Improving Quality of Use-Case Models by Correlating Defects, Difficulties, and Modeling Strategies

Cristiana Pereira Bispo[1,3](✉), Ana Patrícia Magalhães[1,2], Sergio Fernandes[1], and Ivan Machado[3]

[1] Salvador University, Salvador, Brazil
cristiana.bispo@ufba.br, {ana.fontes, sergio.fernandes}@unifacs.br
[2] Department of Exact Sciences and Earth, State University of Bahia, Salvador, Brazil
[3] Computer Science Department, Federal University of Bahia, Salvador, Brazil
ivan.machado@ufba.br

Abstract. Use case (UC) models play an essential role in software specification since they describe system functional requirements. A UC model should be free of defects due to its relevance and impact throughout the software development life cycle. However, inspections in UC models frequently identify defects related to modelers' difficulties in different activities during the modeling process. The quality of a UC model is usually analyzed based on quality criteria such as ambiguity and inconsistency. Several strategies in the literature assist use case modeling in mitigating defects, but these strategies do not identify which potential defects they aim to prevent or eliminate. In this context, we proposed a correlation between UC modeling difficulties and strategies to mitigate these difficulties based on UC's quality criteria. In this paper, we describe each strategy contained in the correlation and present, in detail, the controlled experiment that assesses the correlation effectiveness, including the discrete data collected in analyzing the participants' models and statistical analysis performed in these data. Besides, we also propose a mechanism to guide in elaborating checklists to identify defects in UC models focusing on quality criteria. Through a controlled experiment, we evaluate the Antipattern strategy, and the results showed a clear indication that this strategy mitigates the difficulties in which it is related according to the correlation. Besides, the UC models developed in the experiment were evaluated using the checklist generated based on the proposed mechanism.

Keywords: Use case modeling · Use case inspection · Controlled experiment

1 Introduction

In Information System (IS) development, a technique commonly used to capture and describe systems' functional requirements is Use-Case Modeling (UCM), which produces a Use-Case (UC) model as output. Due to their relevance and impact throughout the software development life cycle, UC models should be free from defects.

J. Filipe et al. (Eds.): ICEIS 2020, LNBIP 417, pp. 360–385, 2021.
https://doi.org/10.1007/978-3-030-75418-1_17

The quality of a UC model can be analyzed based on quality attributes already defined in the literature, as summarized by Tiwari and Gupta [2], such as ambiguity, inconsistency, among others. In fact, several authors have reported that the inspection of use-case models concerning quality attributes frequently identifies the occurrence of defects [2–6]. The low quality of UC models has been related to modelers' difficulties in different activities, such as understanding the requirements and representing them in UC models [7]; and understanding the problem domain [5].

To mitigate defects in UC models, several authors propose strategies that help UCM. In a prior work, we carried out a systematic literature review (SLR) [26] and identified 39 studies that present UCM strategies. For example, some authors indicate the use of Antipattern [3, 9–13], a technique that brings evidence on common modeling mistakes so that developers do not make the same mistakes in their models. Business Process Notation, a standard notation to model business process, is also considered as a feasible strategy to identify requirements [14, 15]. The authors of these works agree that using a UCM strategy can lead to an improvement in the UC model concerning the quality attributes enhanced by the strategy. However, none of these works identifies which potential defects each of these strategies aims to prevent or eliminate. Nor do they identify which difficulties of the modelers the strategy can alleviate.

UCM strategies do not have a systematic method to inspect UC models built using the respective strategy, i.e., do not provide a formalism to assist in identifying defects in UC models. Additionally, they do not generate a list of identified defects that can be reused to inspect other use-case models in the same domain. In summary, it is challenging to assess a UC model's quality by the occurrence of defects because the works found in the literature do not show how the UC models can be corrected.

To assist UCM, we proposed a correlation between UCM difficulties and strategies to mitigate these difficulties based on quality attributes for UC [8]. With this correlation's support, it is possible to identify which strategies are most appropriate to mitigate the defects that affect quality specific attributes. This current investigation elaborates on such preceding work by detailing each literature strategy to support UCM. We also detail the controlled experiment that assesses the correlation effectiveness, including the discrete data collected in analyzing the participants' models and statistical analysis performed in these data. To assist in the inspection of the UCs created based on a strategy, in this article, we also propose a mechanism to guide in elaborating checklists to identify defects in UCMs with a focus on quality criteria. The mechanism comprises a set of steps for the generation of the checklist to inspect UC models. We validated the mechanism in the same experiment to assess the correlation.

The remainder of this chapter is organized as follows: Sect. 2 presents the concepts related to requirements specification and use-case modeling. Section 3 discusses related work. Section 4 details the strategy found in the literature to assist UCM. Section 5 presents the correlation between UCM difficulties and strategies to mitigate these difficulties, and Sect. 6 presents the proposed mechanism to generate a UC inspection checklist. Section 7 details the controlled experiment, and Sect. 8 draws concluding remarks and pinpoints opportunities for further research.

2 Requirements Specification

According to Sommerville [17], a system requirement represents the description of functionality or constraint that the system must fulfill. The process of identifying, analyzing, documenting, and verifying these requirements is called Requirement Engineering. It produces a Software Requirement Specification (SRS), which is an essential input for subsequent software development activities. Essential activity in the Requirement Engineering process are requirements identification, analysis, and validation. There are different techniques in the literature to specify requirements, among them the UC modeling.

UC modeling is the activity of designing a use-case model, which describes in detail the software functional requirements. They make use of graphic and textual notation to, respectively [18] (i) create the UC diagram that provides a visual summary of the system services and their interaction with the environment and users (called actors); and (ii) describe the interactions between the system and its actors.

The Unified Modeling Language (UML) [19] is widely adopted to represent use case diagrams. The UML UC diagram is mainly composed of the following elements: actors, use case, communication relationships, inclusion, extension, and generalization. The description of Use Cases is a textual notation, in natural language, that describes the behavior of each UC in the diagram. Cockburn [20] and Jacobson et al. [21] define the following elements to describe a UC: Name of the UC; Short description of the UC's objective; Actor (s) participating in the UC; Precondition to start the UC; Post-condition that must be met after the execution of the UC; Main Flow describing the main UC scenario; Alternative flows with alternative UC usage scenarios; Exception flows for unexpected occurrences; and Rules that must be considered when executing the UC.

2.1 Difficulties in UCM

Difficulties regarding the syntax and semantics of graphic and textual elements in elaborating the UC model compromise the quality attributes [22] of use case models, such as completeness, ambiguity, and inconsistency. In this paper, a difficulty is defined as any lack of knowledge of requirement modelers that prevent them from modeling UCs meeting specified quality requirements.

There are some works in the literature that present difficulties in UCM, such as difficulties to understand requirements and represent them in use cases [22]; difficulties in understanding the problem domain [5]; difficulties in specifying information unambiguously [23]; difficulties in representing information in a diagram [24]; among others. Nascimento et al. [5] summarize all of these in a model of difficulties in UCM.

Writing UC is an exploratory and visionary task, and a good modeler should have skills, such as the ability to write well; the ability to systematically address a problem; the ability to synthesize user needs; in addition to specific knowledge of the problem domain and understanding of software development, among other skills.

2.2 Rules and Guidelines for UCM Modeling

There are several guidelines and rules in the literature to writing effective UC [19, 21], and [25]. The work of Gregolin [45] presents a synthesis of these rules and guidelines. Some of these are:

- A UC should add value to the related actors, grouping atomic functions in a single functionality, avoiding functional decomposition;
- A UC should avoid individual features of CRUD (Create, Retrieve, Update, Delete). These functionalities should have a single generic UC, such as *Manage Registrations*, or *Keep Registrations*, and to use interface prototypes for each registration.
- The actor's name should reflect their role in the system and avoid titles of positions, organizations, or activities related to an organizational structure.
- The UC's name usually contains a verb followed by nouns, and the sub-nouns can have adjectives. The verb must be in the infinitive or present tense and must use the active voice instead of the passive voice, among others.

2.3 Quality Attributes in UCM

There are many recommendations in the literature on what constitutes quality in a use-case model. Through a Systematic Literature review (SLR), the authors of [2] bring these attributes together. However, different authors often give each attribute a particular understanding. Therefore, in this work, we perform a synthesis of each of them, as Table 1 shows.

Table 1. Synthesis of quality attributes.

Quality attribute	Description
Accuracy or completeness or integrity	There should be no missing information nor elements in the UC diagram and in the corresponding textual descriptions
Consistency	The UC model information should have the expected semantics. There should not be any conflicting elements in the diagrams and in their textual descriptions
Correctness	The UC diagram and its descriptions must correctly represent the requirements
Understandability	The information and rules contained in the UC diagrams and textual descriptions must be accurate and clearly defined
Ambiguity	There should be no information in the UC diagram and textual descriptions with more than one meaning
Redundancy	There should be no excessive, repetitive or superfluous information in the UC diagram and descriptions
Abstraction level	The UC diagram and descriptions should present only what the software should do at an appropriate granularity level. That is, the UC should not be broken down into parts that have no value in themselves

3 Related Work

The studies deemed as related to the purpose of this research focus on the same aspect: the difficulties of UCM requirements specifier that prevent them from building UC models meeting the defined quality requirements.

Nascimento et al. [5] sought to explore and understand difficulties in UCM by conducting four experimental studies. As a result, they presented a model of difficulties. The works [22, 23] and [24] also investigated and reported difficulties in UCM. These works do not present any strategy to mitigate these difficulties.

