Checklist-based Inspection of SMarty Variability Models
Proposal and Empirical Feasibility Study

Ricardo T. Geraldi[1], Edson Oliveira Jr.[1], Tayana Conte[2] and Igor Steinmacher[3]
[1]*Informatics Department, State University of Maringá, Maringá-PR, Brazil*
[2]*Institute of Computing, Federal University of Amazonas, Manaus-AM, Brazil*
[3]*Department of Computing, Federal Technological University of Paraná, Campo Mourão-PR, Brazil*

Keywords: Coding, Empirical Study, Feasibility, SMarty, SMartyCheck, Software Inspection, Software Product Line, Qualitative Analysis.

Abstract: Software inspection is a particular type of software review applied to all life-cycle artifacts and follows a rigorous and well-defined defect detection process. Existing literature defines several inspection techniques for different domains. However, they are not for inspecting product-line UML variability models. This paper proposes SMartyCheck, a checklist-based software inspection technique for product-line use case and class variability models according to the SMarty approach. In addition, it presents and discusses the empirical feasibility of SMartyCheck based on the feedback from several experts. It provides evidence of the SMartyCheck feasibility, as well as to improve it, forming a body of knowledge for planning prospective empirical studies and automation of SMartyCheck.

1 INTRODUCTION

A Software Product Line (SPL) encompasses a set of products for a particular market or mission (Capilla et al., 2013). The SPL concept has been consolidated over the last years as a non-opportunistic reuse strategy (Linden et al., 2007; Capilla et al., 2013).

Nowadays, several organizations handle huge amounts of information, thus depending on software applications for a competitive market (Capilla et al., 2013). Therefore, such organizations need non error-prone information systems, able to change over specific market rules (Mello et al., 2014).

Software inspection is efficient and mostly adopted by industry (Fagan, 2002; Rombach et al., 2008). The existing literature provides a few recent studies concerned on the application of inspection techniques to SPL variability models (Cunha et al., 2012; Mello et al., 2014). However, such recently proposed techniques are not for inspecting UML models. In this context, it is suitable to apply the Stereotype-based Management of Variability (SMarty) approach (OliveiraJr et al., 2010), in combination with a software inspection technique, for reducing potential defects at modeling SPL SMarty-based variabilities. SMarty has been empirically evaluated in the last few years (Marcolino et al., 2014). An inspection technique that takes as a basis SMarty models become an interesting tool towards defect identification in early life-cycle SPL artifacts with variabilities. Therefore, this paper presents the proposal of SMartyCheck, a checklist-based inspection technique for use case and class SMarty SPL models, as well as its feasibility study as an empirical qualitative analysis.

The remainder of this paper is organized as follows: Section 2 presents the background with regard to SPL variability management and the SMarty approach, software inspection, defect types and existing inspection techniques; Section 3 presents SMartyCheck, the proposed inspection technique; Section 4, presents the SMartyCheck empirical feasibility study; Section 5 discusses related work; and Section 6 presents conclusion and directions for future work.

2 BACKGROUND

This section presents essential concepts on variability management and software inspections.

2.1 The SMarty Variability Approach

A Software Product Line (SPL) is a set of software

T. Geraldi R., Oliveira Jr. E., Conte T. and Steinmacher I..
Checklist-based Inspection of SMarty Variability Models - Proposal and Empirical Feasibility Study.
DOI: 10.5220/0005350102680276
In *Proceedings of the 17th International Conference on Enterprise Information Systems* (ICEIS-2015), pages 268-276
ISBN: 978-989-758-097-0

intensive systems that share a common infrastructure formed by the SPL architecture and essential artifacts (Linden et al., 2007). Such architecture and artifacts have common features, the similarities, and variable elements, the variabilities, in order to provide a means to derive specific products for a given domain. Variability represents how SPL specific products differ one another.

In this context, the existing literature presents several variability management approaches based on feature models, UML and domain specific languages. The Stereotype-based Management of Variability (SMarty) approach has been developed OliveiraJr et al. (2010) and empirically evaluated Marcolino et al. (2014) for UML-based SPLs.

SMarty is composed of a UML profile, the SMartyProfile, and a systematic process, the SMartyProcess, to guide one on applying the SMarty stereotypes and meta-attributes. SMarty version 5.1 supports use case, class, component, activity and sequence models.

