/Then, in all the scenarios, while also keeping the integrity of their usage. He declared that *have sometimes seen people do, to try to circumvent the rule of strict steps ordering, is they will start turning Then verifications into When steps, they will say something like When I'm on Alice's page, And I should see look at this dog, Then I focusing And ... - they are violating the integrity of the steps type, and that's no more behavior driven than the original procedure*.

(E) Declarative rather than Imperative

Additionally, some participants (P10, P11, P18) highlighted the need to enforce the use of declarative language, as seen in Figure 4.5, by turning it into an attribute on its own, in separation of concise. P10 said that *you can think about one thing, and that can be done in two different ways, could be both declarative and imperative*, as his interpretation of conciseness focus only on the lack of unnecessary details. P11 also had the same focus and thus put the use of declarative language in separation of every other attribute. P18 interpretation for conciseness, to have *crispy, very clear* scenarios, also have not had room for the use of declarative language and therefore that characteristic was kept in separation as well.

(E) Same as Others

Finally, some participants have suggested to transform into additional attributes some characteristics assigned by other participants, like: uniqueness (P3, P17), already covered under valuable attribute; atomic (P6, P14, P18), already covered under the small attribute; steps re-usability, a characteristic out of our scope due to the focus on the scenario's test automation side; ubiquitous (P9, P10), already covered under understandable; independence (P6), already covered under testable attribute; and modularity (P3), already covered under the testable attribute section.

4.3 Participant's Criteria

In order to better understand how each attribute presence or absence is reflected on a BDD scenarios, participant's criteria, more tied to the actual textual scenario than generic attributes and similar to author's experience-based criteria [34][38], were taken from the answers of Questions 5 and 6 in Table 3.2. Additionally, those criteria were tied back to the attributes in Question 13 (Table 3.2), in the hope it would refine our literature-informed quality attributes and enhance them with good and bad practices – a refinement that have not be possible due to the multiple interpretations of the literature-based quality attributes.

The results presented in the next sections are also based on a partial mind map summarizing the participants opinions, shown collapsed in Figure 4.17. This summary will be expanded in each

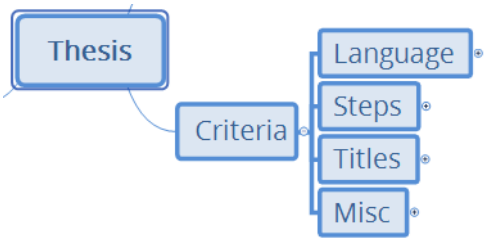

Figure 4.17 – Overview of the criteria to analyze BDD scenarios

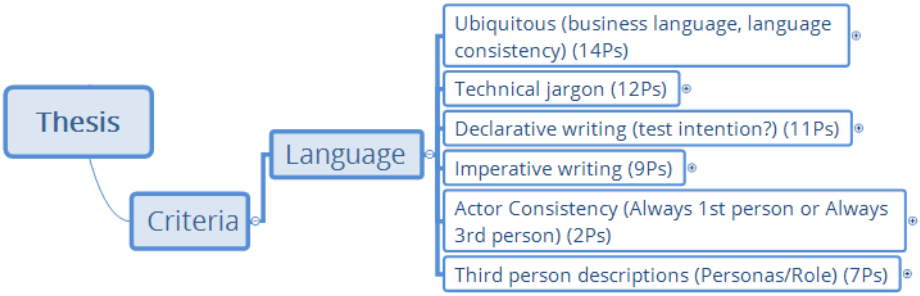

Figure 4.18 – Language criteria to analyze BDD scenarios

section, on which several criteria are grouped together into four groups: language criteria, steps criteria, title criteria, and others criteria.

4.3.1 Language

Language groups together characteristics, summarized in Figure 4.18, that depend on writing decisions, such as the use of business terms instead of technical ones, the declaration of actions rather than the description of steps and the subject point of view.

(A) Ubiquitous

The consistent use of business language, like *a glossary that appears consistently on all scenarios* (P9) is regarded mostly as a good practice. P9 also referenced it properly, saying that *in domain driven design that is known as ubiquitous language* – the same term that we mapped as one of the meaning of understandable attribute in a previous section.

Participants have mapped it as a good practice impacting understandable (P1, P2, P3, P5, P6, P7, P8, P9, P10, P11, P15, P18), unambiguous (P3, P8, P10, P13), and concise (P4, P11) attributes and as a pain-point for the estimable (P3) attribute – for P3, *if it's on technical language it's very easy to estimate*.

(B) Technical Jargon

In opposition, the use of technical language, or technical jargon, as expressed in Figure 4.12, is considered as harmful for understandable (P1, P2, P5, P6, P8, P9, P11, P14, P17, P18),

concise (P2, P10), unambiguous (P2), small (P10), valuable (P7), and testable (P10) attributes. Other examples of technical language would be the use of HTTP response codes (P5), or automation steps on the UI like *When I click a button*.

(C) Actor Consistency

Despite the consistent use of business language terms, some participants also pointed out the need to write steps using third person point of view. P18 highlighted the use of third person as a good thing as *it could get your whole team to think like the user and that would be very useful*. P18 continued by saying that *"I", as in "I reading it", or "I the QA" or "I the developer", what I do might be a bit different of what the actually would be doing. It helps to set a mindset, when you are thinking more about the user*.

A way to use third person point if view is to build personas – fictional characters that are meant to represent the different types of people who will be using the system, as pointed out by Smart [34] on his book about BDD. P2 exemplified this referring how the use of "alice@alice.alice", on scenarios such as the one in Figure 4.5 with a step *When I sign in as "alice@alice.alice"*, could be changed to *Given I sign in as Alice*. He explained that decision saying that *you have a object, a persona, called Alice, with a number of properties, one of them being her email address, and you have that, so you do not have to specifically define those variables again*.

Another way to use third person point of view is to describe roles of users, as described by P6: *but if you say "an user must be logged in" that's already a bad line because the user can have a variety of meaning, it can be, it can have administrator or a guest user or, no, the user must be logged in so it cannot be a guest user. It can be an user with access rights type A or type B or anything like that. What you should say is "an user in group XYZ" or "the user with template ABC" and then you have a bunch of users template where you define what your actual setup is*.

The use of all forms of third person point of view is considered as a good practice and enhances the understandable (P2, P7, P17), concise (P9), unambiguous (P6), and valuable (P14, P18) attributes. Additionally, some participants (P16, P17), while inclined to use third person point of view on their steps, were already happy only with consistency - using always first person or always third person.

(D) Declarative rather than Imperative

Despite the language terms and the point of view, there is also the concern about how granular a scenario step description should be, as exemplified in Figure 4.5.

P2 said scenarios should *focus on the problem, not the solution* and P8 called it as *declarative*, with *imperative* being his opposite. For P17, *Gherkin is meant to be declarative because it tries to describe behavior that add business value, it's not necessarily to give the implementation on how that behavior works*.