To mitigate the difficulties that requirements specifier faces when modeling UCs, several authors propose applying resources already used in other domains to verify their effectiveness in UCM. The work presented in [14] employed business process models to derive UCs because these models are often available in a company as work instructions or administrative manuals in a clear and structured manner. Conversely, the authors of [3] presented an Antipattern-based strategy for UCM, in which bad practices are identified to be replaced by recommended solutions. We identified other strategies in a previous investigation and their respective contributions to UCM [8]. However, these studies do not indicate which strategies could mitigate the UCM difficulties.

The difficulty-strategy correlation proposed in this paper guides the requirements specifier in selecting the most appropriate strategy to mitigate a given difficulty. It avoids adopting ineffective practices and presents various alternatives for applying tested and evaluated procedures to assist UCM.

4 Strategies for UCM

Several strategies in the literature support the use-case modeling, as identified in the SLR that we present in [26].

One of the first strategies proposed in the literature was that of **Ontology** [27–33]. It represents the domain concepts and their relationships, allowing automated reasoning. Consequently, it can minimize problems concerning requirements ambiguity, inconsistency, and incompleteness. Ontology strategy contributes to UCM, making it possible to specify UC requirements more completely and unambiguously.

Antipattern is one of the most common strategies found in the literature [3, 9–13]. It focuses on identifying deficiencies in UC, emphasizing the human cognitive abilities that allow identifying these deficiencies. According to it, a *bad* UC modeling structure does not necessarily indicate a defect, but it can lead to possible harmful consequences. This way, it shows dubious UCM structures and their harmful effects and is useful in detecting possible doubtful structures to perform corrective actions.

Another technique, called **Role Interpretation** [34–39], is used in teaching modeling. Students and instructors assume different roles in a modeling experiment. In this way, it aims to simulate the industrial environment so that students obtain a description very close to the system to be modeled.

The **Natural Language Processing** (NPL) strategy provides semi-automated assistance, through an algorithmic approach, for developers to generate UC models from standardized natural language requirements [46–50]. It inspects the requirements document to find nouns to add to the list of actors, and verbs to add to the list of UCs. The UC diagram is sketched from these lists. NPL protects the developer from the ambiguity, redundancy, and incompleteness inherent in the requirements specifications written in natural language. It uses advanced word processing techniques to rapid discover duplicate functionality, identify and extract actors and UCs, and to clarity specifications that are difficult to understand and communicate.

Scenario Pattern [51–54] uses standard UC specification scenarios that describe requirements and interactions between system actors in a standardized way. Pattern matching algorithms are automatically used to check if there is an omission of any necessary step in the UC specification. It provides automatic tool support to detect the missing part in the requirements specification and recommend appropriate instructions for including it to make the UC specification as complete as possible.

Business Process Notation is also used for modeling use cases by [14, 15]. The proposal uses business processes to derive UCs, through an algorithm that implements meta-models for use case diagrams (UCD) and for business process models (BPM). Thus, it creates UC diagrams more quickly because BPM is often available in a company in work instructions or administrative manuals in a clear and structured way. In addition, it tends to produce consistent and complete specifications.

Domain-Specific Languages (DSL) are used to describe UC [16], specifying the user and system actions clearly and precisely. This language uses a specific text syntax that allows a certain formalization in the UC model description. In this way, it can increase productivity by promoting understandable communication between engineers and domain experts, in addition to allowing the removal of ambiguities and inconsistencies observed in natural language texts.

Another strategy adopted is the **Fragment of Use Cases** [55, 56]. In this, the UC text is written using fragment composition in which each fragment represents a recurring set of interactions necessary to achieve a sub-objective. Each fragment can be customized and is coded using the best practices for writing the UC steps. Thus, the aim is to reduce the time required for the preparation of high-quality UC specifications. This strategy seeks to eliminate ambiguities, redundancies, inconsistencies, and conflicts with domain terminology (UCs contaminated by jargon). In this way, it standardizes and promotes the concise specifications of UCs, facilitating the maintenance and understanding of UCs by those involved with the system.

The **Communication Media** strategy [1] replaces face-to-face communication with Think-Pair-Square, a structured text-based chat, suitable for solving problems in the learning field. Its great application is in offering appropriate resources for distributed modeling, common in current development projects.

The **Mental Models** strategy [6] uses virtualization in UCM to propose a conceptual mental model representing the user's thinking of how it works. It refines the requirements phase by structuring the *imagination* process in a formal visualization stage, to be carried out before creating the UC diagram. In this way, helps inexperienced developers to overcome their difficulties in defining functional requirements at UCM and produces a tangible visual result of what the developer perceives as the user mental model of how something works.

The use of **Visual Languages** to describe UCs, replacing the UML language, is proposed in [43]. The authors consider that UML does not satisfactorily address the concerns (in terms of requirements) of human-computer interaction (HCI) professionals. Thus, they provide a common mechanism of communication and understanding between IHC professionals and software engineers (ES) so that the development and description of UC portray the interests of both together.

Reverse Engineering is a strategy that aims to systematically extract a large amount of information from UC descriptions [44]. The information can be read by machine and serves as a guideline for the "assembly of the UC diagram." Alternatively, the reverse process can be used: to decompose the diagram to assemble the UC specification. It seeks to systematically provide a minimal skeleton as a starting point for UCM.

Finally, [57] proposes the use of **Automatic Layout** with guidelines for defining a UC diagram. It focuses on the lack of appropriate layout mechanisms to provide an understanding of the system graphically and seeks to make up for the significant lack of resources for diagramming UCs as well as for displaying differences visually.

Table 2 summarizes the quality attributes enhanced by each strategy.

5 Correlation Between *Difficulties* and Strategies for UCM

This section presents an overview of the correlation proposed in [8] between modeling difficulties and strategies to mitigate these difficulties concerning the quality attributes for UC presented in Sect. 2.3.

The methodology used to establish the correlation between difficulties and strategies uses as input the information contained in the papers retrieved from the SLR [26]. We analyzed the experiments described in the SLR selected papers with the support of an inductive theory based on data analysis named Grounded Theory (GT) [41] to categorize the difficulties in UCM that we identified in the literature (Sect. 2.1). The result of this analysis made it possible to identify which quality attributes can be affected by each of the defined difficulty categories. Details of this process can be found in [8].

Table 3 shows the relationship established between difficulty and quality attributes. The first and second columns show the difficulty identifier and its description. Similarly, the third and fourth columns show the quality attribute's identifier affected by the difficulty and its description.

The combination of the results presented in Tables 2 and 3 allowed the definition of the correlation. Figure 1 shows part of the correlation, which is described in detail in [8]. On the top, there is the difficulty in identify/Extract/Discover UCs, actors and relationships. This difficulty affect the completeness-accuracy-integrity quality attribute, and can be mitigated by four different UCM strategies (on the botton): Role-playing, Natural Language Processing, Business Process Modeling, and Scenario pattern.

6 The Mechanism to Assist UCM Inspection

The correlation proposed in Sect. 5 helps modelers identify which strategies can mitigate specific difficulties in use-case modeling to enhance certain quality attributes.

Table 2. UC modeling strategies and quality attributes.

Strategies	Ambiguity	Inconsistency	Redundancy	Incompleteness	Incomprehensibility	Communicability
Ontology					X	
Antipattern	X	X				
Roleplaying				X	X	
Natural language processing	X	X	X	X	X	
Scenario patterns		X		X		
Business process notation	X		X	X	X	
DSLs	X	X				
Use case fragment	X	X	X		X	
Communication media						X
Mental model			X			
Visual language		X	X			X
Reverse engineering		X				
Automatic layout					X	

Table 3. Difficulties that affect quality attributes.

Id	Difficulty	Id At	Quality attribute affected
D1	Identify/extract/discover UC, actor and relationship	Q1	Completeness-accuracy-integrity
D2	Represent/express model elements	Q3	Correctness
D3	Write in detail the semantics of the UC model	Q2	Consistency
D4	Understand/interpret the problem domain.	Q1	Completeness - accuracy - integrity
D5	Perceive implicit requirements	Q4, Q5	Understandability; integrity
D6	Synthesize use cases	Q7	Abstraction level
D7	Condense the various information from the UC model	Q6	Redundancy

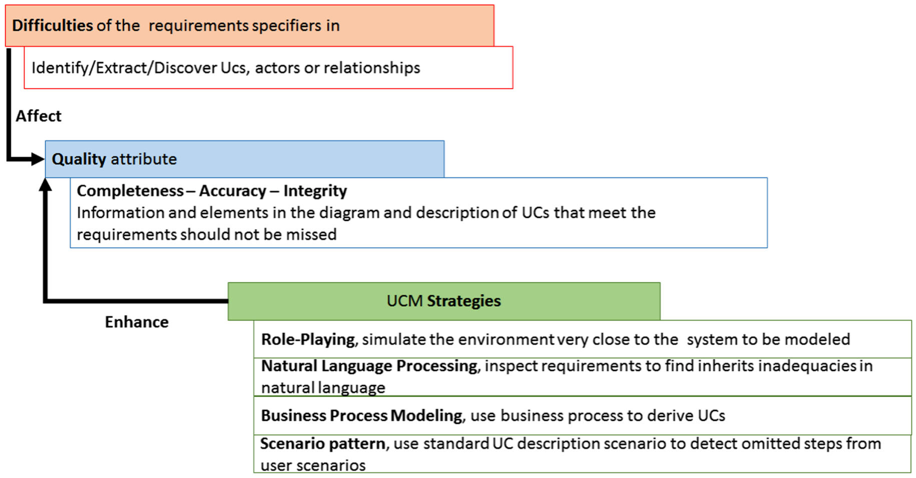

Difficulties of the requirements specifiers in
Identify/Extract/Discover Ucs, actors or relationships
Affect
Quality attribute
Completeness – Accuracy – Integrity
Information and elements in the diagram and description of UCs that meet the requirements should not be missed
Enhance
UCM Strategies
Role-Playing, simulate the environment very close to the system to be modeled
Natural Language Processing, inspect requirements to find inherits inadequacies in natural language
Business Process Modeling, use business process to derive UCs
Scenario pattern, use standard UC description scenario to detect omitted steps from user scenarios

Fig. 1. Part of the proposed correlation.