2.2 Software Inspection

Software inspection is a specific type of software review (Ciolkowski et al., 2003) applied to artifacts by means of a systematic and well-planned defect identification process (Fagan, 1986). More than 60% of defects can be identified in early phases of the software life-cycle[1] (Boehm and Basili, 2001; Fagan, 1986). By means of software inspection it is possible to verify several elements of software artifacts in order to detect different defect types persistent in software specifications (IEEE, 1998b).

Fagan (1986) proposed a software inspection process driven by roles, such as, moderator, inspector and author and activities. The process of defect detection must be standardized and non-ambiguous for the document under revision, as for instance, a requirements specification (IEEE, 1998a). Therefore, Anda and Sjøberg (2002) propose a taxonomy of defect types for checklist-based software inspections specifically for use cases. Such a taxonomy is composed of the following defect types: Omissions, Incorrect facts, Inconsistencies, Ambiguities and Extraneous Information. In addition, these are the main existing defect types in the literature, as we can see in Travassos et al. (1999). Furthermore, the literature improves such a taxonomy by providing several more studies on defect types, such as in Hayes et al. (2006), IEEE (2012), Mello et al. (2014), Travassos et al. (1999) and Cunha et al. (2012).

Therefore, for defect detection it might be used inspection techniques, thus, the study of Anda and Sjøberg (2002) presents the Checklist-Based Reading (CBR), which is a non-systematic technique for defect detection. It does not provide directions or mechanisms on "how" the inspection should be performed, but "what" must be inspected through guidance using a document in a checklist format (Alshazly et al., 2014).

Furthermore, the CBR technique based on checklist is one of the most adopted inspection techniques for analyzing artifacts, as well as for identifying the most relevant information required for certain review tasks (Mello et al., 2014). The checklist verification items are simple to follow. The checklist format is a list of non-ambiguous questions, based on either previous experiences or history, which must be clearly answered as "Yes" or "No". In order to make the checklist able to be applied, a list of defect types must be available for the inspector (Alshazly et al., 2014).

3 SMartyCheck

The SMartyCheck technique acts at the Domain Engineering activities. SMartyCheck was designed as an inspection technique with the objective of detecting defects in SMarty SPL use case and class variability models of SPL by means of a checklist with 20 items. The SMartyCheck technique is composed of two checklists: one for use case models, and one for class models. We are aware that the automation of SMartyCheck is an important issue to contribute for assuring quality in inspection activities, thus, we are developing a tool for it.

In order to perform inspections, inspectors should read the checklist answering each question from a list of assertive questions, marking a single check per question, based on their prior knowledge with relation to the SMartyCheck checklist defect types.

The checklist was developed based on the analysis and adaptation of defect types extracted from a systematic mapping study that we carried out from February to March/2014. From the analysis and classification of the 51 most relevant retrieved primary studies, it was evident the identification of several different studies with distinct defect types taxonomy proposals, most of them empirically evaluated, that can be adapted by existing software inspection techniques, such as SMartyCheck.

The taxonomy of defect types for the SMartyCheck checklist contains 13 defect types, adapted from the following sources: IEEE (2012), IEEE (1998b), IEEE (1998a), Travassos et al. (1999), Hayes et al. (2006), Lamsweerde (2009) and Cunha et al. (2012).

[1] http://www.cebase.org/defect-reduction.html

Thus, each type of SMartyCheck defect is described as follows, in which is possible to understand "what" to inspect throughout the SMartyCheck checklist for use case and class models:

1. **Ambiguous**: use case or class model elements should have only one interpretation. This defect type was adapted from IEEE (2012) aiming at detecting redundancy;
2. **Anomaly**: use case or class model element is associated to another element in an unusual way. This defect type was adapted from Cunha et al. (2012) to the context of SMarty models in order to detect different elements relationships which are not allowed;
3. **Incomplete**: absence of any significant use case or class model elements. It was adapted from IEEE (2012);
4. **Inconsistency**: refers to the lack of internal consistency, in which a subset of individual elements has conflicts with use case or class models. This defect type was adapted from IEEE (2012);
5. **Incorrect**: any element must meet the predicted relationships of use case or class models. It was adapted from IEEE (2012);
6. **Intentional Deviation**: an use case or class model element that requires or depend on its relationship to other elements according to SMarty variability models. Its descripton was interpreted and adapted from Hayes et al. (2006) to the SMarty-Check context;
7. **Instable**: every use case or class model element has a unique identifier (for instance the meta-attribure name) indicating its importance or stability. Such an identifier specifies an element according to its singularity and responsibility. This defect type was adapted from IEEE (1998b,a);
8. **Incorrect Fact**: contradiction in any use case or class model element. This defect type was interpreted and adapted from Travassos et al. (1999);
9. **Extraneous Information**: non-necessary provided information from any use case or class model elements. Over specified or duplicated elements in use case ou class models should be detected by this kind of defect, adapted from Travassos et al. (1999);
10. **Infeasible**: any unachievable element of use case or class models taking into consideration other system factors, such as, adding new elements in SMarty SPL models. This defect type was adapted from Hayes et al. (2006);
11. **Non-modifiable**: when the structure and the style of one or more elements of use case or class models can be maintained by any changes. The elements should be expressed in separate way in order to be consistent and not redundant. It was adapted from IEEE (1998b,a);
12. **Omission**: absence of any element of use case and class models. This defect ype was interpreted and adapted from Lamsweerde (2009);
13. **Business Rule**: the definition of business rules of a specific domain may be incorrectly described or specified according to SMarty use case and class models. Its description was adapted from Lamsweerde (2009).

We present an application excerpt of how SMarty-Check is applied to an SPL, in this case to the Arcade Game Maker (AGM) (SEI, 2009) use case original model (Figure 1(a)). Therefore, for illustration purpose, we have inserted defects to the AGM use case model (Figure 1(b)).

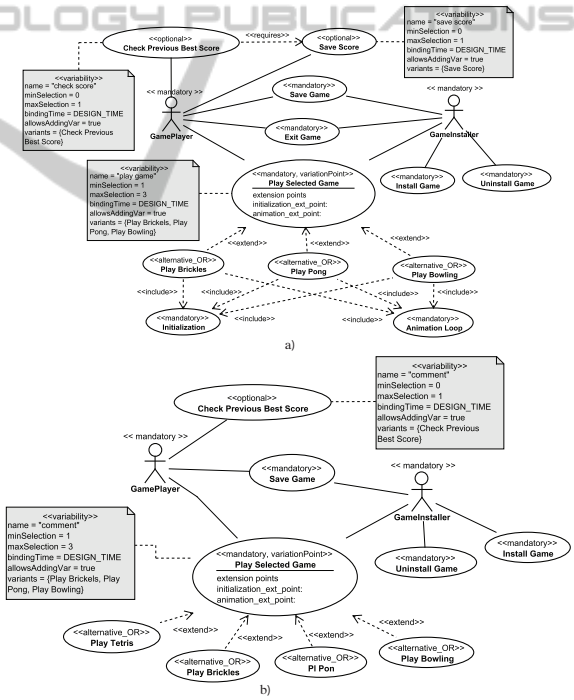

Figure 1: AGM Use Case Model according to SMarty: (a) with no defects (original) (OliveiraJr et al., 2010) and (b) with inserted defects.

The following items present the application of SMartyCheck based on the defined defect types and checklist items answered "Yes":

4 Inconsistency (Incons)

- **Incons.1** Is there any use case in the SPL use case model with the stereotype

<<variationPoint>> whose number of specified variants is larger than defined in maxSelection the variability associated?
Identified Defect: The use case *Play Selected Game* has four variants. Should have at maximum three.

6 Intentional Deviation (ID)

- **ID.1** Is there any use case in the SPL use case model which requires the selection of another (<<requires>>) and this another is not specified in the SPL?
Identified Defect: The use case *Check Previous Best Score* requires the selection of the use case *Save Score*.

8 Incorrect Fact (IF)

- **IF.1** Is there any use case in the SPL use case model which was described incorrectly?
Identified Defect: The use case *Play Pong* was described as *Pl Pon*.
- **IF.2** Is there any use case in the SPL use case model that it is not be mapped its description?
Identified Defect: The use cases *Play Tetris* and *Pl Pon* can not be mapped.