P17 explained that difference in the writer mindset, saying that *the difference between declarative and imperative is in how granular your details are in your steps*. He also said that *the declarative way to say something would be "Given the user is logged into the app", very simple, very descriptive, it's not necessarily saying how is done, it's saying what is done*.

P5 warned about how imperative writing may impact the maintainability of scenario, saying that *if you write it like this [in imperative form] and the UI changes than this test will break. Whereas if you write it further, "I block the user so and so", the change you have to make is much smaller, because it's at a lower level, somewhere in the supporting code*.

In opposition, P17 also described the imperative way of writing this action as *Given the user enters www.whatevermywebsiteis.com And the user enters the username field And the user enters andrewknight And the user enters the password field And the user answers abc123 When the user clicks the login button Then the page is rendered*. For P17, *ultimately, for the automation sake, those specific actions would need to be taken, because you need to use something like Selenium web driver to click various elements on the page, but the thing is, that's impertinent at the Gherkin level right, Gherkin is meant to be descriptive, it's meant to be high level, it's meant to be business oriented, it's meant to be behaviors and not necessarily neat gritty little steps*.

For those participants who mix scenario's writing with conversations, either before or after the writing of them, the use of declarative language is seen as a good practice that enhances understandable (P7, P8, P9, P14, P15), testable (P1, P2, P5), concise (P8, P17), and unambiguous (P14, P15) attributes. Imperative language usage was considered as harmful for testable (P1, P18), small (P5), understandable (P7), and concise (P17).

However, P12 and P16, who use scenarios with a technical approach in mind, considered declarative form of writing harmful for understandable (P12, P16), estimable (P16), feasible (P16), testable (P16), unambiguous (P16), and valuable (P16) attributes. Imperative writing was considered a good practice that enhances estimable (P12, P16), understandable (P12, P16) feasible (P16), testable (P13, P16), unambiguous (P16), and valuable (P16) attributes.

4.3.2 Steps

Steps characteristics, summarized in Figure 4.19, look at how many steps a scenario have, how long are they, and what step keyword to use.

(A) Few and Short Steps

Having long scenarios with many steps, as shown in Figure 4.5, is considered harmful mainly for concise (P2, P3, P6, P7, P8, P9, P15, P17, P18) and small (P1, P3, P4, P5, P10, P11, P13, P14), as already discussed on those attributes' sections. There were also reports about it affecting unambiguous (P3), understandable (P14), and testable (P18) attributes. In a similar

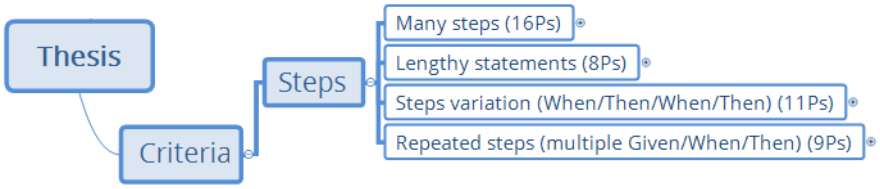

Figure 4.19 – Steps criteria to analyze BDD scenarios

```
Scenario: expand the comment form in the main stream and an individual aspect stream
  Then I should see "Look at this dog"
  And the first comment field should be closed
  When I focus the comment field
  Then the first comment field should be open

  When I select only "Besties" aspect
  Then I should see "Look at this dog"
  And the first comment field should be closed
  When I focus the comment field
  Then the first comment field should be open
```

Figure 4.20 – Scenario alternating When/Then steps

way, lengthy statements, as shown in Figure 4.7, are considered harmful for concise (P3, P5, P7, P9, P17), small (P1, P3, P4, P10), and understandable (P17) attributes.

P17 said that, *the more imperative you write your scenarios the more lines it will have, so if you have too many lines, you can kinda guess, you can probably state some of those things a little bit better*, which could indicate that having fewer steps is not a proper goal, but having scenarios written in a declarative format rather than imperative would.

(B) Steps Order

Additionally, there was a concern with the natural step order of Given/When/Then being violated. On examples such as in Figure 4.20, from which P17 commented as a *major BDD violation*, which he explain saying that *one of the hard and fast rules that I apply for writing BDD scenarios is the strict ordering of steps, Given/When/Then order*. P2 said that alternating the use of When and Then words *means that you need to split up in two smaller scenarios* and that *a good test is exactly one verification and alternating when and then means you are testing more than one thing in the same scenario*. P6 described it as *expanding your scenario to cover multiple pieces of functionality*, and added that *I do not like chaining in scenarios because you are not considering those scenarios might take future changes. Because if you build your scenarios like this, then how can you build code that only does one thing - you get bad code as well, as a result of bad scenarios*.

Therefore, this violation of the natural step order is considered a bad practice that affects understandable (P2, P4, P8, P10, P14, P17, P18), concise (P2, P3, P8, P9, P14, P18), testable (P2, P7, P18), unambiguous (P3, P14, P17), valuable (P3, P18), and small (P6) attributes.

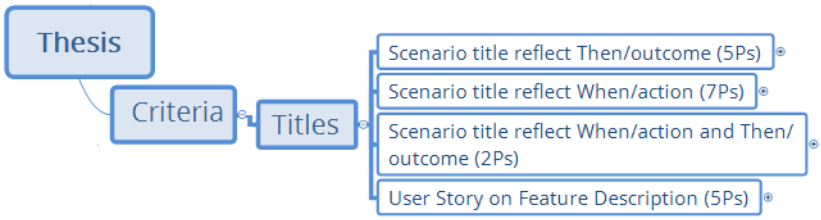

Figure 4.21 – Title criteria to analyze BDD scenarios

```
Feature: commenting
  In order to tell Alice how great the picture of her dog is
  As Alice's friend
  I want to comment on her post
```

Figure 4.22 – Example of a Feature description on comments.feature

(C) Steps Repetition

In a similar way, the multiple repetition of the same step, demonstrated by the excessive use of Given/When/Then steps in sequence or the "And" keyword, is also judged a bad practice. P4 summarized it saying that *If there are many Ands than it makes it probably harder to understand*. Additionally, P6 declared that multiple repeated steps, like the example in Figure 4.8 may indicate those scenarios were *written as test scripts* and *not to describe functionality*.

This repetition of the same step bad-practice affects unambiguous (P4, P7, P11, P12, P14), concise (P4, P7, P8, P14), testable (P4, P6, P9), understandable (P4, P7, P14), and small (P5) attributes.

4.3.3 Titles

Despite analyzing the language and the steps characteristics, participants were also asked to analyze the scenario and feature file titles and descriptions. The characteristics that emerged are summarized in Figure 4.21.

(A) Feature Description

Feature titles and descriptions were often pointed out as *intended business outcome or the indented business value* (P2) in one way or another. That good practice would enhance the understandable (P4, P14) and valuable (P7) attributes.