According to [5], the modeler difficulties in UCM usually insert defects in UC models. Thus, in this paper, we consider the hypothesis that reducing the defects in UC model after using a strategy means that the difficulty that would potentially generate such defects is mitigated. However, to assess this hypothesis, we need a systematic mechanism to inspect a UC model and detect possible defects after applying a strategy.

Inspection is a static activity based on the visual examination of development products to detect defects, violations of development patterns, and other problems without trying to solve the identified problems [45]. In UCM, this activity consists of checking if the model expresses the requirements and can be understood by all involved. A checklist defines a list of questions that the inspectors must answer yes or no, must be prepared to detect the defects in UC model during the inspection [42].

In this section, we propose a mechanism for UCM developers/instructors to build a checklist to detect defects in the UC model based on the strategies presented in Sect. 4. The generated checklist will evaluate the quality attributes enhanced by the strategy adopted in its construction. We consider a mechanism a set of previous and necessary procedures to be performed for the construction of the checklist. The mechanism purpose is to guide the definition of a checklist from which to measure the effect of a specific strategy on difficulties of the students at UCM.

The checklist must contain questions to identify defects related to quality attributes. For example, if a difficulty affects the completeness of a UC model, it is because this difficulty inserts defects that make the model incomplete. The checklist must then contain questions that allow finding the defects related to the attribute of quality completeness. Thus, the mechanism's procedures are as follows (Fig. 2):

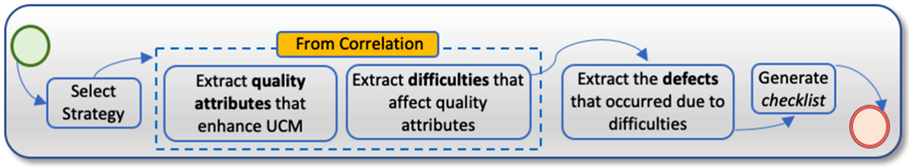

From Correlation
Select Strategy
Extract quality attributes that enhance UCM
Extract difficulties that affect quality attributes
Extract the defects that occurred due to difficulties
Generate checklist

Fig. 2. Mechanism to generate the inspection checklist.

(a) Select Strategy - This procedure aims to choose the strategy to be used for UCM. Not enough criteria were found in the literature to recommend one strategy more than another. Therefore, the selection must be conditioned to previous knowledge of what difficulties affect the group of modelers at the moment of UCM and then, consulting the correlation (Sect. 5), select the corresponding strategy for such difficulties. It is essential to be aware of the strategy goal, specific procedures, and what contribution should be expected from it (Sect. 4).
Procedures (B) and (C) must be based on the correlation.

(b) Extract quality attributes enhanced by the strategy - Knowing what the attributes are, we aim to identify in the rules and guidelines for UCM (Sect. 2.2) conditions related to these attributes. It is recommended to elaborate a small synthesis of the understanding of each attribute extracted in this phase.

- Extract the difficulties that affect the quality attributes, after identifying the difficulty in the correlation, and

(c) Extract the defects inserted according to the difficulties. Add the knowledge obtained in (B) and (C), and draw up the list of defects for each quality attribute of the procedure (B).

(d) Generate checklist. In this procedure, questions are prepared to answer whether each defect generated in procedure (D) exists or not when the inspection of the UC diagram or description is performed.

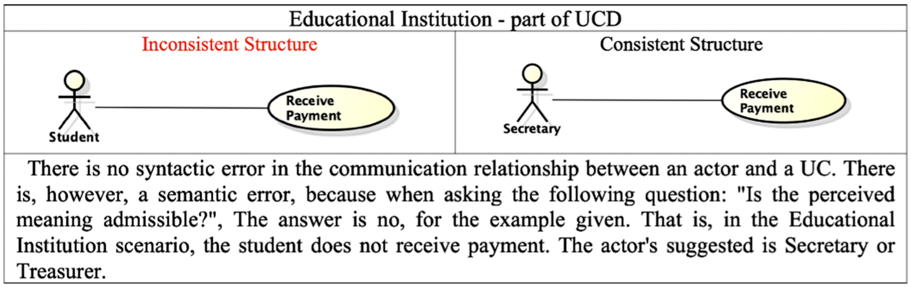
6.1 Applying the Mechanism in a UCM Strategy

To assess the mechanism proposed initially, we defined the mechanism for a specific strategy. Then we apply the mechanism to evaluate the models created in the controlled experiment presented in [8] (Sect. 7).

The following steps illustrate the execution of the procedures defined in Fig. 2:

(a) Select Strategy - Antipattern.
(b) Extract Quality Attributes Enhanced - Consistency and Ambiguity.

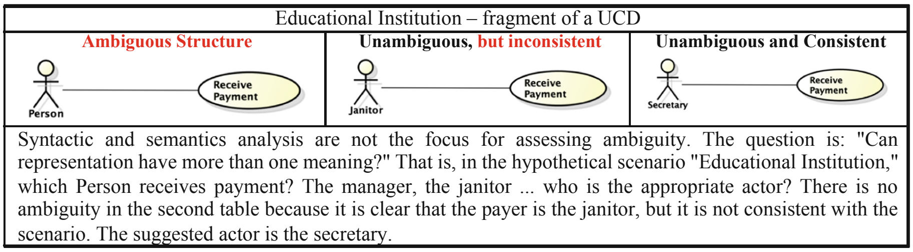
The Consistency attribute refers to structure, elements, language, grammar, and any information in the diagram and description of a UC, which must have the semantics expected of them. They need to be coherent, logical, and consistent. To illustrate this attribute, consider Fig. 3.



There is no syntactic error in the communication relationship between an actor and a UC. There is, however, a semantic error, because when asking the following question: "Is the perceived meaning admissible?", The answer is no, for the example given. That is, in the Educational Institution scenario, the student does not receive payment. The actor's suggested is Secretary or Treasurer.

Fig. 3. Example of inconsistent and consistent diagrammatic structure.

The Ambiguity attribute refers to structure, elements, language, grammar, and any model information (present in the UC diagram or description) must be clear so that the interpretation is unique and the understanding is the same for both the client and for the developer. To illustrate this attribute, consider Fig. 4.


Educational Institution – fragment of a UCD
Ambiguous Structure
Person
Receive Payment
Unambiguous, but inconsistent
Janitor
Receive Payment
Unambiguous and Consistent
Secretary
Receive Payment

Syntactic and semantics analysis are not the focus for assessing ambiguity. The question is: "Can representation have more than one meaning?" That is, in the hypothetical scenario "Educational Institution," which Person receives payment? The manager, the janitor ... who is the appropriate actor? There is no ambiguity in the second table because it is clear that the payer is the janitor, but it is not consistent with the scenario. The suggested actor is the secretary.

Fig. 4. Example of unambiguous and ambiguous diagrammatic structure.

(c) Extract Difficulties that Affect Quality Attributes.

The Consistency attribute is affected when the specifier has difficulty describing/specifying the semantics of the UC model. This difficulty presents itself as an inability to attribute value or highlight the meaning of the information. The specifier finds it challenging to convey what is necessary with logic and coherence not to compromise the modeled scenario with inconsistencies.

The Ambiguity attribute is affected when the specifier has difficulty perceiving implicit requirements. This difficulty presents itself as an inability or doubt to *accurately* visualize a hidden requirement. For example, in Fig. 4, it is possible that in the requirements document, it was not explicit that the secretary is responsible for the *receiving payment* task. Thus, the modeler can associate the functionality *receive payment* to a generic actor *person*, believing that any person can perform this use case. Ambiguity happens because it leads to different interpretations.

(d) Extract the Defects Inserted According to the Difficulties.

For illustration purposes, defects that appear in the UC diagram are listed in Tables 4 and 5 that make it inconsistent and ambiguous, respectively. The first column identifies the defect from an id (e.g. DfC1), and the second column shows the defect.

Table 4. Defects that compromise the quality attribute Consistency in the UC diagram.

Id	Description
DfC1	Relate actor and UC, when the actor is incompatible to interact with UC
DfC2	Name UC with a name inconsistent with the purpose of the UC
DfC3	Appoint an actor with job titles and not with his role in the system
DfC4	The UC diagram is not plausible with the list of requirements
DfC5	Decompose a UC when its parts alone do not represent value to an actor
DfC6	Use CRUD (Create, Retrieve, Update, Delete) functionality instead of a single generic UC (Manage … or Maintain …)
DfC7	Establish a communication relationship between two UCs instead of inclusion or extension
DfC8	Relate UC and actor through generalization
DfC9	Define an inclusion, extension or generalization relationship between UCs whose removal prevents understanding of the main UC's objective
DfC10	Have an inclusion UC that relates to only one UC
DfC11	Have an extension UC that does not add functionality to the base UC
DfC12	Define a UC as inclusion and extension at the same time

Table 5. Defects that compromise the quality attribute ambiguity in the UC diagram.