11 Non-modifiable (Nm)

- **Nm.1** Is there any use case in the SPL use case model with the stereotype <<variationPoint>>, in which variants associated (<<optional>>, <<alternative_OR>> or <<alternative_XOR>>) cannot be combined or selected in accordance with the meta-attribute *variants*?
Identified Defect: The use case *Play Selected Game* has selected variants *Play Tetris* and *Pl Pong* erroneously specified.

12 Omission (Om)

- **Om.1** Is there any use case specified as mandatory in the SPL use case model through the stereotype <<mandatory>> that is not specified on the SPL?
Identified Defect: The use cases *Exit Game*, *Initialization* and *Animation Loop* are not specified in the SPL.

13 Business Rule (BR)

- Domain/Business Rules: The system should be able to allow you to install and uninstall a game, allows choosing a to game, save the state of the game, exit the game, define options and provide the player a way to save his score and check your best score previous.
BR.1 The use cases of the SPL use cases model are not clear with the purpose and the desired functionalities based on defined domain?
Identified Defect: The system is not able to allow exit game, define the options of game and provide the player a way to save his score.

4 THE FEASIBILITY STUDY

A qualitative empirical study was carried out aiming at analyzing the SMartyChek feasibility, then improving the proposed technique.

The analysis of this qualitative study was based on the data collected from 8 experts. The majority of the experts are from software engineering with ten years of expertise in average. Selection criteria took into account lecturers and practitioners, who directly worked with or research on software verification and validation over the past years in different universities and companies in Brazil.

The study documents were given the experts, thus aiming to support them while answering the electronic questionnaires to evaluate the feasibility of SMartyCheck. We created 6 electronic questionnaires with 6 defective models, in which 2 questionnaires (1 checklist with 2 use case models and 1 checklist with 2 class models) were distributed and electronically randomly sent to the experts. At the end of the assignment, we asked the experts to answer an open-ended questionnaire composed of 7 questions about the SMartyCheck technique. Experts spent about 90-120 minutes for performing all given tasks of the study.

Then, experts answered the questionnaire specifically related to the characterization of SMartyCheck. Next items present examples of questions:

In your opinion, the inspection process based on the SMartyCheck technique is adequate for detecting defects in use case and class SMarty SPL models? Please, support your answer with justification.

In your opinion, the format of the SMartyCheck checklist clearly, objectively and accurately contributes for inspecting use case and class SMarty SPL models? Please, support your answer with justification.

The answers received were qualitatively analyzed using procedures of Grounded Theory (GT) (Corbin and Strauss, 2008). GT approach is based on the concepts of Coding (Corbin and Strauss, 2008). Coding allows assigning codes or labels to excerpts of text (Open Coding), which can be grouped and categorized (Axial Coding) according to an expressed idea

for the purpose of elucidate the phenomenon (Corbin and Strauss, 2008). The Selective Coding was not performed in this study, as we do not intend to formalize a theory, but evaluate the feasibility of the SMartyCheck technique.

Analysis included only procedures of Open Coding and Axial Coding. As a result of such coding, enabled the creation a conceptual model.

During the coding process, 3 categories emerged from the data collected from the experts regarding feasibility of SMartyCheck technique: "Feasible for Inspection", "Possible Improvements" and "Automating SMartyCheck".

A brief description of each category along with their related codes are presented, as well as excerpts from the data collected from the experts and the categorization of codes to evaluate questions concerning the feasibility of SMartyCheck.

Feasible for Inspection: this category evaluates the feasibility of SMartyCheck, providing evidence that improvements are required. Figure 2 presents a graphical representation of factors that are part of this category.

Analyzing the experts answers with regard to SMartyCheck, such a technique supports the correctness based on the coding Supports Correctness assigned in accordance to the excerpt from expert #1: "*...the process is interesting and confers greater degree of correctness to the model.*"; and the excerpt from expert #3: "*...supporting the correction of the models.*". Furthermore, the SMartyCheck technique detects inconsistencies in SMarty models based on excerpt from expert #6: "*...facilitates inconsistencies mapping.*"; and the excerpt from expert #8: "*...a checklist guides the reviewer at identifying possible inconsistencies of the model.*".