Some participants (P11, P15, P17) have suggested using the user story standard description on feature descriptions. However, P8 did not believe feature descriptions are useful at all, even worse if they have user stories descriptions like in the example in Figure 4.22. *In this case [comments.feature] this is a user story description which I generally do not like that my feature files to be structured based on user stories because the user stories are somehow arbitrary split of the implementation so it depends on what fits into your Sprint, what is current priority*. And continues

saying what he really want to see: *I just want to see right now the behaviour of my application. [...] This feature file is about comments but I do not see any benefit on a user story description like this, so maybe there's lot of other actors that can do comments, so I generally do not put the user story text into the feature files*.

Additionally, some participants (P10, P15) warned about the use of technical details on feature descriptions, such as in the phrase *Access protected resources using auth code flow*, which P15 declared: *I do not really understand what using auth code flow would be. Yeah I can understand using and I think I can understand what auth code flow might mean but there's probably some less technical that could be used there*.

(B) Scenario Titles

Regarding scenarios' titles, participants declared it should either express the intended action (P3, P5, P9, P10, P13, P16, P17) or the intended outcome (P2, P8, P12, P14), or both (P18, P11). Surprisingly, only few have declared it affects an attribute – from those who did, they mapped it with understandable (P9, P18), testable (P2, P15), and concise (P12) attributes.

Some participants were concerned with how good the correlation of the scenario title and a When step was. For example, when P9 reads a title that *says it's about a user mentioning another user*, he was *lead to assume that this is the main action, in this example, a user mentioning another user, but the action which would normally be found on the When step, is the "I sign in as Alice"*. For others, like P10, that also declared it's important to have scenario titles expressing the intended action, *it should not match any step, necessarily, and it should, however, be nice if it tells me what it is doing*.

For others, the intended outcome is enough, as exemplified by P2, who declared: *I personally like to see a desired outcome scenario name. So regarding the "Hovercards on the main stream" scenario, if I split it up for a single scenario for activation and a single scenario for deactivation I'd call the first one "Hovercards on the main stream can be activated" and the second one "Hovercards on the main stream can be deactivated" [illustrated in Figure 4.23]. And that's because if you just see scenario names into a reporting tool I'd like to know just with the scenario name and the execution result of that scenario I want to be able to determine what feature in my application works or does not work anymore*. For P15, this intended outcome is *what is I'm sort of testing*, while P8 called it *the goal of a scenario* and, when reading a title phrased as *Comment to the status show page*, he inquired: *what do you want to test here? Are you able to comment? To do a comment? So I would like to have something that describes kind of a business rule, something that says this was fulfilled or not fulfilled, not just a topic*. P17 is pleased without the outcome, as he declared that *many times when you read the action, the outcome is sort of implied*.

Additionally, there are others who want to see both, outcome and action, expressed on the title – as P11, who says that a scenario title *should briefly tell me what I'm testing and what I'm expecting as the outcome. So just, obviously if it takes too much space, you would not want to over-complicate things, you would want to keep it brief, otherwise it basically defeats the purpose.*

```
Scenario: Hovercards on the main stream
  Given I sign in as "alice@alice.alice"
  And I am on "bob@bob.bob"'s page
  Then I should see "public stuff" within ".stream-element"
  When I activate the first hovercard
  Then I should see a hovercard
  When I deactivate the first hovercard
  Then I should not see a hovercard
```

```
Scenario: Hovercards on the main stream can be activated
  Given I sign in as "alice@alice.alice"
  And I am on "bob@bob.bob"'s page
  When I activate the first hovercard
  Then I should see a hovercard

Scenario: Hovercards on the main stream can be deactivated
  Given I sign in as "alice@alice.alice"
  And I am on "bob@bob.bob"'s page
  When I deactivate the first hovercard
  Then I should not see a hovercard
```

Figure 4.23 – Breaking hovercards scenarios to allow better titles

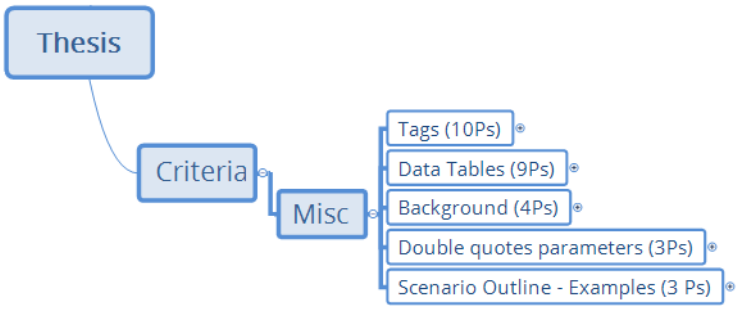

Figure 4.24 – Additional criteria to analyze BDD scenarios

So what you could do this, if you would write more text, just add a comment below the scenario keyword. So it would basically appear as informative test, just under scenario keyword, you can do that in a feature file.

A common goal shared by some participants (P2, P11, P17) was to have scenario titles help on failure reporting. As summarized by P11, if you are able to write good titles, *when this test report is shown to someone, your project manager, your business owner or your tester, they would instantly know, without looking at the steps, what exactly is being validated here, or what would you expected.* P2 added up for that goal by saying that *you are using an living documentation and this is not only for the specification side but also for the reporting side.*

4.3.4 Additional Criteria

Despite the scenario's language, titles and steps descriptions, Gherkin language allow other types of constructions such as the use of a background section, tags, scenario outlines examples, data tables or double quotes aid to pass parameters. Those characteristics are summarized in Figure 4.24.

(A) Background

For P18, a Background *sets the context for the rest of the feature. And when you start reading it, ok, these users will always exist for all these scenarios. So that's useful, but, that's fine,*

it's also very helpful when you try to automate things, because the Given can be defined to run before each test, so it also helps on that purpose. Actually, one last thing about the Background, potentially it can show, it can be sort of a sanity check to confirm that the entire feature in fact is about one feature so it's tied to one background, so it might help there as well. However, P15 reflected on the need of a background section, when asking himself: *does adding a Background make it harder for people to read? If it makes it harder for people to read, then no, it's a bad idea. If you've got a quite large number of scenarios, the Background should have a different title as well*.

P10 said a background section should not mix Given/When/Then steps and *it is most likely the Given stuff*. Additionally, confirming P15's opinion, P10 said that *where we have 3 scenarios in a feature file, I would not put it in the background. It could be argued that this is a duplication [...] but in almost any situation and specially in any test case, I prefer clarity over duplication*.

The use of background section can enhance the concise (P2, P4, P17, P18) and understandable (P2, P4, P17) attributes, as shown before in Figure 4.4.

(B) Tags

Tags were reported as being used for scenario's categorization (P1, P2, P3, P11, P12, P13, P14, P16) and as a communication tool (P3, P7, P10, P15, P17).