Id	Description
DfA1	Name actor or UC with long or inexpressive terms, which do not reflect their role or goal, with varying meaning or adverbs, synonyms, adjectives, pronouns, homonyms, and references
DfA2	Define a single UC as inclusion and extension simultaneously
DfA3	Define a generic UC and another UC that is part of the generic one without relating one with the other
DfA4	Specialize an actor who cannot establish inherited relationships
DfA5	Add brief description notes, without which it is not possible to understand the diagram
DfA6	Do not clearly show users and system functionality according to the list of requirements
DfA7	Have two or more UCs that perform the same functionality
DfA8	Modeling two or more actors who have the same role in the system with a different name
DfA9	Not making the relationships between the actors and the UCs clear

(e) Generate Checklist.

A question is formulated related to each defect to verify the defects presented in Tables 4 and 5. For example, for defect DfC1: Was a relationship between actor and UC found in the inspected diagram when the actor is incompatible with the UC? The answer must always be Y or N, yes or no, respectively.

7 Proposal Evaluation

This section details the controlled experiment briefly presented in [8]. The defect detection mechanism presented in Sect. 6 was also used to inspect the models generated by the experiment participants. The experiment was carried out following [40] guidelines and is structured in four stages: scope, planning, operation, and data analysis and interpretation, detailed in the following subtopics.

7.1 Experiment Scope

Each UMC strategy identified in Sect. 4 must be assessed individually. We started this validation with the Antipattern strategy because it is the strategy most found in the literature and the most detailed in terms of steps to guide the use of the strategy. Following the proposed correlation (Sect. 5), this strategy is related to the mitigation of difficulties that affect the consistency and ambiguity attributes of a UC models.

Experiment Goal. The experiment goal is defined according to the GQM (Goal Question Metric) template. It consists of *analyzing* the Antipattern-based strategy *for the* *purpose of* assess its effectiveness in mitigating the difficulties of describing/detailing the semantic of UC models, and implicit understanding requirements *concerning* consistency and ambiguity *from the point of view of* undergraduate students in the software engineering discipline.

Research Questions. Based on the objective of this experiment, the following research questions (RQ) were defined: *RQ1: Is the diagram produced with the support of the strategy free from defects that would make it inconsistent and ambiguous?* This question sought to assess whether the use of the strategy corrected defects or prevented the appearance of new ones. *RQ2: Does using the Antipattern strategy mitigate the difficulties that affect the consistency and ambiguity of use-case diagram?* With this question, we tried to verify if the difficulties of students that affect consistency and ambiguity disappeared or reduced.

Metric. The metric used to assess the correlation was the number of defects observed in the diagrams produced. Modeled diagrams were inspected using the defect detection mechanism defined in Sect. 6, based on the defect count according to Tables 4 and 5. These defects are inserted due to difficulties in describing/specifying the UC diagram's semantic and understanding implicit requirements. It was assessed whether the Antipattern strategy mitigates these difficulties by reducing these defects.

7.2 Experiment Planning

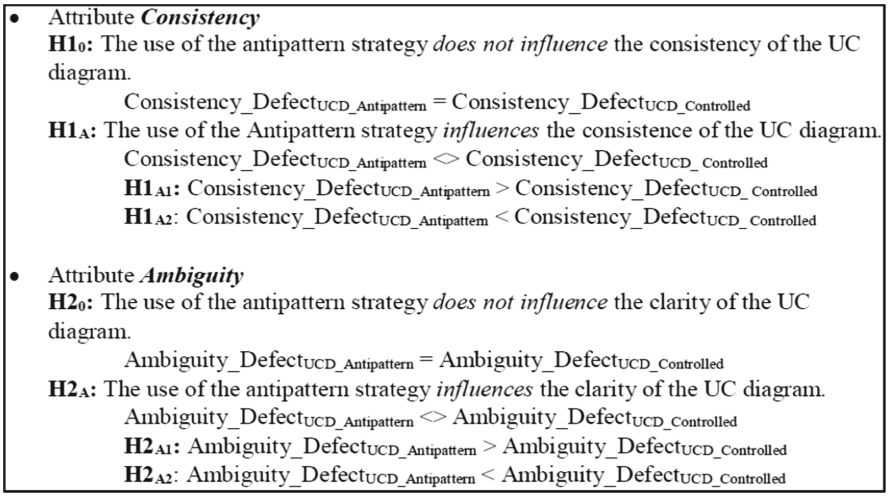
The experiment was planned in terms of context selection, type of experiment, formulation of hypotheses, dependent and independent variables, and instrumentation.

Context Selection. The experiment was conducted in an academic environment formed by undergraduate students with knowledge of the basic syntax and semantics of UML use-case diagram. To ensure that students had this knowledge, only those studying or who had completed the Software Engineering discipline in Computer Science courses were selected.

Experiment Type. The type of design used in this experiment was based on [40], comprising one factor, the strategy for UCM based on Antipattern, and two treatments: (1) the modeling of the UC diagram without the Antipattern strategy and (2) the modeling of the UC diagram with the Antipattern strategy. Each participating student used the two treatments to model the same scenario. Two sets of diagrams were generated: one without the Antipatterns strategy help (we called this set of UCD_Controlled); and another set based on Antipattern (called this set of UCD_Antipattern).

Hypotheses Formulation. Hypotheses were formulated to conduct the evaluation: null (H0) and alternative (HA), corresponding to the existence of defects in the UCD_Controlled and UCD_Antipattern, as showed in Fig. 5.

Variables Definition. The independent variable consisted of modeling the UC diagram that assumed two levels, modeling without the strategy and modeling using the Antipattern. The dependent variables were the attributes of UCM quality, consistency, and ambiguity, directly related to the measures used to test the hypotheses.



- Attribute ***Consistency***

  $H1_0$: The use of the antipattern strategy *does not influence* the consistency of the UC diagram.

  $\text{Consistency\_Defect}_{\text{UCD\_Antipattern}} = \text{Consistency\_Defect}_{\text{UCD\_Controlled}}$

  $H1_A$: The use of the Antipattern strategy *influences* the consistence of the UC diagram.

  $\text{Consistency\_Defect}_{\text{UCD\_Antipattern}} <> \text{Consistency\_Defect}_{\text{UCD\_ Controlled}}$

  $H1_{A1}$: $\text{Consistency\_Defect}_{\text{UCD\_Antipattern}} > \text{Consistency\_Defect}_{\text{UCD\_ Controlled}}$

  $H1_{A2}$: $\text{Consistency\_Defect}_{\text{UCD\_Antipattern}} < \text{Consistency\_Defect}_{\text{UCD\_ Controlled}}$

- Attribute ***Ambiguity***

  $H2_0$: The use of the antipattern strategy *does not influence* the clarity of the UC diagram.

  $\text{Ambiguity\_Defect}_{\text{UCD\_Antipattern}} = \text{Ambiguity\_Defect}_{\text{UCD\_Controlled}}$

  $H2_A$: The use of the antipattern strategy *influences* the clarity of the UC diagram.

  $\text{Ambiguity\_Defect}_{\text{UCD\_Antipattern}} <> \text{Ambiguity\_Defect}_{\text{UCD\_Controlled}}$

  $H2_{A1}$: $\text{Ambiguity\_Defect}_{\text{UCD\_Antipattern}} > \text{Ambiguity\_Defect}_{\text{UCD\_Controlled}}$

  $H2_{A2}$: $\text{Ambiguity\_Defect}_{\text{UCD\_Antipattern}} < \text{Ambiguity\_Defect}_{\text{UCD\_Controlled}}$

Fig. 5. Data collected on defect inspection that makes the UC diagram inconsistent [8].

Instrumentation. The resources used to carry out this experiment involved: guidance for carrying out the experiment; a scenario for modeling the UC diagram; training to present and understand the strategy; selection of the material needed to execute the strategy; checklists to inspect the modeled diagrams. The experiment was conducted in two stages: stage 1, where all students modeled the UC diagram based on the same specification, and step 2, in which students were trained to understand and use the strategy for UCM and the artifacts to support its application. After completing the two stages, diagrams modeled with and without the strategy for inspection, comparison, and subsequent analysis were collected from each student.

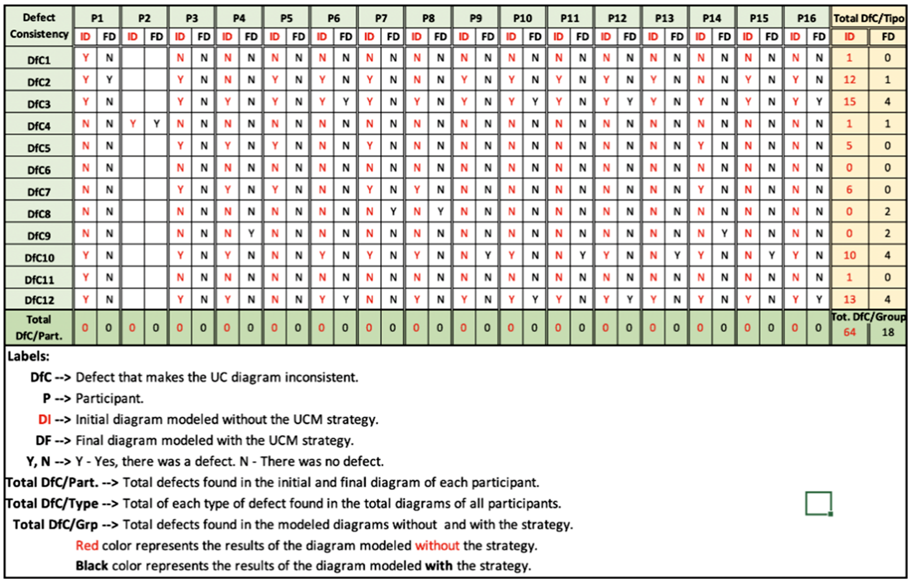
7.3 Experiment Operation

The operation stage comprised the following steps: data preparation, execution, collection, and validation.

