Reducing the challenges faced when learning use cases by using inspection

Elizamary de Souza Nascimento, Adriana Lopes, Williamson Alison Freitas Silva, Igor Steinmacher*
Tayana Uchôa Conte
{elizamary.souza, adriana, williamson.silva, tayana}@icomp.ufam.edu.br
*igorfs@uftpr.edu.br

USES Research Group
Federal University of Amazonas (UFAM)
Institute of Computing (IComp)
Manaus – AM, 69077-000

USES Technical Report
TR-USES-2018-0008
Abril 2018

Institute of Computing (IComp)
Federal University of Amazonas (UFAM)
Manaus, Amazonas 69077-000
URL: https://uses.induspam.com/wp-content/uploads/2018/08/TR-USES-2018-0008.pdf

Reducing the challenges faced when learning use cases by using inspection

Elizamary de Souza Nascimento, Adriana Lopes, Williamson Alison Freitas Silva, Igor Steinmacher*, Tayana Uchôa Conte
{elizamary.souza, adriana, williamson.silva, tayana}@icomp.ufam.edu.br
*igorfs@uftpr.edu.br

USES - Grupo de Pesquisa em Usabilidade e Engenharia de Software
Programa de Pós-Graduação em Informática
Universidade Federal do Amazonas
Manaus AM, 69077-000

Universidade Federal do Amazonas, Programa de Pós-Graduação em Informática. RT-USES-2018-0008
Abril 2018

Abstract: Use Cases (both diagram and textual description) are widely recognized and used as a means for capturing system's requirements. However, software engineers and students have difficulty in understanding the syntax and semantics of these models. These difficulties may affect the final quality of the software, since these models may incorrectly and incompletely represent the system. To improve the quality of these models, software inspection assists in the process of reviewing and detecting defects in software artifacts. This paper describes an empirical study carried out with the goal of understanding the initial difficulties perceived by students during the creation of these models. Furthermore, we analyzed the perception of the studies on how the use of the inspection activity can help gaining a better understanding of use cases. The quantitative results showed that the main identified defects in the use cases were: omission, incorrect fact and ambiguity. The qualitative results showed that the main difficulties perceived by the students were related to the semantic understanding of the use cases. We also noticed that the inspection activity assisted the teaching of use cases, since some difficulties with regards to the semantics of use cases were better understood by the students.

Keywords: Use Case; Inspection; Empricial Study

1. Checklists Used in the Inspection of Documents Generated in Assignment 1 (A1)

To evaluate the UC diagrams, textual description of UCs and the consistency of these documents (Diagram vs Textual Description vs Requirements) generated in assignment A1, we prepared three inspection checklists. The checklist technique was chosen because it is one of the most applied techniques to inspect software documents (Anda et al., 2009). The checklists were elaborated based on the type of defects proposed by Travassos et al. (1999) (see Appendix). In addition, for each generated checklist, we created items for each type of defect based on the check items by Anda et al. (2009). We refer to these items by the initials of the defect type including a numerical sequence. For example, AMB_01 refers to the first item (01) related to the Ambiguity (AMB) defect type OMI_03 refers to the third item (03) related to the Omission defect type (OMI), and so on. The purpose of the checklists is to detect defects in (1) UC diagrams; (2) in the textual description of UCs; and (3) the consistency between diagrams, textual description of UCs and requirements (see Table 1). This last checklist was based on Travassos et al. (1999), was called Vertical Inspection (VI).

The items in each checklist are specific to the model(s) being inspected. For example, in the use case diagram checklist, items in the Omission defect type will check if there is a missing actor or use case that should be included, as listed in the system requirements. In the checklist on the textual description of use cases, the items of the same type of defects (Omission) verify if there was a lack of description in the use case of any information or functionality that should be fulfilled, according to the list of system requirements. The checklist for the vertical inspection verifies if there was a lack of description in the use case or diagram of any information or functionality that should be met according to the list of requirements (Omission). We highlight that two researchers reviewed the three checklists before they were delivered to students for execution of assignment A2.