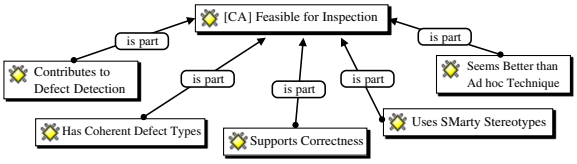

Figure 2: Graphical Representation with the Associations related to the Feasible for Inspection Category.

We evidenced that the SMartyCheck Contributes to Defect Detection. This factor was supported by some quotes from the experts, who answered "*Yes.*" to "***In your opinion, the format of the SMartyCheck checklist clearly, objectively and accurately contributes for inspecting use case and class SMarty SPL models? Please, support your answer with justification.***". Expert #3 also answered "*Yes, because it allowed to detect different defect types in the UML models.*", whereas expert #6 answered "*Yes, it is possible to identify process failures throughout the mapping.*".

In addition, the SMartyCheck technique Has Coherent Defect Types, according to the experts. When asked "***In your opinion, the defects types are coherent with the description of the checklist items of SMartyCheck technique for inspection SMarty SPL models taking into account use case and class models? Justify.***", the expert #3 answered: "*Yes, they are coherent as they allowed to analyze possible errors in different elements and correct such errors.*"; and the excerpt from expert #4: "*Yes, I believe that the defect types are fine segmented.*"; and the excerpt from expert #6: "*I believe so, because they express the problem to be mapped.*"; and the excerpt from expert #7: "*Yes... Defect types are clear and objective. However, they could be grouped and placed on the natural order.*".

Furthermore, several experts reported that SMarty stereotypes are important elements to better performing inspections. Based on this, Uses SMarty Stereotypes factor emerged, supported by the following quotes: "*Yes, the specification of stereotypes is interesting and think their use is essential for the correct reading of the model.*" [Expert #1]; "*Yes, because they represent the key elements of variability, variation points and variants.*" [Expert #3]; "*Yes, stereotypes are essential because they present clear and intuitive elements related to the variability of a SPL.*" [Expert #4]; "*Yes, they are key elements for inspection, since it is also based mostly on these stereotypes.*" [Expert #5].

In addition, SMartyCheck is different in comparison to the *Ad hoc* technique. The Ad hoc is an "informal" technique that does not provide instructions, procedures, criteria or systematic mechanisms for inspectors performing the reading of software artifacts. Therefore, the Ad hoc technique depends on the viewpoint of the inspector, his knowledge and experience in the detection of defects (Rombach et al., 2008). Thus, the SMartyCheck is better in comparison to the Ad hoc based on the coding Seems Better than Ad hoc technique assigned in accordance to the expert answer #8: "*Very different when the reviewer has to do its work performing an ad hoc comparison.*".

Figure 2 presents the representation of the SMartyCheck feasibility by supporting correctness (`Supports Correctness`) of SMarty use case and class models, detecting inconsistencies and defects (`Contributes to Defect Detection`). Furthermore, SMartyCheck is suitable for detecting several different coherent defect types (`Coherent Defect Types`). With regard to the SMarty approach, its

stereotypes are essential and intuitive for defect detection (`Uses SMarty Stereotypes`).

Possible Improvements: SMartyCheck requires possible improvements, because it has some limitations, considering redundancies reported by experts on the items of the checklist.

SMartyCheck has a few redundancies in items of the checklist based on the coding Redundant in the Checklist Items, supported by quotes extracted from answers of expert #4: "*...some questions seemed to me somewhat recurring, which can be considered a threat to the validity of this proposal.*"; and expert #5: "*...I consider the technique with some redundant inspections.*"; and expert #7: "*...there are duplicate and ambiguous questions.*"; and expert #7: "*The items of SMartyCheck: Incons.1, Incom.1, An.1, Incons.3, Incons.2 can be done automatically by syntax analysis of the model.*".

Furthermore, the SMartyCheck technique has confused descriptions related to the items of the checklist. It can be evidenced this Confusion in the Checklist Items on answers from expert #1: "*...some items have caused a little of confusion initially.*"; and expert #3: "*...some of the questions are formulated to hinder understanding.*".