P1, who has scenarios that run on desktop or mobile, argued that tags can be used to *separate those scenarios*. P11 said *you could have some feature files tagged as end-to-end UI based scenario, some could be tagged as integration, some could be tagged as a component or you could also have feature based tags or module based tags*. P12, P13, P16 also use tags for run specific tests on specific situation

Additionally, P15 reported that *tag is all about metadata, so tags are all about adding information from a usage or metadata point of view which is not associated with the behavior*. P1 reported *there are tags for smoke tests – I have a regression pack which is quite huge and when we need to do a smoke test with just basic scenarios we tag those as smoke*. In the same way, P3 reported *it can give the context of the importance of the scenario or the feature*, and explained it further by saying that *if you categorize your feature and scenarios into mainly different areas, something like UI, or maybe say it depends on integration to start, you know... just a basic story, a back end test, it gives a sense of the importance of this scenario or feature. Where this feature stands from an application stand point of view*.

However, P5 warned about excesses. For P5, *if you are using the tags than you want to be able to run a certain test of sets, given a certain tag, and why would you want to run only a certain set and not all the tests? Because maybe something changed in this area, but if your test suite is fast enough you could run all of them. There would be no need to say in this case I will only need to run there and not those. So this could also be a code smell, excessive use of tags*. Additionally, P2 warned about the use of tags for technical purposes only, when he said that a tag *tells something about the implementation, not about the intention or the behavior of your application and specify something of the intent or the implementation of your test scenarios may*

not... if you want to group features or scenarios together I think you'd be better of just regrouping the features or placing features into folders, what is more understood by humans than tags that can appear whatever. I do not use those for technical purposes.

The use of tags affects valuable (P1, P3, P4, P14, P17), understandable (P3, P11), prioritized (P3, P14), and unambiguous (P3) attributes. However, P2 said it can harm understandable due to their technical nature.

(C) Data Tables

Data tables had caused different reactions from participants. As summarized by P8, *tables has pros and cons, such as, well, the pros are for repetitive tests that needs to cover a different range in the input and they could be considered one scenarios. The cons are that you might accidentally bunch many scenarios together that should be kept separate - so it should be less and less easy to read, less clear*.

P11 considered that *tables are a good idea, actually, so instead of having to you know write several lines if you can structure then in a nice table, obviously the table should not have 20 columns, you know, if it's a nice concise small table which is easy to read and as long as it does not have like 20 rows or 20 columns, it looks good, it's easier to read, it's better than writing 10 steps, as it's possible that you are able to summarize them in one table. And that makes much easier to read, as long it is a small, concise table with a minimum at steps, minimum rows and columns*. P14 also agrees with that same opinion of having small tables, and highlights the importance of headers on that table. P18 said *tables are a good way to input data, and that's why most people use Cucumber is because of data tables in particular*. P13 was in favor of even using multiple tables at a same scenario.

However, some participants (P1, P2, P7, P10) would not use data tables at all. For example, P1 said: *I would not use a data table, I'd use examples, because data tables are complex because it's just a bunch of data you can send but you do not really know what's the context in the step. Because you are not setting a variable value, that you have in the middle of the step*. P2 said that, when *creating an end-to-end test using a tool like Selenium*, using tables *is often a sign of bad automated test design if you use tables in those tests, because then basically you are repeating the same process, the same customer flow, with different parameters. And for me that's a sign that those tests with parameters should be executed at the lower level*. P7 dislike data on scenarios, completely. P10 even declared: *I'm not always convinced that they [data tables and examples] lift their own weight. I think it that, if you need so many examples, maybe we're actually starting to use Gherkin as a testing tool rather than a communication tool*.

The use of data tables affects concise (P1, P3, P11, P17, P18), understandable (P3, P7, P17, P18), unambiguous (P14), and testable (P14) attributes.

(D) Scenario Outline - Examples

In a similar way as tables, scenario outlines have received mixed reactions from participants.

P3 uses it *when I have to execute the same scenario for different type of data - there's no change in the scenario only data is different. So on the form example we were talking before, for the web page, so there maybe I entered the data and verifying the validity, but depending on the data my result is different*. P5 also uses it, in a pragmatic way, as stated: *so we used these scenarios outlines with examples to check the different type of cases. It's a really easy way to get all of this, to test all of these cases, to check if all the different options are covered and are correct. But I also understand how, you know, the fact that they are so easy to use and it's so easy to add more test cases is also a risk because then you end up with huge test suites and you have to be critical of does each example actually add value*.

However, P15 argued that *they are generally misused and often a code smell. Because they lead people time to test combinatorial testing, the test of all permutations of a particular situation. That's not what BDD is for - BDD is to tell you What you are gonna do and you have all other tests to show you that you build it right*. P17 enforces a rule on his code reviews – *every time that you have a row in your example table of the scenario outline, you have to justify why there's an equivalence class for unique input into scenario – because I've seen teams, also I work with other teams like internally consultancy to get them to do BDD practices, and on those teams specially I've saw people trying to blow like 500 rows in an example table and I have to go back to them and say do you really need all those rows, because 500 rows means 500 test iterations, is that really what you, is that the best usage of our time, as resources, test runs over night*.

The use of scenario outlines affects concise (P17, P18), understandable (P17, P18), unambiguous (P1), and testable (P17) attributes.

4.4 Newly-defined Quality Attributes

In order to analyze our results, we first have to understand what we have: in one hand, we have multiple interpretations to traditional attributes; at the other, we have participants' personal criteria, attached to BDD scenarios details.

Each of those interpretations of each attribute can be seen as a potential attribute, as we cannot decide for ourselves what interpretation of each attribute is best suited than others without enforcing our own bias. In order to couple those interpretations with BDD scenarios, we mix them with participants' personal criteria, that are naturally attached to BDD scenarios. Those are now referred to as characteristics, grouped to similar ones to define 8 quality attributes. The full mapping of the mentioned characteristics into our newly-defined quality attributes can be found in Table 4.4, while a summary of the newly-defined quality attributes can be found in Table 4.5

Table 4.4 – Newly-Redefined attributes mapping to characteristics

Attribute	Characteristic
Essential	(Concise) Not Too Many Details
Essential	(Concise) Not Too Many Steps
Essential	(Concise) No Unnecessary Lines
Essential	(Small) Not Many Steps
Essential	(Steps) Few and Short Steps
Essential	(Steps) Steps Repetition
Essential	(Additional Constructions) Background
Essential	(Additional Constructions) Data Tables
Essential	(Additional Constructions) Scenario Outline
Focused	(Concise) Focused
Focused	(Testable) Focused
Focused	(Language) Declarative rather than Imperative
Focused	(Additional Characteristics) Declarative vs Imperative
Singular	(Small) Test One Single Thing
Singular	(Testable) Clear Outcomes and Verifications
Singular	(Testable) Clear and Simple Given's
Singular	(Unambiguous) Single Clear Intention
Singular	(Unambiguous) Scenarios Testing the Same Thing
Clear	(Language) Technical Jargon
Clear	(Concise) Clear
Clear	(Unambiguous) Vague Statements
Clear	(Unambiguous) High Granularity Steps Descriptions
Complete	(Testable) Follow the Steps
Complete	(Testable) Completeness
Complete	(Understandable) Self Contained
Complete	(Unambiguous) Completeness
Complete	(Additional Characteristics) Complete
Unique	(Valuable) Unique
Unique	(Valuable) Business Value
Unique	(Additional Characteristics) Cohesion
Unique	(Titles) Feature Description
Unique	(Titles) Scenario Titles
Unique	(Additional Constructions) Tags
Ubiquitous	(Understandable) Ubiquitous
Ubiquitous	(Language) Ubiquitous
Ubiquitous	(Language) Actor Consistency
Ubiquitous	(Unambiguous) Ubiquitous
Ubiquitous	(Additional Characteristics) Writing Patterns
Integrous	(Understandable) Good English
Integrous	(Additional Characteristics) Integrity
Integrous	(Steps) Steps Order