Table 1. Checklist Example for: UC Diagram, Textual Description of UC and Vertical Inspection.

Type of Defect	**Code**	**Items**	**Checklist**
Omission	OMI_03	Have all system functionalities been represented in the diagram?	**UC Diagram**
	OMI_03	Did you have to describe some event flow (alternative and / or exception) necessary for the completeness of the use case?	**Textual Description of UC**
	OMI_01 OMI_05	Did you identify any actors in the use case diagram that are described in the list of requirements? Did you identify the extension or inclusion relationships in the textual description of use case according to the diagram?	**Vertical Inspection**
Incorrect Fact	INF_04	Is the direction of the arrow used to represent the relationship between the actors (Generalization) correct?	**UC Diagram**
	INF_01 INF_04	Does the use case name correctly express its purpose? Are the business rules' descriptions complete and correct?	**Textual Description of UC**
	INF_01	Is there any actor / use case included in the use case diagram that is not correct according to the list of requirements?	**Vertical Inspection**
Inconsistency	INC_02	Is the association of an actor with a use case consistent with the context of the actor's task?	**UC Diagram**
	INC_02	Is the use case description consistent with the use case behavior?	**Textual Description of UC**
	INC_01	Is the use case diagram consistent with the requirements list?	**Vertical Inspection**
Ambiguity	AMB_01	Are the actors' and / or use cases' names clear or can they lead to double interpretation?	**UC Diagram**
	AMB_01	Do the actors' names reflect their roles? Or are they ambiguous (they can lead to double interpretation)?	**Textual Description of UC**
	AMB_01	Does the use case diagram clearly show the actors and functionalities that have been described in the requirements list?	**Vertical Inspection**
	EIN_01	Are the actors part of the system context?	**UC Diagram**

Type of Defect	**Code**	**Items**	**Checklist**
Extraneous Information	EIN_03	Are the described actors or use cases part of the use case context?	**Textual Description of UC**
	EIN_01	Is there an actor and / or use case in the diagram that is not described in the list of requirements? That is, are there actors and / or use cases that were not part of the context and were included in the diagram?	**Vertical Inspection**

2. Qualitative Results

For the qualitative analysis of the difficulties cited by the students, we applied some coding procedures (Strauss and Corbin, 1998), with the help of the Atlas.ti[1]tool. We have analyzed the comments described by the students about the difficulties reported during the modeling of UC (diagram and textual description). The data extracted from each questionnaire response were analyzed and we created relevant codes about students' perceptions regarding the use of UC models (diagram and textual description) in software development. From the codes, we create the categories and relationships between the codes. The coding process was reviewed by a researcher with 15 years of experience teaching modeling and 7 years of experience with qualitative analysis. Quotations and codes have been discussed and associated in different categories and subcategories. Through the discussion of these codes, we analyze students' perceptions regarding use case modeling. We also analyzed the students' perception of the inspection activity as support in teaching use cases. We highlight that since all questionnaires were answered in the native language of the students, the quotations presented in this paper are translations of the subjects' answers excerpts into English that corroborate our findings.

2.1 Students' Perception of UC Modeling (Diagram and Textual Description)

From the qualitative results, we identified the category "**Perception on UC Models in software development**". In this category, we evidenced that UC modeling can "Facilitates understanding of the system and communication between stakeholders". To illustrate this code, the student S5 mentioned that the UC model *"helps who will program to have an idea of how the system behavior should be in face of the needs of the actors".* Another student S32 mentioned that the use case model can help *"... to estimate the effort required for development".* To add, the student S10 complemented saying that *"... the diagram alone is not a powerful tool, however when coupled with the specification (of UCs) not only I can decide the direction of the functionalities as I also know what each diagram will become in the code later".*

[1] Atlas.ti± http://www.atlasti.com

2.2 Students' perception of the difficulties during the Modeling of the UC Diagram

For this perception we identified the category “**Difficulty in the UC Diagram**” and four subcategories: “**Using the extend and include relationships**”; “**Using generalization among actors**”; “**Identifing the tasks of the system actors**” and “**Identifing the actors of the system**”.