Data Preparation. The experiment participants were 16 students from two different software engineering classes from a Computer Science course.

Execution. The experiment was carried out in two replications, one for each class, in different days. Each student initially modeled the UC diagram without using the Antipattern strategy (in the UCD_Controlled group). Diagrams generated were collected for later evaluation. The strategy was then explained, and material supporting the strategy was provided to the students, e.g., a list of Antipattern with examples of its application. Using this material, the students improved the first modeled diagram, when necessary (in the UCD_Antipattern group). So as, each student produced a second UC diagram, all of which were also collected by the researchers for subsequent analysis.

Data Collection. The researchers inspect the diagrams collected in both groups using the checklist defined according to the mechanism showed in Sect. 6. Figures 6 and 7 show the data collected in this inspection.

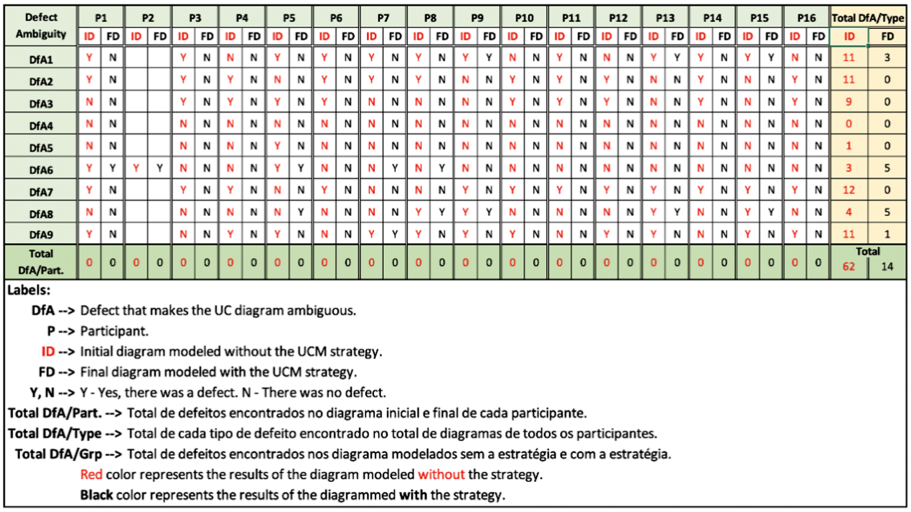


Defect Consistency	P1		P2		P3		P4		P5		P6		P7		P8		P9		P10		P11		P12		P13		P14		P15		P16		Total DfC/Tipo	
	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD
DfC1	Y	N			N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	1	0
DfC2	Y	Y			Y	N	N	N	Y	N	Y	N	Y	N	N	N	Y	N	Y	N	Y	N	Y	N	Y	N	N	N	Y	N	Y	N	12	1
DfC3	Y	N			Y	N	Y	N	Y	N	Y	Y	Y	N	Y	N	Y	N	Y	Y	Y	N	Y	Y	Y	N	Y	N	Y	N	Y	Y	15	4
DfC4	N	N	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	1	1
DfC5	N	N			Y	N	Y	N	Y	N	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N	N	N	N	5	0
DfC6	N	N			N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	0	0
DfC7	N	N			Y	N	Y	N	Y	N	N	N	Y	N	Y	N	N	N	N	N	N	N	N	N	N	N	Y	N	N	N	N	N	6	0
DfC8	N	N			N	N	N	N	N	N	N	N	N	Y	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	0	2
DfC9	N	N			N	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N	N	N	0	2
DfC10	Y	N			Y	N	Y	N	N	N	Y	N	Y	N	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	10	4
DfC11	Y	N			N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	1	0
DfC12	Y	N			Y	N	Y	N	N	N	Y	Y	N	N	Y	N	Y	N	Y	Y	Y	N	Y	Y	Y	N	Y	N	Y	N	Y	Y	13	4
Total DfC/Part.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Tot. DfC/Group 64	18

Labels:

DfC --> Defect that makes the UC diagram inconsistent.
P --> Participant.
DI --> Initial diagram modeled without the UCM strategy.
DF --> Final diagram modeled with the UCM strategy.
Y, N --> Y - Yes, there was a defect. N - There was no defect.
Total DfC/Part. --> Total defects found in the initial and final diagram of each participant.
Total DfC/Type --> Total of each type of defect found in the total diagrams of all participants.
Total DfC/Grp --> Total defects found in the modeled diagrams without and with the strategy.
Red color represents the results of the diagram modeled without the strategy.
Black color represents the results of the diagram modeled **with** the strategy.

Fig. 6. Data collected on defect inspection that makes the UC diagram inconsistent.



Defect Ambiguity	P1		P2		P3		P4		P5		P6		P7		P8		P9		P10		P11		P12		P13		P14		P15		P16		Total DfA/Type	
	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD	ID	FD
DfA1	Y	N			Y	N	N	N	Y	N	Y	N	Y	N	Y	N	Y	Y	N	N	Y	N	N	N	Y	Y	Y	N	Y	Y	N	N	11	3
DfA2	Y	N			Y	N	Y	N	N	N	Y	N	Y	N	Y	N	N	N	Y	N	Y	N	Y	N	N	N	Y	N	N	N	Y	N	11	0
DfA3	N	N			Y	N	Y	N	Y	N	Y	N	N	N	N	N	N	N	Y	N	Y	N	Y	N	N	N	Y	N	N	N	Y	N	9	0
DfA4	N	N			N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	0	0
DfA5	N	N			N	N	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	1	0
DfA6	Y	Y	Y	Y	N	N	N	N	Y	Y	N	N	N	Y	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	3	5
DfA7	Y	N			Y	N	Y	N	N	N	Y	N	N	N	N	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	12	0
DfA8	N	N			N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N	N	N	Y	Y	N	N	Y	Y	N	N	4	5
DfA9	Y	N			N	N	Y	N	Y	N	N	N	Y	Y	Y	N	Y	N	Y	N	N	N	Y	N	Y	N	N	N	Y	N	Y	N	11	1
Total DfA/Part.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Total 62	14

Labels:

DfA --> Defect that makes the UC diagram ambiguous.
P --> Participant.
ID --> Initial diagram modeled without the UCM strategy.
FD --> Final diagram modeled with the UCM strategy.
Y, N --> Y - Yes, there was a defect. N - There was no defect.
Total DfA/Part. --> Total de defeitos encontrados no diagrama inicial e final de cada participante.
Total DfA/Type --> Total de cada tipo de defeito encontrado no total de diagramas de todos os participantes.
Total DfA/Grp --> Total de defeitos encontrados nos diagrama modelados sem a estratégia e com a estratégia.
Red color represents the results of the diagram modeled without the strategy.
Black color represents the results of the diagrammed **with** the strategy.

Fig. 7. Data collected in the inspection of defects that make the UC diagram ambiguous.

In both Figs. 6 and 7, column 1 shows the code corresponding to each defect listed in Tables 4 and 5. For example, DfC1 indicates the defect *Relating actor and UC, when the* *actor is incompatible with interacting with UC*, which makes the diagram inconsistent. Columns 2 to 17 show the participants of the experiment. For example, P1 is participant 1. Each participant modeled a diagram without knowing the strategy, represented by DI (Initial Diagram) and the diagram using the strategy, represented by DF (Final Diagram). Each S and N indicates the researcher response when inspecting the presence or absence of defect in the initial and final diagram. The last line shows the total of defects found in the initial and final diagram of each participant. At the end of this line is the total of defects for the entire group. Finally, the last column shows the total of each type in the set of diagrams.

Data Validation. The validation was performed to assess any inconsistency in the data collected. The diagrams of participant P2 were discarded because he did not perform the modeling for the scenario presented. No other discrepancies were found. Therefore, all other answers were used in the data analysis.

7.4 Data Analysis and Interpretation

This step is responsible for the analysis of the data collected so that conclusions can be drawn. Figure 8 illustrates the result of the diagrams inspection. Eighty-two defects related to the inconsistency, of which 64 were identified in the UCD_Controlled group and 18 in the UCD_Antipattern group. Regarding ambiguity, 76 defects were found, of which 62 were identified in the UCD_Controlled group and 14 UCD_Antipattern group. The defects found in UCD_Antipattern group represented by the number 18 and 14, related to inconsistency and ambiguity, respectively, are defects found in UCD_Antipattern. They were not corrected with the strategy usage or are new defects that emerged after using the strategy.