Table 4.5 – Summary of the newly defined quality attributes

Attribute	**BDD scenarios should...**
Essential	... avoid unnecessary details and steps
Focused	... declare "what" is being done rather than "how"
Singular	... formalize a single intention
Clear	... be correctly understood by all parties involved
Complete	... have all the information needed to cover all feature
Unique	... be fundamentally different to the other scenarios
Ubiquitous	... use business terms and roles in a consistent way
Integrous	... respect the rules of Gherkin language

Some of the participants opinions had to be left aside. Due to our focus on textual scenario's descriptions only, the interpretation of small as in a *small code for that step* did not hold a position in our groups. In the same way, the missing characteristic of independent scenarios, that would allow the evaluation of how a scenario execution could affect the other, was left out of our list. Neither are other automation characteristics described on prior sections.

Additionally, the interpretation that an understandable scenario is one where *technical people understand what to do* conflicts with our notion about how BDD scenarios should bring together business and technical roles and serve as a *standard documentation source* (P18) – therefore, our groups do not cover that as well.

4.4.1 Essential

The *essential* attribute represents the fact that only essential information should be written into textual BDD scenarios. It is derived from some of the interpretations of concise, such as the avoidance of unnecessary details and unnecessary lines. The use of unnecessary lines would make more steps to be written, and therefore the "not many steps" interpretation, that was assigned to concise and small attributes, also falls into this newly-defined attribute.

The steps' characteristics "unnecessary lines" and "many steps" reinforces that belief – both have affected either concise or small attributes badly. One practice to avoid pre-conditions repetition is the use of a background section – a characteristic that aid on concise attribute and therefore would also aid to the newly-defined essential attribute.

Other good practices are to either summarize steps into a data table, short and with meaningful titles, or to group similar scenarios together using a scenario outline. Both practices have benefits and potential problems, as already mentioned in the corresponding previous sections, but they seem to positively affect the essential attribute idea as all of them had positively affected concise attribute according to some participants.

4.4.2 Focused

The *focused* attribute represents the need of declaring "what" a scenario should do (writing it in a declarative way), rather than describing "how" that action will be performed (writing it in an imperative way). It comes directly from one of the interpretations of concise and testable attributes in Table 4.4, that helped name this newly-defined attribute.

It can also be understood by looking from the "declarative rather than imperative" language characteristic's perspective, that some participants (P1, P5, P8, P16, P17) declared to affect either concise or testable literature attributes. Strangely enough, that criteria had also affected understandable attribute in a similar degree, but no participant had said that understandable BDD scenarios should be more focused.

Additionally, "declarative rather than imperative" also appeared as a missing quality attribute in the list used in the interviews, reinforcing the idea of having this characteristic as an exclusive attribute.

4.4.3 Singular

The *singular* attribute represents the need of a scenario to have a single purpose and to demonstrate this purpose clearly. The fact that BDD scenarios should have a "single intention" was an interpretation mentioned for unambiguous attribute that is similar to the "test one thing" interpretation of the small attribute in Table 4.4. Another interpretation of the unambiguous attribute was also that, sometimes, scenarios written differently were "testing the same thing".

Therefore, we believe that the singular intention should be an attribute evaluated in separation of the others. Clear outcomes (assigned to verifications on Then steps) and pre-conditions (assigned to Given steps), both mentioned on testable attribute, would also be interpretations tied to this attribute.

4.4.4 Clear

The *clear* attribute appeared due to the fact that vague statements can harm as much as an excess of details. Additionally, high granularity steps, an interpretation of unambiguous attribute in Table 4.4, could be easily identified as vague steps for technical people. However, the use of technical jargon, a criteria that harms understandable attribute, could represent less clear steps for business people.

Therefore, there should be a certain balance that allows a scenario to be correctly understood by all parties involved. The word clear appeared as an interpretation of the concise attribute in Table 4.4 that would also reinforce the before mentioned unambiguous' interpretations.

4.4.5 Complete

The *complete* attribute, also an interpretation from testable and unambiguous literature attributes, can be seen from multiple perspectives. On the scenario level, all the information needed to understand and follow those steps should be present – represented by the "follow the steps" and the "self contained" characteristics in Table 4.4. On the feature level, the set of scenario's should provide enough coverage for that feature – a missing attribute on the interviews list mentioned in the early subsections.

4.4.6 Unique

The *unique* attribute can be summarized by the quote *is it testing something fundamentally different to the other scenarios* from P15, stating that a scenario's business value should be evident from its description and from the fact it is interesting. That interpretation came directly from the valuable attribute's interpretation in Table 4.4. Even tough we could keep using the valuable word to describe this characteristic, uniqueness seems to be more suited to represent it.

To aid on that assessment, each scenario title should inform the reader about its behavior, preferably expressing its action and its outcome while correlating those with the actual steps. Also, feature file descriptions, often in the user story format, can aid on recognizing a set of scenarios importance. Tags, serving as the scenario's meta-data, can provide important information to express the scenario context.

4.4.7 Ubiquitous

The *ubiquitous* attribute represents the use of business terms, either taken from a glossary or a team's common knowledge, and helps to reinforce the need to bring scenarios closer to technical and business people alike. Ubiquitous, also a word that appeared as an interpretation of understandable and unambiguous attributes, seems to be the correct word to represent that common vocabulary.

The criteria of using business defined roles and/or personas in a consistent way enhances the ubiquity of a scenario description, as some participants (P2, P6, P7, P17) argue that it positively affect either understandable or unambiguous attributes. Also, that consistency need matched the need of a writing pattern, highlighted as a missing attribute in Table 4.4.

4.4.8 Integrous

The *integrous* attribute remind us that a scenario should respect the rules of Gherkin language, as the natural sequence of Given/When/Then steps. Also, it should use proper steps tenses (Given steps in the past, When steps in the present, Then steps in the future), and it should respect each keyword type (Given describing pre-conditions, When describing actions and Then describing verifications) as described on both the understandable and the missing integrity attribute in Table 4.4. Finally, as pointed out on another missing attribute on the interviews list, it's interesting to keep a certain writing pattern, to avoid having different styles of descriptions appearing on different scenarios written by different people.