Regarding the subcategory “**Using the extend and includes relationships**”, we created the code “<u>Using extend and include for different actors</u>”. Related to this code, the student S22 mentioned that his difficulty was: *“for different actors, I'm not sure how to use existing relationships when there are tasks that are not common to both but need some extend / include one from the other”*. Student S27 also mentioned that he had difficulty using this type of relationship because *“I got confused a little with includes and extends [relationships]”.*

In the subcategory “**Using generalization between actors**”, we created the codes “<u>Confusing the actors that inherit the other actor's functionalities</u>”, “<u>Identifing the generalization relation between the actors</u>” and “<u>Using the direction of the arrow in generalization</u>”. Regarding the first code, the student S11 mentioned that *“I often confused myself with which actor would inherit the functionalities of a certain actor”.* To add, the student S8 commented that *“Understand which actors do the same function what differentiates each one of them in the system”.* In the second code, the student S6 commented that *“the main questions I had was when it came to verify whether a generalization of actors was really necessary and, if so, to create a more general actor who described well the features used by him”*. In the third code, the student S10 commented that: *“I always forget the direction of the generalization arrow: does it point from specific to general or from general to specific?”*. Finally, the student S30 confirmed the same difficulty saying, “*In the case of generalization, I had difficulty understanding the generalization / specialization pattern (arrow direction)*”.

In the subcategory “**Identifing the tasks of the system actors**”, we created the following codes: “<u>Granularity of Use Cases</u>” and “<u>Identifing the system functionalities when the requirements are not explicit</u>”. In the first code, the student S23 mentioned that *“even knowing the functionalities and actors, how to represent them, and divide them into UCs”.* In the second code, the student S21 mentioned that *“my greatest difficulty is to look at requirements that are not ‘explicit’. Which I think has more to do with the requirements survey itself”.*

2.3 Students' perception of the difficulties during the Textual Description of UC

From the qualitative results, we identified the category “**Difficulty in Specifying UCs**” and subcategories, such as: “**Including dependency on other UCs**”, “**Referencing Business Rules and Flows**” and “**Identifying Business Rules**”, “**Identifying and organizing Flows**” and “**Abstracting the Requirement**”.

In the subcategory "**Including dependency on other UCs**", we created codes that reported difficulties in including dependencies on other UCs, for example, the code "It know where to include dependencies of other UCs in the specification". To this respect, the student S12 mentioned "*I do not know if include / extend can be done anywhere in the specification or just in the beginning*". To add, the student S13 mentioned *"I had difficulty knowing where to reference an extend or include in the UC specification".*

Another code is related to "Understanding and describing when the use case relates to others". Student S29 stated: *"I had this difficulty of understanding and describing the use case mainly in use cases that were related, because at first, I did not understand very well how it would work, there was much doubt about what would make the step that would come after this step of the relationship in the flow. I could not understand the relationship".*

With respect to code "Confusing Describes Dependencies of Other UCs as Flow", the student S10 commented that *"the difficulty was to understand whether a UC that extends another or should be described as alternative flow in UC specification of extended, as long as it was identified with extend association in the diagram, or if, in addition to being identified with extend association in the diagram, it should be described as another UC in the specification and referenced in the extended UC flows"*. Finally, in the code "Confusing if extend or include as a dependency or another UC", the student S25 stated that "*sometimes confused whether it should be a separate dependency or a separate UC".*

Regarding the subcategory **"Referencing Business Rules and Flows",** we created the code "Referencing Business Rules in UC Flows". To this respect, the student S5 mentioned that *"it was unclear whether business rules were only referenced along with exception streams or could also be in the main and alternative flows, along with reference to exception flow".* In the code "References Flow, Business Rules and dependencies of other UCs are similar", the student S4 reported that *"...things sometimes seem to be somewhat similar, so I believe they can be streamlined for a better understanding and execution of themselves".* In the code "Referencing flows in UC", the student S26 mentioned: *"the main question was how to reference streams that can happen at any time during the execution of the UC".*