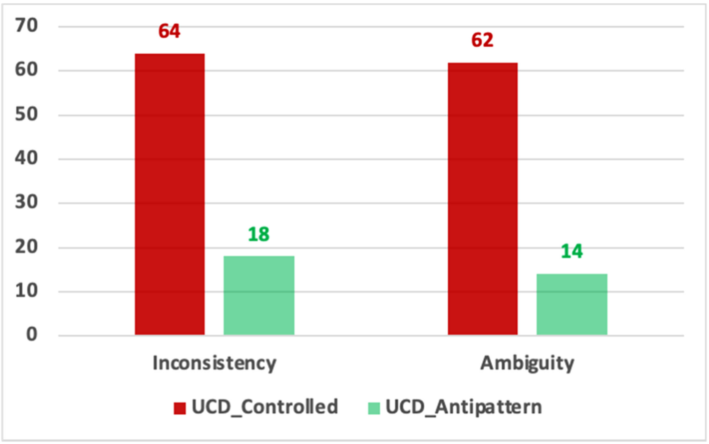


Fig. 8. Number of defects found in UC diagrams inspection.

In the UCD_Controlled group, the number of occurrences of each defect that made the diagrams inconsistent is illustrated in Fig. 9. The type of defect that was most evident was *DfC3 (Name actor with job titles and not with his role in the system)* with 15 occurrences, followed by *DfC12 (Define a UC as inclusion and extension at the same time)* with 13 occurrences, and *DfC2 (Name UC with a name inconsistent with the purpose of the UC)* with 12 occurrences. Three types of defects occurred only once, *DfC1 (Relate actor and UC, when the actor is incompatible with interacting with the UC), DfC4 (The UC diagram is not plausible with the list of requirements),* and *DfC11 (Have an Extension UC that does not add functionality to the base UC).* The defect *DfC6 (Use CRUD, i.e. Create, Retrieve, Update, Delete, functionality instead of a single generic UC)* was not found in any diagram.

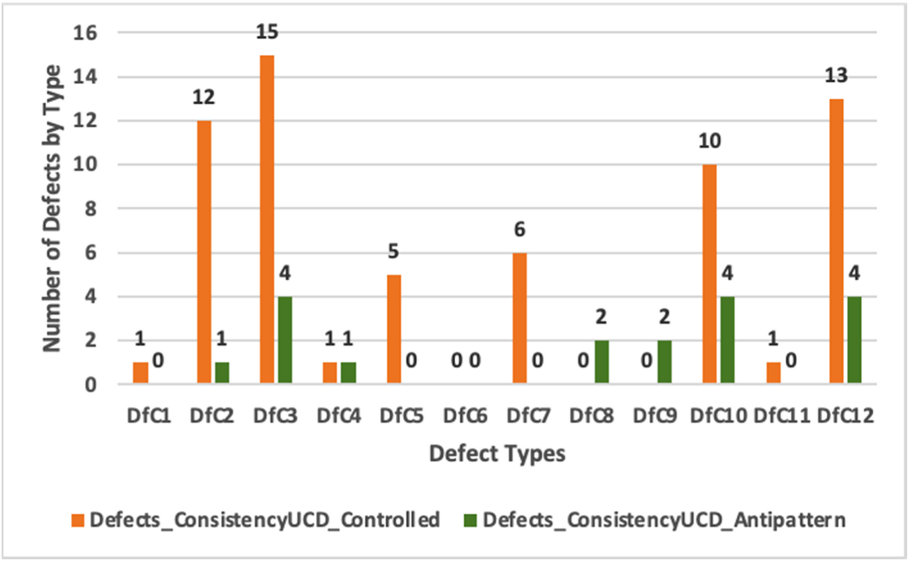


Fig. 9. Number of occurrence of the different types of defects related to consistency found in the inspection in the UC diagrams.

Figure 9 also shows the number of occurrences of each defect in the UCD_Antipattern. In this data set, in particular, two types of defects, *DfC8 (Relate UC and actor through generalization),* and *DfC9 (Define inclusion, extension or generalization relationship between UCs whose removal prevents the understanding of the main UC objective)* occurred two times, however, not previously found in UCD_Controlled group.

Figure 10 illustrates the number of occurrences of each type of defect that made the diagrams ambiguous. In the UCD_Controlled, group the type of defect that was most evident was DfA7 (Having two or more UCs that perform the same functionality) with 12 occurrences, followed by DfA9 (Do not make the relationships between the actors and the UCs clear), DfA2 (Define a single UC as inclusion and extension simultaneously), and DfA1 (Name actor or UC with long or inexpressive terms), which occurred the same number of times, 11 occurrences. Various types of defects that appeared in the UCD_Controlled group disappeared in the UCD_Antipattern, such as DfA2 (Define a single UC as inclusion and extension simultaneously), DfA3 (Define a generic UC and another UC that is part of the generic without having a relationship between them), and DfA5 (Add brief description notes, without which it is not possible to understand the diagram). Two types of defects, DfA6 (Do not clearly show the users and the functionality of the software according to the list of requirements) and DfA8 (Have two or more actors with the same role in the system with a different name) increased in UCD_Antipattern.

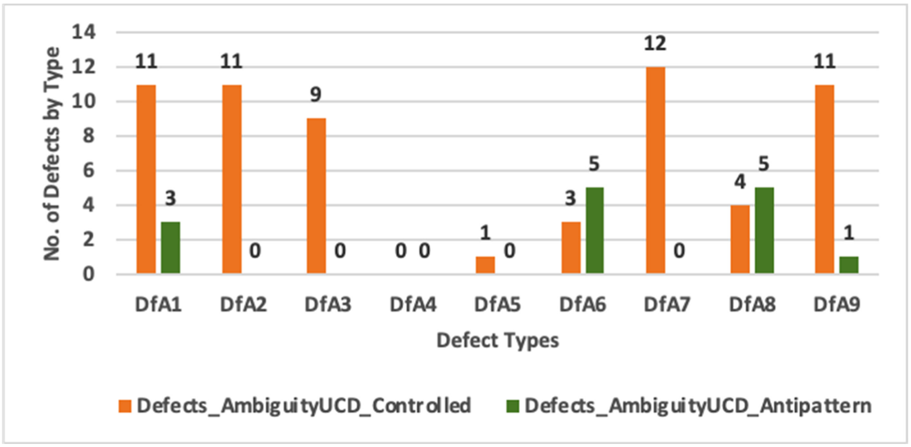


Fig. 10. Number of occurrence of the different types of ambiguity-related defects found in the inspection in the UC diagrams.

Hypothesis Assessment. In order to verify whether there is a significant difference in the quality (regarding consistency and ambiguity) of the diagrams modeled with and without the Antipattern based strategy, the hypothesis test was performed [40]. To select the test to be used, the normality of the data was verified. The tests showed that the distributions of the number of defects per diagram are expected in the UCD_Controlled group and in the UCD_Antipattern group. Figures 11 and 12 present the scatter plot showing that the number of defects generated by each participant is close to the average (linear across the points of the graph), concerning consistency and ambiguity.

In Fig. 11, for example, for participant P1, six defects were found in the UCD_Controlled group and one in the UCD_Antipattern group. On average, there are four defects in the UCD_Controlled group, and one in the UCD_Antipattern. The standard deviation is 1 and a tolerance of ± 1 for the two data sets. Hence, both six defects in the UCD_Controlled group and one defect in the UCD_Antipattern group are in the normality of the data.

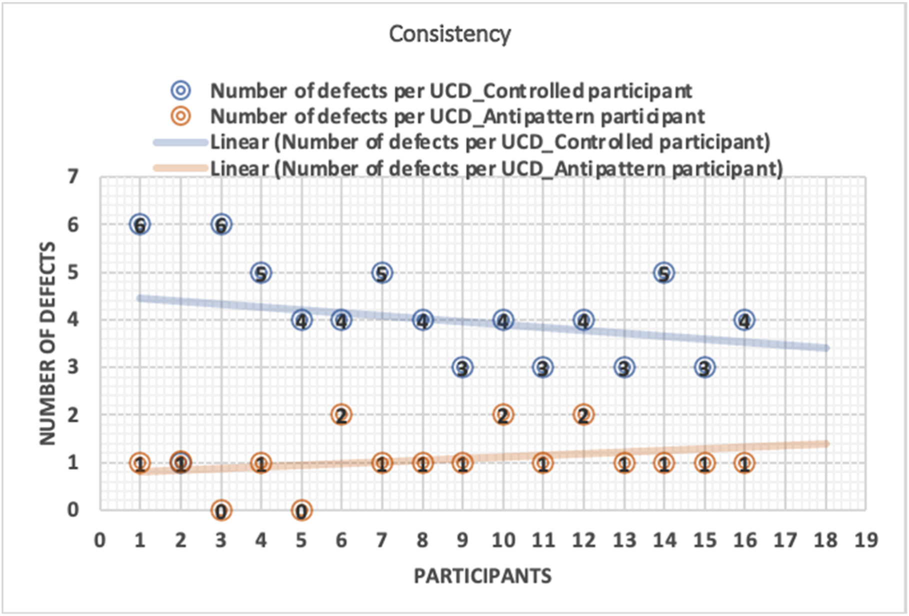


Fig. 11. Scatter plot of the number of defects related to consistency in the initial and final diagram of each participant.

Figure 12 presents the same analysis for ambiguity attribute. The average defect number for UCD_Controled group is 3.875, with a standard deviation 1 and tolerance of ±1. For UCD_Antipattern group, the average is 0.875, with a standard deviation 1 and tolerance of ±1. In both Figs. 11 and 12 x-axis show the participants P1, P2, up to P16, and the defects in the initial and final diagrams of each are the graph points.