In addition, experts answered that some defect types are incoherent, ambiguous and hard to interpret based on the coding Has Incoherent Defect Types assigned in accordance to the expert answer #1: "*The defect identified as "Non-modifiable" seemed to me confused as to its description.*"; and expert #5: "*Defect #5 (Incor.2), I do not understand why relating 2 mandatory items is incorrect.*"; and expert #7: "*In An.1, it must be clear the type of association, the question leads to an incorrect markup. In Incons.1, it must be clear the implication of this.*".

Automating the SMartyCheck: some experts suggested the automation of some checklist items to improve SMartyCheck technique. Automating SMartyCheck was suggested by expert #4: "*...I believe that automating process of systematic development, as in the case of SPL, can aggregate much in the quality requirement.*".

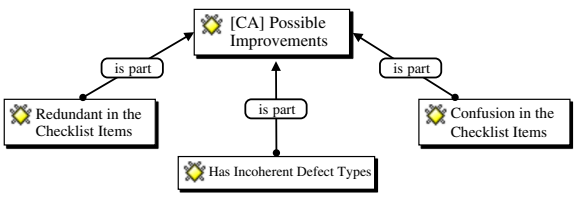

Figure 3: Graphical Representation with the Associations related to the Possible Improvements Category.

Therefore, it is clear that some adjustments are necessary to improve the SMartyCheck technique according to the issues presented under the categories "Possible Improvements" and "Automating SMartyCheck". Figure 3 presents a graphical representation of the factors, which are part of the category "Possible Improvements" and the defect types suggested by the experts that can be automated in prospective improvements related to the category "Automating SMartyCheck".

Figure 3 presents the possible improvements codings of SMartyCheck, which are due to some of the checklist items are redundant (`Redundant in the Checklist Items`) and confusing (`Confusion in the Checklist Items`). Thus, they were improved. Incoherent defects are also present in SMartyCheck as we can see in the `Has Incoherent Defect Types` coding. Such issues that form the `Possible Improvements` coding were reviewed and improved.

The following items present the improved checklist of SMartyCheck. Although such items refers to use case models, they also serve for class models.

1. **Business Rule (BR)**
   BR.1 The use cases of the SPL use cases model (with defects) model are not clear with the purpose and the desired functionalities based on defined domain?
2. **Incomplete (Incom)**
   Incom.1 All use cases with `<<mandatory>>` (required) are not specified in the SPL use case model (with defects)?
3. **Inconsistency (Incons)**
   Incons.1 Is there any use case in the SPL use case model (with defects) specified with the stereotype `<<variationPoint>>` whose number of specified variants is larger (greater) than defined in *maxSelection* the variability (`<<variability>>`) associated?
   Incons.2 Is there any use case in the SPL use case model (with defects) specified with the stereotype `<<variationPoint>>` whose number of specified variants is smaller (minor) than defined in *minSelection* the variability (`<<variability>>`) associated?
4. **Incorrect (Incor)**
   Incor.1 Is there any use case in the SPL use case model (with defects) specified with the stereotype `<<variationPoint>>` which is associated with a use case in SPL (with defects) that is not `<<alternative_OR>>`?
5. **Incorrect Fact (IF)**
   IF.1 Is there any use case in the SPL use case model (with no defects) which was described in-

correctly in SPL (with defects)?
IF.2 Is there any use case in the SPL use case model (with defects) that it is not be mapped its description in SPL (with no defects)?