In the subcategory **"Identifying Business Rules",** the codes "Identifying and describing the Business Rules in the UC" and "Confuse on-functional requirements with Business Rules" were created. In the first code, the student S11 commented that *"Sometimes the business rule is not very clear, and I do not know how to describe it".* Student S16 added that *"Not all requirements present business rules clear, which leads to forgetting in the elaboration of the specification".* In the second code, the student S30 stated that *"I sometimes have doubts about whether I should write nonfunctional requirements such as business rules for some UCs or just list them in the vision document".*

In the subcategory **"Identify and organize flows",** codes were two codes. First, created that reported difficulties during the description of use case flows, such as the "<u>Identify and describe flows in the use case</u>" code. Related to this code, the student S13 mentioned that *"at first he had difficulties in understanding when something was an alternative flow. I had difficulty specifying CRUD''s at the outset, and their alternative flows".* The second code is related to "<u>Organize flows in the use case</u>". According to student S9, he had *"difficulty knowing how to organize to know, what was in my mind, fit in exactly what flow".* Student S32 added that *"in flows like CRUD, where we have several possible paths, it is confusing to know what will be the main and what will be the alternatives".*

Finally, in the subcategory **"Abstracting Requirement",** we create the code "<u>Understand the requirement to describe the UC</u>". To this respect, the student S1 stated that *"The only difficulty was in understanding the requirement and describing it in the use case...".* Student S2 also commented that *"I had difficulty in understanding the problem, in very long texts with enough requirements I was in doubt of what really needed to be modeled. A lot of it for detecting what actually added value to the modeling".*

2.4. Students' Perception of Inspection as Support for Learning UCs

In the category "**Inspecting UC Models helps learning UCs**", we identify four main codes that were associated with the fact that the inspection activity helped learning UCs. For example, "<u>It helped to see other ways to specify and improve the specification of UCs</u>", "<u>It helps to improve Modeling and Describing the UCs</u>" "<u>It clarified questions about the Modeling of UCs</u>", and "<u>Inspecting the Diagram Helped Revise its Concepts</u>".

Regarding the first code (<u>It helped to see other ways to specify and improve the specification of UCs</u>), the student S6 stated that "*It (inspection activity) helped a lot to understand how to analyze the specifications without being its author... I even came to notice ways in which I could have made my own specifications*". Student S22 add saying that *"It helped to better visualize ways to formalize and specify the UCs related to the project functionalities and to verify which tend to be common / frequent defects that can be prevented with more attention and care when writting the specification".* In addition, the student S21 added that by inspecting it is possible to *"...I find errors that are often overlooked when modeling and come up with great ideas for possible future modeling*".

In the second code (<u>It helps to improve Modeling and Describing the UCs</u>), the student S7 mentioned that *"...even though we think we already know how to read in theory, many questions arise when practicing that were solved in the class and should no longer occur in moments when it is* *necessary to make a diagram or specification for other people"*. The student S14 indicated that inspecting helped him learned: *"with the practice of 'inspecting' I learned to make better diagrams and specifications"*.

In the third code (<u>It clarified questions about the Modeling of UCs</u>), we found evidence that the inspection was useful to better understand the UC model, as mentioned by the student S12: *"the inspection clarified doubts about the modeling and made me know criteria to evaluate if a modeling is good or if it needs to be corrected"*.

Finally, in the fourth code (<u>Inspecting the Diagram Helped Revise its Concepts</u>), the student S4 mentioned that the inspection helped to recall remember the concepts: *"...It was a concept I already knew and applied in some of my projects, but it was good to review and hear more formal explanations about it"*.