5. PROPOSED QUESTION-BASED CHECKLIST

This chapter takes the newly defined attributes from Section 4.4 and transform them into a question-based checklist in Table 5.1, as follows, that we believe could be suited to evaluate BDD scenarios – thus, answering our RQ2. The output of those questions are meant to help the evaluator decide how good her BDD scenarios are and, if not satisfactory, may be the input of additional conversations around those scenarios.

The question-based checklist is aimed to evaluate a feature file, a group of BDD scenarios, similar to the ones on Diaspora project that were evaluated by the participants. Using Shull et. al's [33] definition of a reading technique, the questions on our question-based checklist are organized per scope with the purpose to guide the reader on using the questions – first on the feature file level, than on scenario level and then on steps level. That concrete procedure, added to the list of questions, allow us to characterize our question-based checklist as a reading technique.

This scope decision came from the fact that no participant was inclined to analyze a scenario per time – all of them read the feature file rapidly before focusing in one scenario per time. Unfortunately, none of our interview questions focused on the scope of their evaluation, to understand what was their rationale in doing so, so we assume the participants natural behavior is a good enough guidance for us.

Questions 1 and 2 are suited for the feature level, while Questions 3 to 8 scopes are on the scenario level, and therefore should be used to analyze each scenario at once, and Questions 9 to 12 scopes are on the steps level, and therefore should be used to analyze each step at a time. We recommend the evaluator of the quality of BDD scenarios to answer Questions 1 and 2 by looking at the bigger picture first – the feature file as a whole, then proceeding to analyze every scenario on it with Questions 3 to 12.

Question 1 focuses the analysis on the feature description, so the evaluator can check if the intended business outcome or business value is expressed in that area – a characteristic of the unique attribute. It can be represented in an user story format, like the one found in Figure 4.22, as recommended by some participants (P11, P15, P17), but this is not enforced on BDD technique nor on this question-based checklist as the real purpose of that description is to make evident to the evaluator the business outcome or value of that feature – a trait P8 has not seen in that figure. Figure 4.16 also shows a feature title that could be re-think – P4 said it to lack completeness, but Question 1 could certainly be the one to reason about that matter.

Question 2 focus is also on the feature level, but it aims to have the evaluator question how completely the BDD scenarios cover that feature. The question purpose is to motivate the evaluator on having a glimpse of all the scenarios in order to check what are they testing, in the hope that the evaluator can identify missing BDD scenarios that should be part of that feature file coverage using his own knowledge about the system. For example, thoughts such as the ones shown by P4 when analyzing the feature file in Figure 4.16 are advisable, as he questions: there is no thing in this feature if Robert deletes a profile photo, so can Alice continue to see it?

Table 5.1 – Question-based checklist for BDD scenarios

ID	Question	Scope	Attribute
1	The feature file business value or outcome can be identified by its description?	Feature	Unique
2	The feature file have any missing scenarios?	Feature	Complete
3	The scenario carries all the information needed to understand it?	Scenario	Complete
4	The scenario have steps that can be removed without affecting its understanding?	Scenario	Essential
5	How different each scenario is from the others?	Scenario	Unique
6	The scenario single action can be identified on its title and match what the scenario is doing?	Scenario	Singular
7	The scenario outcome or verifications can be identified on its title and match what the scenario is doing?	Scenario	Singular
8	This scenario respects Gherkin keywords meaning and its natural order?	Scenario	Integrous
9	Does that step correctly employs business terms, including a proper actor?	Step	Ubiquitous
10	Does that step have details that can be removed without affecting its meaning?	Step	Essential
11	Does that step express "what" it is doing by being written in a declarative way?	Step	Focused
12	Does the step allow different interpretations by being vague or misleading?	Step	Clear

Question 3 aims to avoid scenarios that need additional information to be understood. One evaluator should be able to "follow the steps", as the scenario is "self contained" enough to allow anyone from the team to do that, as in the scenario in Figure 4.10. However, scenarios are not meant to be as complete as test cases, so a certain balance has to be achieved.

Therefore, Question 4 targets unnecessary steps, that adds more information than what is essential to validate the behavior being tested, like on the scenario found in Figure 4.8. This question also intends to evaluate how additional constructions available on the Gherkin language were used, raising questions such as: is the link between the background section and every scenario strong enough? can a background be generated to avoid steps repetition, such as in Figure 4.4? is the use of data tables necessary, such as in Figure 4.3? is the use of scenario outline and example tables necessary?

Similarly, Question 5 aims to question the need of a scenario in that feature file. The answer to that question may well come from the completeness analysis on Question 2, but here the focus is on how different a scenario is from the others in the sense of the business value this scenario bring to the whole feature file. The evaluator may even wonder why this scenario is being tested at this acceptance test level and not under unit test or integration test levels, or even why this needs to be automated at all.

Question 6 and 7 are meant to evaluate the relationship between what is advertised in the scenario title and the actual steps' descriptions. In Question 6, the evaluator is asked to focus on the scenario single action, represented on the When step. The evaluator should question if the action is represented in a single step (and if not, why not) and if the action writing is aligned with the scenario title. In an analogous way, in Question 7 the evaluator is asked to focus on the scenario verifications and outcomes, found on the Then steps, and in how aligned they are with the purpose expressed in the title. Ideas of a better writing strategy, such as the one in Figure 4.23, can be raised when the evaluator is thinking on those questions

Question 8 validates the integrity of the Gherkin rules on every scenario, allowing the evaluator to question if Given steps are really representing pre-conditions, if When steps are representing actions and if Then steps are representing outcomes. If necessary, the evaluator can question the statements tenses as well. Also, this question protects against violations on the order of those keywords, such as the violation shown in Figure 4.20.

Question 9 assumes that all the steps are using correct business terms in a consistently way, empowering scenarios with an ubiquitous language that technical and non-technical people could understand. Examples such as in Figure 4.12 are not recommended. Additionally, the actor performing every step action should represent a business role rather than "I" or "the user".

Question 10, similar to Question 4, aims to reduce unnecessary information, but on the step level rather than on the scenario level. If some details are important to be added to represent how different the scenario data is from others, maybe a data table would be needed, like in Figure 4.3. Additionally, if a data table is present, the evaluator should question what is the need of it to understand the step description.

Question 11 brings the evaluator to question how focused a step is on the "what" is being done rather than in expressing "how" it is doing that. If the step is not written in a declarative way, such as in Figure 4.5, then the evaluator should question it now.

Finally, Question 12 aims to balance the need to express actions with fewer words and details with the need to make the scenario understandable for every team member involved. If a step is too abstract or vague that makes the evaluator confused about what would be the actual user flow to perform that step, than it needs to be clearer than it is.

6. FINAL CONSIDERATIONS

This chapter elaborates a summary of the results found, revisiting the research question. Later, it presents the publications we performed to acquire early feedback on our work, limitations of the current document and suggested future research ideas.