6. **Ambiguous (Am)**
   Am.1 Is there any use case in the SPL use case model (with defects) that your description does not reflect on your real goal?
   Am.2 Is there any use case in the SPL use case model (with defects) in that its name or description is equal to the use case which is associated to owning a duplicate interpretation?
7. **Non-modifiable (Nm)**
   Nm.1 Is there any use case in the SPL use case model (with defects) specified with the stereotype <<variationPoint>>, in which variants associated (<<optional>>, <<alternative_OR>> or <<alternative_XOR>>) can not be combined or selected in accordance with the variants already specified in the meta-attribute *variants* in SPL (with no defects)?
8. **Anomaly (An)**
   An.1 Is there any use case in the SPL use case model (with defects) specified with the stereotype <<alternative_OR>> associated with a use case that is not specified with the stereotype <<variationPoint>>?
9. **Instable (Ins)**
   Ins.1 Is there any use case in the SPL use case model (with defects) specified with the stereotype <<variationPoint>>, in which has the stereotype associated <<variability>> whose meta-attribute *name* equals to other use cases elsewhere specified with the stereotype <<variationPoint>>?
10. **Infeasible (Inf)**
    Inf.1 Is there any use case in the SPL use case model (with defects) specified with the stereotype <<variationPoint>> whose number of variants speciifed is not (false) allows you to add new variants as defined in the meta-attribute *allowsAddingVar*?
11. **Omission (Om)**
    Om.1 Is there any use case specified as mandatory (required) in the SPL use case model (with no defects) through the stereotype <<mandatory>> that is not specified on the SPL (with defects)?
12. **Extraneous Information (EI)**
    EI.1 Is there any use case in the SPL use case model (with defects) specified besides use cases existing in the SPL (with no defects)?
    EI.2 Is there any use case in the SPL use case model (with no defects) with its functionality duplicated in SPL (with defects)?
13. **Intentional Deviation (ID)**
    ID.1 Is there any use case in the SPL use case model (with no defects) which requires the selection of another (<<requires>>) and this another is not specified in the SPL (with defects)?
    ID.2 Is there any use case in the SPL use case model (with no defects) not which requires selection of another (<<mutex>>) and this another is specified in the SPL (with defects)?

By analyzing the overall obtained results, we can provide evidence that SMartyCheck is feasible, as it: (i) supports correctness of models; (ii) explicitly contributes for defect detection; (iii) has coherent defect types; (iv) correctly uses SMarty stereotypes, making such a detection intuitive and objective; and (v) seems to be more appropriate than the ad hoc technique.

The following major threats to the validity of this empirical study are discussed:

Threats to Internal Validity. Tasks performed by the experts were conducted in a similar manner, except for the electronic forms (questionnaires) randomly mailed. Experts were trained on the basics of SPL, variability and software inspection, beyond the understanding and comprehension of the SMartyCheck. 20 to 27 days was required to the experts fill in the questionnaires, thus, fatigue effects were reduced;

Threats to External Validity. Pedagogical SPLs used in training sessions (SPL Mobile Media) and in the empirical study (SPL Arcade Game Maker (AGM)) can endanger the external validity. Therefore, we tried to use SMartyCheck and AGM standardized technical documents. Obtaining practitioner experts from industry was one of the greatest difficulties encountered in this study;

Threats to Conclusion Validity. The main threat is related to the sample size, 8 experts, however the previous knowledge of the experts was significant. Therefore, we intend to perform more empirical studies with a larger sample;

Threats to Construct Validity. This study was defined and tested based on a pilot project carried out, refining the SMartyCheck and AGM instrumentation, which was adequate to apply to a real study. Although the level of knowledge required on the concepts of UML modeling, SPL and software inspection is basic, experts presented high-level skills.

5 RELATED WORK

The set of techniques "The Software Product Line Inspection Techniques (SPLIT)" by Cunha et al. (2012) is aimed at comparing the SPL requirements document, feature model and product map. SPLIT has contributed to the definition of defect types, adopted by SMartyCheck. The study conducted by Anda and Sjøberg (2002) also has. Such a study presents a taxonomy of typical defects of use case models based on two empirical studies.

The study of Belgamo et al. (2005) proposes TUCCA, a "Technique for Use Case Model Construction and Construction-based Requirements Document Analysis", which is composed of two reading techniques. In such a work, it was identified new defect types in experiments that might contribute to improve SMartyCheck. These defect types are not explicitly mentioned in the study of Belgamo et al. (2005).

6 CONCLUSION

This paper presented the proposal of SMartyCheck technique for inspecting SMarty use case and class models, as well as an empirical qualitative study, which provided evidence that SMartyCheck is feasible. Such a technique contributes to SPL model inspections by means of coherent defect types, leading to correctness of models based on a well-defined structure supported by SMarty stereotypes.

In fact, the qualitative results helped to improve SMartyCheck since it was possible to identify defects that require adjustments and correcting the checklist items considered ambiguous and difficult to understand. Furthermore, the feedback from subjects provided evidence that SMaryCheck is feasible, because stated support correctness SMarty SPL models. Further empirical studies must be planned and conducted in academia and industry to evaluate the accuracy of SMartyCheck.