In addition, we created codes that show that inspection helped to identify defects and avoid future errors in upcoming modeling, such as the "<u>It helped to identify defects and avoid repeating the mistakes of others</u>" code. To this respect, the student S5 stated that the inspection *"...helped remind some of the main problems that may occur and in this way, I can avoid them in the future"*. The student S15 added saying that *"... I identified mistakes made by me when I made my [model]. These mistakes help us remember how to avoid them"*. The student S24 commented that the inspection *"made me realize some mistakes I might have made in my specifications and made me think better of how to describe them next time, in a way that would precisely prevent me from making the ambiguities that I found in the descriptions I evaluated"*.

We evidenced that the checklists used during the inspection helped students identify defects and see other types of defects, as commented by student S28: *"the errors I identified in the inspection of the diagram and vertical inspection were detected through the checklist items that indicated mostly known defects. But some helped to see non-trivial defects"*. We also evidenced that learning by observing the errors of the inspected use case models was a positive way to learn the concepts as mentioned by the student S30: *"...the inspection can show us what not to do, which in my opinion is the best way to learn something new, that is, learning the basis of mistakes for me is better"*. Student S16 stated that the inspection *"helped because we learned from the mistakes of others and focus on the main mistakes..."*

The qualitative results show that inspecting the UC diagram, textual description and vertical inspection (UC diagram vs textual description of UCs vs Requirements) can help learning UCs. The results indicate that the students had a better understanding of the taught concepts (UCs) and that the inspection activity assisted in this process. The students reported that the inspection helped to remember concepts and how to better create UC diagrams and how to better describe the textual descriptions of

UCs. The inspection activity improved the students' understanding of defects that should be avoided during the construction of the diagram and the textual description of UCs. In addition, we identified that the use of checklists guided students to systematically identify defects in the models.

3. Discussion

In this section, the research questions are answered, and the results of the empirical study are discussed based on the analysis presented in the previous section.

To answer research question **RQ1** (*What are the main difficulties of students during the modeling of use cases (diagram and textual description?*), we present Fig. 6 with a graphical representation that summarizes the difficulties perceived by students during the modeling of the diagram and textual description of UCs, presenting a model with the perceived difficulties during the creation process of UCs. These difficulties are organized as follows:

- **UC Diagrams:** (D-I) using of relationships include and extend; (D-II) using of relationships generalization between actors; (D-III) identifing the tasks of each actor; and (D-IV) identifing the actors in the system.
- **UC Specifications**: (S-I) including the dependency of other UCs; (S-II) referencing business rules and flows; (S-III) identifing business rules; (S-IV) identifing and organizing flows; and (S-V) abstracting the requirements.

In a previous study, we developed the first version of a model of use case difficulties based on the analysis of qualitative data (Nascimento et al., 2017). In this paper, we present a second version of the difficulties model, highlighting the perceived difficulties in the modeling of diagrams and in the textual description of UC, because in the first version of the model these difficulties (from the UC diagram) were not identified. In order to develop the model a few steps were followed. First, we draw the difficulties from the results of the qualitative analysis, such as the difficulties mentioned by students S11, S3 and S30, respectively: "*confusing the actors who inherit the other actor's functionalities*", "*identifying the generalization relationship among the actors*" and "*use the correct direction of the arrow in a generalization*". These difficulties were related to the modeling of UC diagrams. Next, we identified the categories that were related to these difficulties. For example, the subcategory "(D-II) Using generalization among actors" groups the difficulties mentioned above as shown in Fig. 6. We highlight that all difficulties and categories of the model were reviewed together with three other researchers.

This model of difficulties aims to: (a) assist students and software development practitioners who employ UCs in the academy and / or industry; (b) indicate improvement opportunities in guidelines, techniques and approaches that assist software practitioners in modeling UC diagrams and writing textual descriptions of UCs; and (c) assist researchers in the development of techniques for teaching UCs.