As the data obey a normal distribution, the Shapiro-Wilk [58] test was used with the p-value (probability on the null hypothesis) equal to 0.05 to accept or reject the hypotheses. If the p-value < 0.05, the result is significant, and the null hypothesis can be rejected. After the test was carried out, the result showed that all null hypotheses (H10: Consistency_DefectsUCD_Antipatten = Consistency_ Defects UCD_Controlled) and H20: Ambiguity_ Defects UCD_Antipattern = Ambiguity_ Defects UCD_Controlled must be rejected according to the following: p-value (p-value) = 0.034 for the UCD_Controled group and p = 0.040 for the UCD_Antipttern; in relation to ambiguity, p = 0.035 for the UCD_Controlled group and p = 0.037 for the UCD_Antipattern group.

Having rejected the null hypotheses H10 and H20 that the use of the Antipattern-based strategy does not influence, respectively, the consistency and/or ambiguity of the UC diagram, the alternative hypotheses, Ambiguity_ DefectsUCD_Antipattern <> Ambiguity_ DefectsUCD_Controlled, were accepted. That is, the use of the Antipattern-based strategy influences the consistency and/or ambiguity of the UC diagram, which can be confirmed in the boxplot graph in Fig. 13.

Research Questions Answers. Having evaluated the hypotheses, we sought to answer the research questions formulated for this experiment. Related to RQ1, as showed in

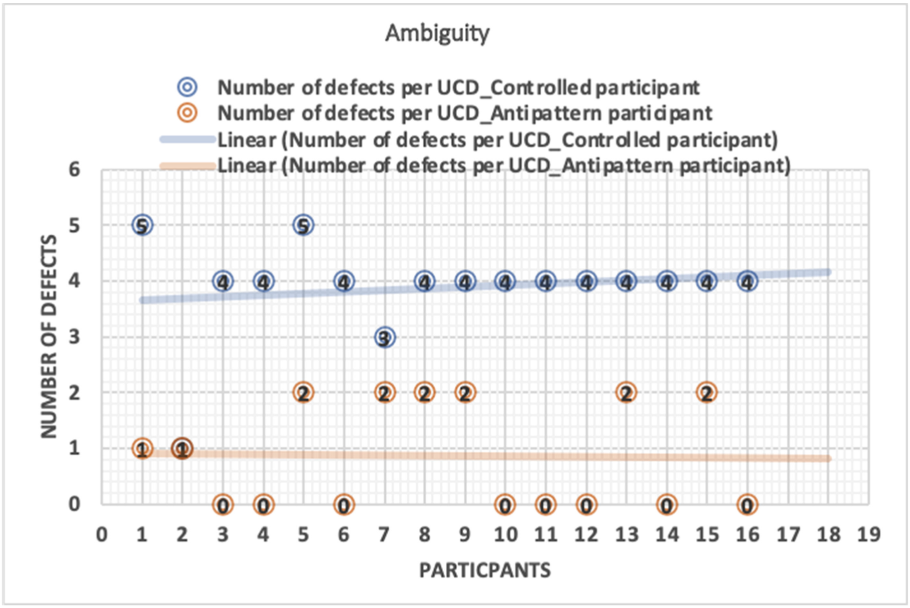


Fig. 12. Scatter plot of the number of defects related to ambiguity in the initial and final diagram of each participant.

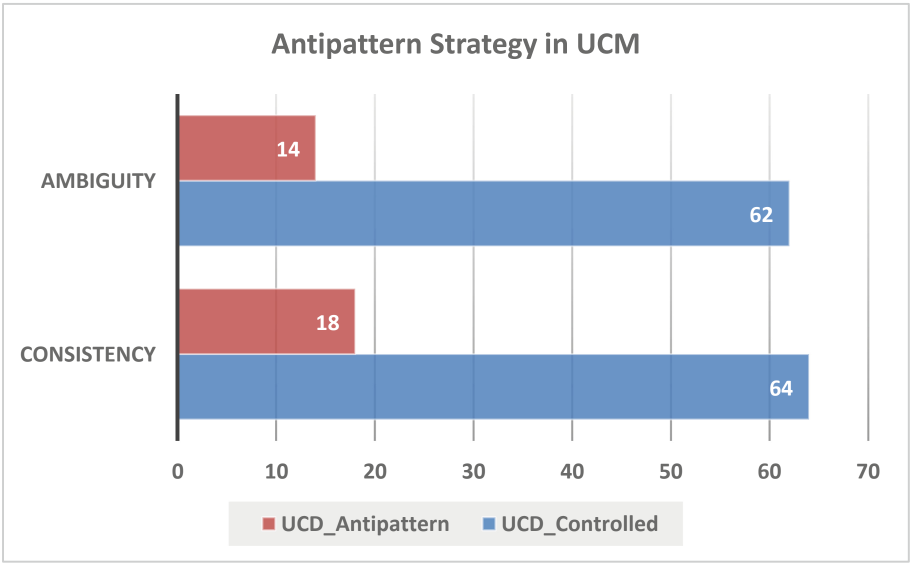


Fig. 13. Effect of Antipattern-based strategy on reducing ambiguity and inconsistency in the UC diagram [8].

Fig. 13, the use of the Antipattern based strategy in modeling the UC diagram considerably reduced the defects that make the diagram ambiguous and inconsistent. Concerning ambiguity, total defects reduced from 62 to 14 and, concerning consistency, total defects reduced from 64 to 18, when modeling with Antipattern. However, not all defects have been corrected, nor has it prevented the appearance of new ones.

Concerning RQ2, in this work, it was considered that the difficulties of the modelers generate the defects. Hence, if using Antipatterns reduces defects that affect consistency and ambiguity, then the difficulties that insert these defects were mitigated. However, there is a limitation in the results, which are considered as evidence and not conclusive.

7.5 Threats to Validity

To prevent bias in the validation, we now discuss some threats to this empirical study validity. Regarding internal validity, as the level of knowledge of the participant and experience in use case specification may influence the study results, we provided training for every participant in modelling use cases. Besides, we only selected participants who were enrolled in the software engineering discipline. Concerning external validity, to minimize the risk of sample representativeness, the diagrams produced by the participants were also inspected by people who did not participate in the experiment. The representativeness of the chosen domain and the size and complexity of the scenario are other threats to the experiment. As there was no possibility of using a real case, a scenario widely used in software modeling was chosen. However, experiments using real scenarios are necessary to validate our proposal better. Related to construction validity, we performed two pilot studies to validate the material used in the experiment. Distortions in understanding the antipattern strategy were minimized through a summary of examples of its use. Finally, concerning conclusion validity, the statistic method used may influence on the conclusion. Therefore, we consulted a specialist to define which method to adopt.

8 Conclusions and Future Works

Recent studies show the various difficulties of developers to model use cases. Therefore, the quality of the models is compromised due to the occurrence of defects. To mitigate these difficulties, we proposed a correlation between UCM defects and strategies to mitigate these defects based on UC' quality attributes.

This paper presented the controlled experiment performed to evaluate the correlation concerning the Antipattern strategy. In the experiment, participants developed a UC model in two stages, first without using the Antipattern strategy, and then assisted by this strategy. The produced UC models were analyzed using a checklist, which construction was guided by the proposed mechanism. A set of possible difficulties were identified in order to generate questions to be used in models inspection.

Shapiro-Wilk test was used with the p-value equal to 0.05 to accept or reject the hypotheses defined in the experiment. The results showed that the Antipattern-based strategy influences the consistency and ambiguity of quality criteria for the adopted scenario. We considered this a clear indication that Antipattern mitigates the difficulties related to it: difficulty to describe or to detail semantics in the UC model, and difficulty in understanding implicit requirements. Other difficulties may be mitigated by other strategies indicated in the correlation.

The strategy-difficulty correlation proposed in this paper organizes and guides the requirements specifier to select the most appropriate strategy to mitigate a given difficulty. This oriented indication that the correlation provides avoids adopting ineffective practices and makes the requirements specifier aware of several possibilities of applying tested and evaluated procedures to assist UCM. We are now working on new experiments to validate the other strategies contained in the correlation.

References

1. Erra, U., Portnova, A., Scanniello, G.: Comparing two communication media in use case modeling: results from a controlled experiment. In: ESEM 2010 Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (2010)
2. Tiwari, S., Gupta, A.: A systematic literature review of use case specifications research. In: Information and Software Technology, vol. 67, pp. 128–158 (2015)
3. El-Attar, M., Miller, J.: Constructing high quality use case models: a systematic review of current practices. Requir. Eng. **17**(3), 187–201 (2012)
4. Liu, S., Sun, J., Xiao, H., Wadhwa, B., Dong, J.S., Wang, X.: Improving quality of use case documents through learning and user interaction. In: 21st International Conference on Engineering of Complex Computer Systems (ICECCS), Dubai, pp. 101–110 (2016)
5. Nascimento, E.S., Silva, W., França, B.B.N., Gadelha, B., Conte, T.: Um Modelo sobre as Dificuldades para Especificar Casos de Uso. In: Conference Ibero-American on Software Engineering (CIBSE), Argentina (2017)
6. Beimel, D., Kedmi-Shahar, E.: Improving the identification of functional system requirements when novice analysts create use case diagrams: the benefits of applying conceptual mental models. Requir. Eng. **24**, 483–502 (2019). https://doi.org/10.1007/s00766-018-0296-z
7. Anda, B., Dreiem, H., Sjøberg, D.I.K., Jørgensen, M.: Estimating software development effort based on use cases—experiences from industry. In: Gogolla, M., Kobryn, C. (eds.) UML 2001. LNCS, vol. 2185, pp. 487–502. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45441-1_35
8. Bispo, C., Magalhães, A., Fernandes, S., Machado, I.: Mitigating difficulties in use-case modeling. In: Proceedings of the 22nd ICEIS, 22nd International Conference on Enterprise Information Systems, 2020, Prague, vol. 2, pp. 43–52 (2020)
9. El-Attar, M., Miller, J.: Improving the quality of use case models using antipatterns. Soft. Syst. Model. **9**(2), 141–160 (2010)
10. Khan, Y.A., El-Attar, M.: A model transformation approach towards refactoring use case models based on antipatterns. In: 21st International Conference on Software Engineering and Data Engineering, Los Angeles, California, USA, pp. 49–54 (2012)
11. Khan, Y.A., El-Attar, M.: Using model transformation to refactor use case models based on antipatterns. Inf. Syst. Front. **18**(1), 171 (2016)
12. El-Attar, M.: Improving the quality of use case models and their utilization in software development. Department of Electrical and Computer Engineering, Alberta University (2009)
13. Fourati, R., Bouassida, N., Abdallah, H.B.: A metric-based approach for anti-pattern detection in UML designs. In: Lee, R. (ed.) Computer and Information Science 2011, vol. 364, pp. 17–33. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21378-6_2