6.1 Summary of Results

With the 18 interviews performed, we have gathered enough data to define 8 newly-labeled quality attributes in Chapter 4.4, suited to evaluate BDD scenarios and summarized in Table 4.5. This list of attributes answers our RQ1 about *What are the quality attributes suited to describe a "good" BDD scenario by the view of different members of software development team?*

Due to the deep understanding about how those newly-defined attributes were assembled, we could also represent them into a question-based checklist format, similar to the one used by Cockburn [4] and represented in Figure 2.2 to evaluate use cases. Our question-based checklist is summarized in Table 5.1 and presented in Chapter 5, along with a reading strategy to use it – to start evaluating the feature file as a whole (answering Questions 1 and 2), then proceeding to focus on each scenario (answering questions 3 to 8) and on each step (questions 9 to 12).

6.2 Lessons Learned

Since the beginning, we were guided by a certain theory on how requirements are evaluated, in the form of our literature-informed attributes. These attributes would ground our findings into a known and accepted terminology and would be mapped to practitioner's criteria – each attribute to a single personal evaluation criteria specific to BDD scenarios, a thought that motivated the question 13 in our interview questionnaire in Table 3.2.

In the end, these attributes aided practitioners to phrase their evaluation criteria, often abstract and subjective, into proper words. The contemplation about the meaning of those words into BDD scenarios alone was often informally praised by practitioners as a nice experience. However, as explained in the beginning of Chapter 4, we could not map each attribute uniquely to a certain personal criteria due to the many interpretations of each attribute. That impossibility reduced the need of the answers to question 13 in our interview questionnaire in Table 3.2, which were used only to aid in the grouping of interpretations and criteria that lead to our 8 newly-labeled quality attributes.

Without these literature-informed attributes, maybe some evaluation criteria would have been forgotten by the practitioner's which would make it harder to join their spoken criteria into a proper model. However, the suggestion that BDD scenarios could be evaluated with a set of criteria

had lead to some debate into the Cucumber community[1], as some see more value in evaluating the whole BDD process (and conversations) or not to evaluate them at all. We fear our use literature-informed attributes to guide our interviews may have motivated some practitioner's to dismiss our invite to be interviewed, so we suggest future researchers to be more attentive on how practitioner's could react to certain research design decisions.

Additionally, the idea of using Diaspora's scenarios has been proposed only to comply with the ideas underlying the questions used by Heck and Zaidman [15]. In the end, those convenient scenarios were an excellent media to channel practitioner's critics and allow the emergence of suggestions to improvements, which in turn lead the interviewer to better understand practictioner's evaluation criteria and what they valued in BDD scenarios. We strongly suggest similar efforts to use requirements from open source real projects in future studies.

6.3 Publications

In order to collect early feedback about our research design decisions, explained in Chapter 3, we reported the pilot study with 15 novice students in the 2017 Workshop on Empirical Requirements Engineering (EmpiRE'17) in conjunction with the International Requirements Engineering Conference under the title *On the Empirical Evaluation of BDD Scenarios Quality: Preliminary Findings of an Empirical Study* [26].

Additionally, in order to gather early feedback on how to present our findings from the interviews, explained in Chapter 4, we reported the insights taken from 8 interviews in the 2018 International Working Conference on Requirements Engineering (REFSQ'18) under the title *On the Understanding of BDD Scenarios' Quality: Preliminary Practitioners' Opinions* [27].

6.4 Limitations

Due to our focus on requirements engineering, we explicitly focused on the business facing part of BDD scenarios, as shown in Figure 2.5. Therefore, the most explicit limitation of our newly-defined attributes and question-based checklist is the fact that they may not be suited to review the automation code required to allow BDD scenario's steps to validate the product and work as a executable specification as envisioned by Adzic [1].

Additionally, we only focused on the formalization part of a typical BDD process, as seen in Figure 2.6. How to evaluate conversations and examples taken from those, or even how to better construct those examples to generate good BDD scenarios, was out of our scope.

Furthermore, a study threat to validity is the fact that only the main researcher was involved into the coding process may have impacted the themes creation in an unforeseen way, even with the careful review of the advisor. Also, we have not taken into consideration the gender, role, location

[1] https://groups.google.com/forum/#!msg/cukes/wfjhwURkt4E/95f9ZOmEAQAJ;context-place=forum/cukes

nor the type of industry a participant works into to reflect upon the data taken from the interviews. Therefore, we acknowledge that different contexts may need to use different quality criteria and thus a different question-based checklist. We tried to mitigate this effect interviewing people from many different areas and companies, but acquiring people from very different backgrounds was not out main goal when selecting participants for the interviews.

Finally, we could have phrased the items in our question-based checklist in many different ways, so we have to acknowledge that our own language bias may have driven us to write them in that way presented in Chapter 5.

6.5 Future Work

The findings of this dissertation, its contributions, and limitations, indicate several possibilities for further research, as follows:

1. Further research considering good and bad automation coding practices used with BDD scenarios. As this side of BDD scenarios is not covered by our research, the question on *how to create "good" BDD scenarios automation code* may yield additional quality criteria that could enhance ours;

2. Further research considering good and bad conversations practices impact BDD scenarios. As these serve as inputs for BDD scenarios, the question on *how to have conversations to generate "good" examples* may yield additional quality criteria that could enhance ours;

3. Further empirical research to validate our question-based checklist and newly-defined attributes, validating how useful it can be to practitioners is envisioned by us. We suggest an additional care to select practitioners based on different experience level and contexts in order to also mitigate our threat to validity.

There are certainly other roads to future research that we have not foreseen. Due to the fact this is the first work on the topic of requirements quality in BDD scenarios, our focus was strictly on the requirements' side of BDD scenarios. However, BDD scenarios are the formalization of examples taken from conversations among the development and client teams and are most often written alongside their automation code – so there is still much to explore in order to qualify how good BDD writing practices are.

BIBLIOGRAPHY

[1] Adzic, G. "Specification by Example: How Successful Teams Deliver the Right Software". Greenwich, USA: Manning Publications Co., 2011, 1st ed., 296p.

[2] Alexander, I. F.; Maiden, N. "Scenarios,Stories, Use Cases: Through the Systems Development Life-Cycle". Hoboken, USA: Wiley Publishing, 2004, 1st ed., 548p.

[3] Bjarnason, E.; Unterkalmsteiner, M.; Borg, M.; Engström, E. "A multi-case study of agile requirements engineering and the use of test cases as requirements", *Information and Software Technology*, vol. 77, Sep 2016, pp. 61–79.

[4] Cockburn, A. "Writing Effective Use Cases". Boston, USA: Addison-Wesley Professional, 2000, 1st ed., 304p.

[5] Cohn, M. "User Stories Applied: For Agile Software Development". Redwood City, USA: Addison Wesley Longman Publishing Co., Inc., 2004, 1st ed., 304p.

[6] Creswell, J. W. "Research Design: Qualitative, Quantitative, and Mixed Methods Approaches". Thousands Oaks, USA: Sage Publications Ltd., 2009, 3rd ed., 260p.

[7] Cruzes, D. S.; Dyba, T. "Recommended steps for thematic synthesis in software engineering". In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement, 2011, pp. 275–284.