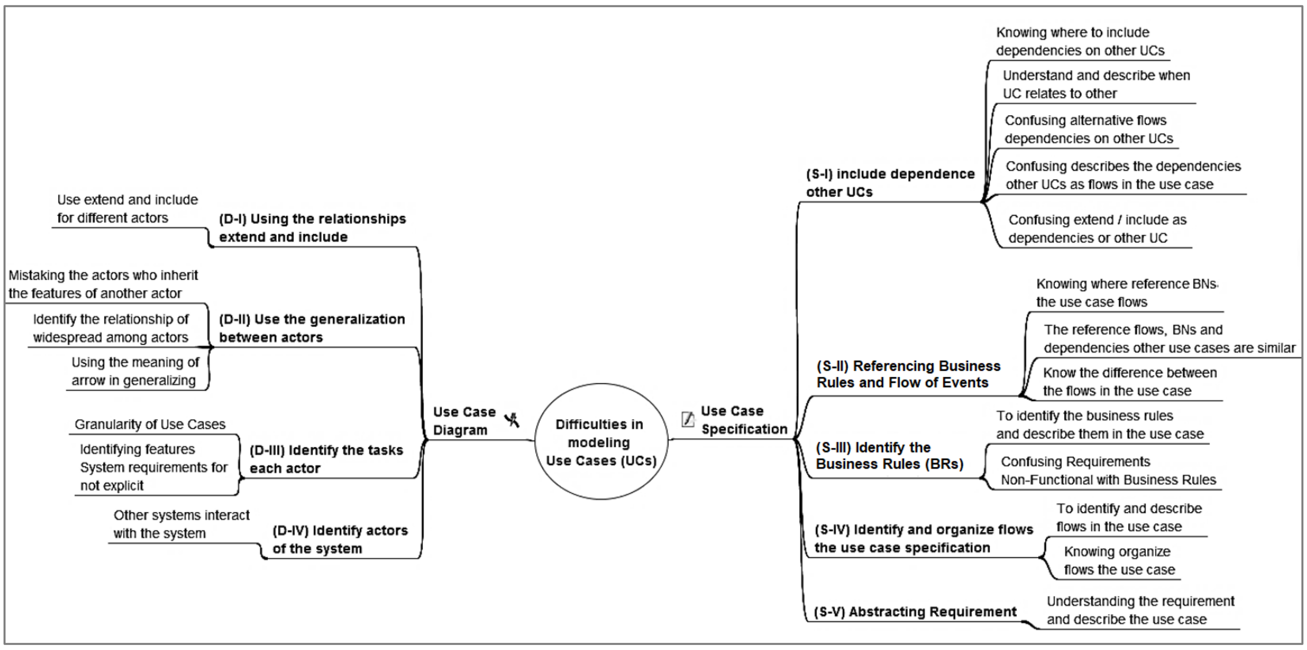
Difficulties in modeling Use Cases (UCs)
Use Case Diagram
Use Case Specification
Use extend and include for different actors
(D-I) Using the relationships extend and include
Mistaking the actors who inherit the features of another actor
Identify the relationship of widespread among actors
(D-II) Use the generalization between actors
Using the meaning of arrow in generalizing
Granularity of Use Cases
Identifying features System requirements for not explicit
(D-III) Identify the tasks each actor
Other systems interact with the system
(D-IV) Identify actors of the system
Knowing where to include dependencies on other UCs
Understand and describe when UC relates to other
Confusing alternative flows dependencies on other UCs
Confusing describes the dependencies other UCs as flows in the use case
Confusing extend / include as dependencies or other UC
(S-I) include dependence other UCs
Knowing where reference BNs. the use case flows
The reference flows, BNs and dependencies other use cases are similar
Know the difference between the flows in the use case
(S-II) Referencing Business Rules and Flow of Events
To identify the business rules and describe them in the use case
Confusing Requirements Non-Functional with Business Rules
(S-III) Identify the Business Rules (BRs)
To identify and describe flows in the use case
Knowing organize flows the use case
(S-IV) Identify and organize flows the use case specification
Understanding the requirement and describe the use case
(S-V) Abstracting Requirement

Fig. 6. Model of difficulties in Use Case Modeling.