As future work, we plan to: (i) allow the inspection of different SMarty models, such as component, sequence and activity; (ii) automate, partially or totally, the inspection process by means of a tool for sintax analysis of SMarty models; (iii) combine SMartyCheck and perspective-based techniques to increase its inspection power; and (iv) conduct more empirical studies to refine SMartyCheck, as well as corroborate and generalize the obtained results.

ACKNOWLEDGEMENTS

CAPES-Brasil for granting R. Geraldi a two-year master's degree scholarship. **Experts** from several universitites.

REFERENCES

Alshazly, A. a., Elfatatry, A. M., and Abougabal, M. S. (2014). Detecting defects in software requirements specification. *Alexandria Engineering Journal*.

Anda, B. and Sjøberg, D. I. K. (2002). Towards an inspection technique for use case models. In *Proc. Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, pages 127–134, New York, NY, USA. ACM.

Belgamo, A., Fabbri, S., and Maldonado, J. C. (2005). TUCCA: Improving the Effectiveness of Use Case Construction and Requirement Analysis. In *Proc. Int. Symposium on Empirical Software Engineering (ESEM)*, pages 257–266. IEEE.

Boehm, B. and Basili, V. R. (2001). Software Defect Reduction Top 10 List. *IEEE Computer*, pages 135–137.

Capilla, R., Bosch, J., and Kang, K.-C. (2013). *Systems and Software Variability Management: Concepts, Tools and Experiences*. Springer Berlin Heidelberg.

Ciolkowski, M., Laitenberger, O., and Biffl, S. (2003). Software Reviews: The State of the Practice. *IEEE Software*, 20(6):46–51.

Corbin, J. M. and Strauss, A. L. (2008). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, 3 edition.

Cunha, R., Conte, T. U., de Almeida, E. S., and Maldonado, J. C. (2012). A Set of Inspection Technique on Software Product Line Models. In *Proc. Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, pages 657–662.

Fagan, M. (2002). *A History of Software Inspections*. Springer Berlin Heidelberg.

Fagan, M. E. (1986). Advances in Software Inspections. *IEEE Transactions on Software Engineering (TSE)*, pages 744–751.

Hayes, J., Raphael, I., Holbrook, E., and Pruett, D. (2006). A Case History of International Space Station Requirement Faults. In *Proc. Int. Conf. on IEEE Engineering of Complex Computer Systems (ICECCS)*, page 10. IEEE Computer Society.

IEEE (1998a). Recommended Practice for Software Requirements Specifications, Standard 830-1998.

IEEE (1998b). Software Reviews, Standard 1028-1997.

IEEE (2012). System and Software Verification and Validation, Standard 1012-2012.

Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. John Wiley & Sons.

Linden, F., Schmid, K., and Rommes, E. (2007). *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer.

Marcolino, A., OliveiraJr, E., Gimenes, I., and Barbosa, E. (2014). Empirically Based Evolution of a Variability Management Approach at UML Class Level. In *Proc. Int. Conf. Computers, Software & Applications (COMPSAC)*, pages 354–363, Vasteras, Sweden.

Mello, R. M., Teixeira, E. N., Schots, M., and Werner, C. M. L. (2014). Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections. *Journal of Universal Computer Science (JUCS)*, 20(5):720–745.

OliveiraJr, E., Gimenes, I., and Maldonado, J. (2010). Systematic Management of Variability in UML-based Software Product Lines. *Journal of Universal Computer Science (JUCS)*, 16(17):2374–2393.

Rombach, D., Ciolkowski, M., Jeffery, R., Laitenberger, O., McGarry, F., and Shull, F. (2008). Impact of research on practice in the field of inspections, reviews and walkthroughs. *ACM SIGSOFT Software Engineering Notes*, 33(6):26.

SEI (2009). *Software Engineering Institute Arcade Game Maker (AGM) Pedagogical Product Line*. Carnegie Mellon University.

Travassos, G. H., Shull, F. J., Fredericks, M., and Basili, V. R. (1999). Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality. In *Proc. Int. Conf. (ACM SIGPLAN) on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 47–56, New York, USA. ACM Press.