[8] Davis, A.; Overmyer, S.; Jordan, K.; Caruso, J.; Dandashi, F.; Dinh, A.; Kincaid, G.; Ledeboer, G.; Reynolds, P.; Sitaram, P.; Ta, A.; Theofanos, M. "Identifying and measuring quality in a software requirements specification". In: Proceedings of the International Software Metrics Symposium, 1993, pp. 141–152.

[9] ElAttar, M.; Miller, J. "Constructing high quality use case models: a systematic review of current practices", *Requirements Engineering*, vol. 17, Sep 2012, pp. 187–201.

[10] Fernández, D. M.; Wagner, S.; Kalinowski, M.; Felderer, M.; Mafra, P.; Vetrò, A.; Conte, T.; Christiansson, M.-T.; Greer, D.; Lassenius, C.; Männistö, T.; Nayabi, M.; Oivo, M.; Penzenstadler, B.; Pfahl, D.; Prikladnicki, R.; Ruhe, G.; Schekelmann, A.; Sen, S.; Spinola, R.; Tuzcu, A.; de la Vara, J. L.; Wieringa, R. "Naming the pain in requirements engineering", *Empirical Software Engineering*, vol. 22–5, Oct 2017, pp. 2298–2338.

[11] Gartner, M. "ATDD by Example: A Practical Guide to Acceptance Test-Driven Development". Boston, USA: Addison-Wesley Professional, 2012, 1st ed., 240p.

[12] Génova, G.; Fuentes, J. M.; Llorens, J.; Hurtado, O.; Moreno, V. "A framework to measure and improve the quality of textual requirements", *Requirements Engineering*, vol. 18–1, Mar 2013, pp. 25–41.

[13] Gojko, A. "Impact Mapping: Making a big impact with software products and projects". Woking, UK: Provoking Thoughts, 2012, 1st ed., pp 86p.

[14] Haugset, B.; Stalhane, T. "Automated acceptance testing as an agile requirements engineering practice". In: Proceedings of the International Conference on System Sciences, 2012, pp. 5289–5298.

[15] Heck, P.; Zaidman, A. "A framework for quality assessment of just-in-time requirements: the case of open source feature requests", *Requirements Engineering*, vol. 22–4, Nov 2017, pp. 453–473.

[16] Heikkila, V. T.; Damian, D.; Lassenius, C.; Paasivaara, M. "A mapping study on requirements engineering in agile software development". In: Proceedings of the Euromicro Conference on Software Engineering and Advanced Applications, 2015, pp. 199–207.

[17] IIBA. "A Guide to the Business Analysis Body of Knowledge". Toronto, Canada: International Institute of Business Analysis, 2009, 2nd ed., 264p.

[18] IIBA. "A Guide to the Business Analysis Body of Knowledge". Toronto, Canada: International Institute of Business Analysis, 2015, 3rd ed., 512p.

[19] Jeffries, R. "Essential xp: Card, conversation, confirmation". Retrieved from: http://ronjeffries.com/xprog/articles/expcardconversationconfirmation/, Visited in: Mar 2018, 2001.

[20] Jeffries, R. E.; Anderson, A.; Hendrickson, C. "Extreme Programming Installed". Boston, USA: Addison-Wesley Professional, 2000, 1st ed., 288p.

[21] Kaner, C. "The power of 'what if. . . ' and nine ways to fuel your imagination", *Software Testing and Quality Engineering magazine*, vol. 05, Sep 2003, pp. 5.

[22] Kent Beck, M. F. "Planning Extreme Programming". Boston, USA: Addison-Wesley Professional, 2000, 1st ed., 160p.

[23] Laitenberger, O. "A survey of software inspection technologies". In: *Handbook on Software Engineering and Knowledge*, 2nd ed., River Edge, USA: World Scientific, 2002, pp. 20.

[24] Lucassen, G.; Dalpiaz, F.; VanDerWerf, J.; Brinkkemper, S. "Forging high-quality user stories: Towards a discipline for agile requirements". In: Proceedings of the International Requirements Engineering Conference, 2015, pp. 126–135.

[25] Melo, W.; Shull, F.; Travassos, G. H. "Software review guidelines", Technical Report, COPPE/UFRJ, 2001, 21p.

[26] Oliveira, G.; Marczak, S. "On the empirical evaluation of bdd scenarios quality: Preliminary findings of an empirical study". In: Proceedings of the Workshop on Empirical Requirements Engineering in conjunction with the International Requirements Engineering Conference, 2017, pp. 4.

[27] Oliveira, G.; Marczak, S. "On the understanding of bdd scenarios' quality: Preliminary practitioners' opinions". In: Proceedings of the International Working Conference on Requirements Engineering, 2018, pp. 7.

[28] Oliveira, G.; Marczak, S. "Quality aspects of agile requirements", Technical Report, PUC-RS, 2018, 26p.

[29] Patton, J. "User Story Mapping: Discover the Whole Story, Build the Right Product". Farnham, England: O'Reilly Media, 2014, 1st ed., 324p.

[30] Phalp, K.; Adlem, A.; Jeary, S.; Vincent, J.; Kanyaru, J. M. "The role of comprehension in requirements and implications for use case descriptions", *Software Quality Journal*, Dec 2011, pp. 461–486.

[31] Pressman, R. "Software Engineering: A Practitioner's Approach". New York, USA: McGraw-Hill, 2009, 7th ed., 928p.

[32] Rinzler, B. "Telling Stories: A Short Path to Writing Better Software Requirements". Hoboken, USA: John Wiley and Sons, 2009, 160p.

[33] Shull, F.; Carver, J.; Travassos, G. H.; Maldonado, J. C.; Conradi, R.; Basili, V. R. "Replicated studies: Building a body of knowledge about software reading techniques", *Lecture Notes on Empirical Software Engineering*, 2003, pp. 42.

[34] Smart, J. "BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle". Shelter Island, USA: Manning Publications, 2014, 1st ed., 384p.

[35] Smart, J. F. "Broad brushes and narrow brushes: there's more to bdd than given/when/then". Retrieved from: https://johnfergusonsmart.com/theres-bdd-givenwhenthen/, Visited in: Dec 2017, 2017.

[36] Williams, C.; Kaplan, M.; Klinger, T.; Paradkar, A. "Toward engineered, useful use cases", *Journal of Object Technology*, Jun 2005, pp. 45–57.

[37] Wynne, M. "Introducing example mapping". Retrieved from: https://cucumber.io/blog/2015/12/08/example-mapping-introduction, Visited in: Jan 2018, 2015.

[38] Wynne, M.; Hellesoy, A. "The Cucumber Book: Behaviour-Driven Development for Testers and Developers". Raleigh, USA: Pragmatic Bookshelf, 2012, 1st ed., 336p.

[39] Zhu, Y.-M. "Software Reading Techniques: Twenty Techniques for More Effective Software Review and Inspection". Solon, USA: Apress, 2016, 1st ed., 126p.

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br