The model presents the difficulties that were perceived by students during the modeling of the use case diagram and the textual description of use case, generated from assignment A1. When we presented this model of difficulties, we noticed that the difficulties related to the diagram showed that the students did not know how to use the semantics of the diagram correctly, for example: misuse of the include and extend relationships, and omission of the relationships between actor and use cases. These results are similar to the difficulties reported in the study by Boberic-Krsticev and Tesendic (2013). For example, during the UC modeling, they identified that students had difficulties in using include and extends relationships. In the UC descriptions, we noticed that the students did not describe the alternative flows and the business rules, which are the basis for the completeness of the UC. These results are similar to the difficulties that students had in the study carried by Nascimento et al. (2017). In addition, there were omissions of features that were in the list of requirements and were not modeled as use cases that should be represented in the diagram.

To answer research question **RQ2** (*can the activity of inspecting use cases help students better understand use cases?*), we have analyzed the questionnaire responses asking if the inspection helped learning the use cases. In addition, we also analyzed the number of identified defects by the inspections performed by the teams.

We noticed that the inspection checklists for the UC diagram, textual description of UCs and vertical inspection (UC diagram vs textual description of UCs vs Requirements) helped the students detect defects in these artifacts. The students commented that inspecting diagrams and textual descriptionsof UCs was important because they could observe other forms of modeling and they were able to have different views on how to describe a UC. It was possible to observe that most of the teams could detect different types of defects in their inspections. The students were able to see which defects are common and which are simple defects that can be avoided. In addition, we noticed that students identified the unusual defects that are most difficult to detect. With the help of the checklist, they learned to identify these types of defects and how to avoid them.

We highlight that the students commented that they realized what errors they made when inspecting other UC diagrams. In addition, when the students inspected the textual descriptions of UCs in assignment A2, they were able to see the errors they made and other ways to better describe their UC. The defects found by the students during the inspection of the textual descriptions of UCs showed where they had difficulties when constructing their models and how to avoid them in future modelings. In addition, we identified that the inspections helped the students remember the concepts about modeling the diagram and writing the textual description of use casesUCs.

Finally, the inspection presented students with insights about the defects that can occur in use cases, how to avoid them in the future, and how to improve their models. We observed that the verification items related to the types of defects assisted the students in learning UCs. This leads us to a indications that the activity of inspecting in diagrams and textual description of UCs with checklist can help students in the process of learning UCs.

4. Appendix

Table 1. Checklist for UC Diagram

Category	Code	Check items
Omission	OMI_01	Are the actors (key users and systems) performing the key software tasks identified?
	OMI_02	Need to identify other systems that will communicate with the system to be built? Are they identified as actors?
	OMI_03	Have all system functionalities been represented in the diagram?
	OMI_04	Did not identify any functionality that will be executed (automatically / programmed) by the software?
	OMI_05	Did you include any association of an actor with the use case in the diagram?
	OMI_06	Need to include the name of the association of a use case with another use case (include or extend)?
	OMI_07	Did you include an actor who inherits the profile of another actor but doing something else in the system? (Generalization)
Incorrect Fact	INF_01	Does the use case name correctly express the goal that the actor will accomplish in the system?
	INF_02	Are actors' names and use cases unique?
	INF_03	Are there use cases that have been broken into small parts but which are not correct in isolation? That is, should they be clustered in a use case to maintain full functionality?
	INF_04	
	INF_05	Is the direction of the arrow used to represent the relationship between the actors (Generalization) correct?
	INF_06	Is the direction of the arrow used to represent the extension (extend) association of another use case correct?
Inconsistency	INC_01	Is the use-case name consistent with the purpose of the features that will be performed by the actors?
	INC_02	Is the association of an actor with a use case consistent with the context of the actor's task?
Ambiguity	AMB_01	Are the actors' and / or use cases' names clear or can they lead to double interpretation?
	AMB_02	Is there an actor who has the same role in the system with a different name?
	AMB_03	Is there a use case that performs the same functionality?
Extraneous Information	EIN_01	Are the actors part of the system context?
	EIN_02	Are use cases part of the system context?