14. Bouzidi, A., Haddar, N., Abdallah, M.B., Haddar, K.: Deriving use case models from BPMN models. In: IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, 2017, pp. 238–243 (2017)
15. Cruz, E.F., Machado, R.J., Santos, M.Y.: From business process models to use case models: a systematic approach. In: Aveiro, D., Tribolet, J., Gouveia, D. (eds.) EEWC 2014. LNBIP, vol. 174, pp. 167–181. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06505-2_12
16. Saviä, D., Vlajiä, S., Lazareviä, S., Antoviä, I., et al.: Use case specification using the SilabReq domain specific language. Comput. Inf. **34**(4), 877–910 (2015)
17. Sommerville, I.: Software Engineering, 10th edn. University of St Andrews, Pearson, Scotland, London (2016)
18. Jacobson, I.: Use cases - yesterday, today, and tomorrow. Soft. Syst. Model. **3**(3), 210–220 (2004)
19. OMG Unified Modelling Language Superstructure - version 2.3. http://www.omg.org/spec/UML/2.3/ (2010)
20. Cockburn, A.: Writing Effective Use Cases. Addison Wesley, Reading (2000)
21. Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object-Oriented Software Engineering: A Use-Case Driven Approach. Addison-Wesley, Reading (1992). Edition
22. Anda, B., Hansen, K., Sand, G.: An investigation of use case quality in a large safety-critical software development project. Inf. Soft. Technol. **51**(12), 1699–1711 (2009)
23. Bolloju, N.: Exploring quality dependencies among UML artifacts developed by novice systems analysts. In: 12th Americas Conference on Information Systems, p. 472 (2006)
24. Siau, K., Poi-Peng, L.: Identifying difficulties in learning UML. Inf. Syst. Manag. **23**(3), 43–51 (2006)
25. Spence, I., Bittner, K.: Use Case Modeling. Addison-Wesley, Reading (2003)
26. Bispo, C., Fernandes, S., Magalhães, A.P.: Strategies for use case modeling: a systematic literature review. In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES 2019), pp. 254–263. ACM, New York (2019)
27. Ahmed, E.: Use of ontologies in software engineering. In: SEDE, pp. 145–150 (2008)
28. Dermeval, D., Vilela, J., Bittencourt, I.I., et al.: Applications of ontologies in requirements engineering: a systematic review of the literature. Requir. Eng. **21**, 405 (2016)
29. Gašević, D., Kaviani, N., Milanović, M.: Ontologies and software engineering. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies. IHIS, pp. 593–615. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-92673-3_27
30. Couto, R., Ribeiro, A.N., Campos, J.C.: Validating an approach to formalize use cases with ontologies. In: Proceedings of the 13th International Workshop on Formal Engineering Approaches to Software Components and Architectures, vol. 205, pp. 1–15 (2016)
31. Yuan, X., Tripathi, S.: Combining ontologies for requirements elicitation. In: IEEE International Model-Driven Requirements Engineering Workshop, Ottawa, ON, pp. 1–5 (2015)
32. Bagiampou, M., Kameas, A.: A use case diagrams ontology that can be used as common reference for software engineering education. In: 6th IEEE International Conference Intelligent Systems, Sofia, pp. 035–040 (2012)
33. Dzung, D.V., Ohnishi, A.: Ontology-based reasoning in requirements elicitation. In: 2009 7th IEEE International Conference on Software Engineering and Formal Methods, pp 263–272 (2009)
34. Portugal, R.L.Q., Engiel, P., Pivatelli, J., do Prado Leite, J.C.S.: Facing the challenges of teaching requirements engineering. In: IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), Austin, TX, pp. 461–470 (2016)
35. Nkamaura, T., Tachikawa, Y.: Requirements engineering education using role-play training. In: IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), Bangkok, pp. 231–238 (2016)

36. Biddle, R., Noble, J., Tempero, E.: Role-play and use case cards for requirements review. In: Proceedings of the 12th Australasian Conference on Information Systems (2012)
37. Costain, G., Mckenna, B.: Experiencing the elicitation of user requirements and recording them in use case diagrams through role play. J. Inf. Syst. Educ. **22**(4), 367–380 (2011)
38. Kumar, B.S., Krishnamurthi, I.: Improving user participation in requirement elicitation and analysis by applying gamification using architect's use case diagram. In: Vijayakumar, V., Neelanarayanan, V. (eds.) Proceedings of the 3rd International Symposium on Big Data and Cloud Computing Challenges (ISBCC – 16'). SIST, vol. 49, pp. 471–482. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30348-2_39
39. Costain, G.: Cognitive support during object-oriented software development: the case of UML diagrams. Doctoral thesis. Auckland University, New Zealand (2008)
40. Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29044-2
41. Corbin, J.M., Strauss, A.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, 3rd edn. SAGE Publications, Thousand Oaks (2008)
42. Kalinowski, M., Card, D.N., Travassos, G.H.: Evidence-based guidelines to defect causal analysis. IEEE Softw. **29**, 16–18 (2012)
43. de Souza, A.J., Cavalcanti, A.L.O.: Visual language for use case description. Softw. - Pract. Exp. **46**(9) 1239–1261 (2016)
44. El-Attar, M., Miller, J.: Producing robust use case diagrams via reverse engineering of use case descriptions. Softw. Syst. Model. **7**(1), 67–83 (2008)
45. Gregolin, R.: Uma proposta de inspeção em modelos de caso de uso. São Paulo. Dissertação (Mestrado em Engenharia de Computação) – Instituto de Pesquisas Tecnológicas do Estado de São Paulo, 108 p. (2007)
46. Jebril, E.M., Imam, A.T., Al-Fayuomi, M.: An algorithmic approach to extract actions and actors (AAEAA). In: Proceedings of the International Conference on Geoinformatics and Data Analysis, Prague, Czech Republic, 20–22 April (2018)
47. Sawant, K.P., Roy, S., Parachuri, D., Plesse, F.: Enforcing structure on textual use cases via annotation models. In: ISEC 2014 Proceedings of the 7th India Software Engineering Conference, Chennai, India, 19–21 February (2014)
48. Rago, A., Marcos, C., Diaz-Pace, J.A.: Identifying duplicate functionality in textual use cases by aligning semantic actions. Softw. Syst. Model. **15**(2), 579–603 (2016)
49. Deeptimahanti, D.K., Sanyal, R.: Semi-automatic generation of UML models from natural language requirements. In: Proceedings ISEC 2011 the 4th India Software Engineering Conference, Thiruvananthapuram, Kerala, India, 24–27 February, pp. 165–174 (2011)
50. Liu, S., Sun, J., Xiao, H., Wadhwa, B., Dong, J.S., Wang, X.: Improving quality of use case documents through learning and user interaction. In: 21st International Conference on Engineering of Complex Computer Systems (ICECCS), Dubai, 2016, pp. 101–110 (2016)
51. Ko, D., Kim, S., Park, S.: Automatic recommendation to omitted steps in use case specification. Requir. Eng. (2018)
52. Ochodek, M., Koronowski, K., Matysiak, A., Miklosik, P., Kopczyńska, S.: Sketching use-case scenarios based on use-case goals and patterns. In: Madeyski, L., Śmiałek, M., Hnatkowska, B., Huzar, Z. (eds.) Software Engineering: Challenges and Solutions. AISC, vol. 504, pp. 17–30. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-43606-7_2
53. Issa, A.A., Alali, A.I.: Automated requirements engineering: use case patterns-driven approach. IET Softw. **5**(3), 287–303 (2011)
54. Silva, A., et al.: Patterns for better use cases specification. In: Proceedings EuroPLOP' 2015. Hillside Europe (2015)

55. Dias, F., Schmitz, A., Campos, M., Correa, A., Alencar, A.: Elaboration of use case specifications: an approach based on use case fragments. In: ACM Symposium on Applied Computing (SAC), Fortaleza, Ceará, Brazil, pp. 614–618 (2008)
56. El Miloudi, K., Ettouhami, A.: A multiview formal model of use case diagrams using Z notation: towards improving functional requirements quality. J. Eng. **2018**, 9 (2018). Article ID 6854920
57. Holger, E.: Automatic layout of UML use case diagrams. In: SoftVis 2008 Proceedings of the 4th ACM symposium on Software visualization, Ammersee, Germany, 16–17 September, pp. 105–114 (2008)
58. Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika **52**, 591–611 (1965)