Table 2. Checklist for Textual Description of UC

Category	Code	Check items
Omission	OMI_01	Did you identify / describe the name of the actor (s)?
	OMI_02	Did you describe in the precondition some condition that is indispensable to start the use case?
	OMI_03	Did you have to describe some event flow (alternative and / or exception) necessary for the completeness of the use case?
	OMI_04	Did you have to describe any business rules necessary for the completeness of the use case?
	OMI_05	Need to reference some business rule, alternative flows or exception streams at some stage of the event flows?
	OMI_06	Need to include dependency on other use cases (include or extend) that are associated with the use case? (according to the use case diagram)
Incorrect Fact	INF_01	Does the use case name correctly express its purpose?
	INF_02	Is the identification of alternative flows (A1, A2, etc.), exception flows (E1, E2, etc.) and business rules (BR1, BR2, etc.) unique?
	INF_03	Is the description of the main streams, alternatives, and exceptions complete and correct?
	INF_04	Are the business rules' descriptions complete and correct?
Inconsistency	INC_01	Is the precondition consistent with the behavior of the use case? Is it really a condition that prevents the use case from starting?
	INC_02	Is the use case description consistent with the use case behavior?
	INC_03	Is sequencing in the main, alternative, and exception streams coherent?
Ambiguity	AMB_01	Do the actors' names reflect their roles? Or are they ambiguous (they can lead to double interpretation)?
	AMB_02	Does the use-case name allow for different interpretations of your purpose?
	AMB_03	Is there more than one use case that has the same name?
	AMB_04	Is the description of pre and postconditions (when they exist) clear and unambiguous?
	AMB_05	Is the description of event flows and business rules clear and unambiguous?
	AMB_06	Do the event flows (main, alternative, and exception) follow a logical and clear path?
	AMB_07	Is the information exchanged between the actor and the system clear and well-defined?
Extraneous Information	EIN_01	Is information described in the steps of the event flows (main, alternative and / or exceptions) part of the context of the use case?
	EIN_02	Is the information described in the business rules part of the use case context?
	EIN_03	Are the described actors or use cases part of the use case context?

Table 3. Checklist for Vertical Inspection

Category	Code	Check items
Omission	OMI_01	Did you identify any actors in the use case diagram that are described in the list of requirements?
	OMI_02	Did not identify any use cases in the diagram to meet a functionality described in the requirements list?
	OMI_03	Did you identify the name of the use case (s) in the specification according to the diagram?
	OMI_04	Did you identify / describe the name of the actor (s) in the use case specification according to the diagram and list of requirements?
	OMI_05	Did you identify the extension or inclusion relationships in the textual description of use case according to the diagram?
Incorrect Fact	INF_01	Is there any actor / use case included in the use case diagram that is not correct according to the list of requirements?
	INF_02	Is there a business rule described in the use case specification that is not correct according to the requirements list?
	INF_03	Is there any information described in the event flows in the use case specification that is not correct according to the requirements list?
Inconsistency	INC_01	Is the use case diagram consistent with the requirements list?
	INC_02	Is the use case specification consistent with the diagram?
	INC_03	Is the association to extend the use case (extend) consistent? That is, when the base-use case is run the extended-use case might run as well?
	INC_04	Is the association for inclusion of the use case included? That is, when the base-use case is executed the inclusion-use case will be executed?
Ambiguity	AMB_01	Does the use case diagram clearly show the actors and functionalities that have been described in the requirements list?
	AMB_02	Does the specification of the use case fulfill the functionalities described in the list of requirements clearly and unambiguously?
Extraneous Information	EIN_01	Is there an actor and / or use case in the diagram that is not described in the list of requirements? That is, are there actors and / or use cases that were not part of the context and were included in the diagram?
	EIN_02	Actors or use cases that are not in the diagram and not part of the context have been described in the use case specification?
	EIN_03	Is there a business rule that is not part of the system context, according to the requirements list, but has been described in the use-case specification?