Towards Initial Evidence of SMartyCheck for Defect Detection on Product-Line Use Case and Class Diagrams

Ricardo Theis Geraldi, Edson OliveiraJr*
State University of Maringá, Informatics Department, Maringá, Paraná, Brazil.

* Corresponding author. Tel.: +55 44 30115121; email: edson@din.uem.br
Manuscript submitted February 2, 2017; accepted May 2, 2017.
doi: 10.17706/jsw.12.5.379-392

Abstract: Software Product Line (SPL) is a promising approach for specific domain software artifacts reuse. In order to ensure SPL quality it is essential to perform activities for verification and validation. In this scenario, one applies software revision to the quality control process in order to ensure the quality of a software engineering process in each stage. Amongst software revision techniques, there is the checklist-based reading technique. This technique allows the detection of several defect types, whereas improves the quality of the software products. This paper presents the empirical evaluation of an SPL inspection technique based on checklist, named SMartyCheck. The main objective of SMartyCheck is to inspect Unified Modeling Language (UML) use case and class SPL diagrams based on the Stereotype-based Management of Variability (SMarty) approach. For evaluating SMartyCheck it was adopted the sequential exploratory strategy based on mixed-methods, aiming at analyzing the SMartyCheck feasibility throughout qualitative, then quantitative studies. The results obtained with such empirical studies allowed providing initial evidence improving SMartyCheck corroborating the relevance of the technique quality in the academic environment. Therefore, we provide an initial body of knowledge for planning prospective studies in industrial sets, as well as the automation of SMartyCheck.

Key words: Inspection, smartycheck, software product Line, UML.

1. Introduction

In order to support software reuse in a systematically non-opportunistic manner, the concept of Software Product Line (SPL) arises. SPL is a set of software systems that share common features managed for the development of customized products [6]-[23]. Several activities are essential to the adoption of SPL, such as variability management. Variability is the way as members of an SPL differentiate from each other and is described as variation points, variants and constraints between variants [7], [13].

According to studies conducted by Chen *et al*. [8] and Galster *et al*. [13], there is a large number of variability management approaches in the literature. The Stereotype-based Management of Variability (SMarty) approach allows the variability management in SPL, based on Unified Modeling Language (UML) diagrams [28], [29]. SMarty is comprised of an UML profile, the SMartyProfile, and a process, the SMartyProcess, consisting of activities and guidelines to identify and trace variability in UML diagrams. Experimental evaluations performed in the studies of Marcolino *et al*. [24]–[26] and Bera *et al*. [4] evidenced the effectiveness of SMarty.

Different defect types must be detected and removed early in a project; otherwise, they can generate high costs in developing and maintaining products [5]. Benefits provided by software inspections [1], [31], [39] and a reduced number of current inspection techniques to SPL in the literature [10], [27] motivate this work in inspecting UML diagrams containing variability.

The adoption of the SMarty approach makes it favorable inspecting UML diagrams, as it can improve the detection and mitigation of potential defects in UML-based SPLs. SMarty has several stereotypes applied to different UML diagram elements, as well as specific stereotypes [4]. Therefore, it comes up necessary to proposing a checklist-based inspection technique initially for use case and class SPL diagrams. We desired to keep the technique checklist size feasible to encourage its comprehension and readability. Anda and Sjøberg [2] present defect types for use cases and Travassos *et al*. [35] to OO design, but neither of them refers to SPL. In addition, Mello *et al*. [27] and Cunha *et al*. [10] adapted defects to problem space feature models. We, thus, concentrated our efforts on the solution space, although we strongly believe that our technique can be combined to such techniques focused on feature models.

This paper presents the SMartyCheck as checklist-based technique to detect defects in SPL use case and class diagrams with variability modeled to improve the quality of SMarty diagrams in the SPL Domain Engineering phase. With the initial adoption of defect types based on a systematic mapping study (Section 2), we decided to develop, initially, our SPL inspection technique based on checklist. We empirically evaluated SMartyCheck in two studies (qualitative and quantitative) based on the sequential exploratory strategy using mixed-methods. We previously evaluated the theoretical proposal of our technique by means of a qualitative study taking into consideration experts' opinion [14]. Initial evidence pointed out to the feasibility of SMartyCheck according to such experts.

This paper provides the following structure: Section 2 presents background and related; Section 3 describes the SMartyCheck technique as well as its characterization and taxonomy of defect types; Section 4 presents the empirical feasibility studies of SMartyCheck; and Section 5 presents conclusion, contributions, limitations and directions for future work.

2. Background and Related Work

This section presents the main concepts for this work.

2.1. Software Product Line and Variability

Software Product Line (SPL) is an approach to represent a set of software systems defined as a family of products. Members of this family are specific products developed from a central infrastructure, named core assets. It consists of a set of common features and variable artifacts [23]. In this scenario, the framework proposed by Pohl *et al*. [30] aims to incorporate the central concepts of SPL engineering. This framework facilitates the reuse of artifacts to guarantee mass customization through variability by means of two processes: Domain Engineering and Application Engineering.

The successful adoption of SPL is due to the technical and organizational activities, such as variability management [6], [7], [13]. Variability allows the development of customized products based on the configuration of reusable artifacts to a particular context [8], [30]. Variability resolution achieved throughout the selection of one or more variants related to a specific variation point. A variation point can occur at different abstraction levels of SPL artifacts. It allows the resolution of variability in one or more places by means of its associated variants. These variants represent possible elements chosen to solve a variation point. Constraints establish relationships between one or more variants [23], such as mutex and requires.

2.2. The SMarty Approach for Variability Management

The SMarty main objective is to allow variability management of SPL in UML diagrams. We initially evaluated SMarty by means of a set of experiments [4], [24]–[26]. SMarty is composed of an UML Profile, the SMartyProfile, and a Process, the SMartyProcess [28], [29]. SMartyProfile is a mechanism, which allows graphical representation of SPL variability in UML diagrams as a set of stereotypes and meta-attributes for variation points and variants [28]. SMartyProcess is a systematic process that consists of a set of guidelines that aim at guiding the SPL engineer in the identification, representation and variability tracing.

For the illustration purpose, Fig. 1 presents the Arcade Game Maker (AGM) SPL[1] use case diagram according to SMarty. AGM is a pedagogical SPL developed for games. The Software Engineering Institute (SEI) proposed AGM to support learning and testing of SPL concepts. It has a full set of documents and UML diagrams, as well as a set of tested classes and source code. AGM has been widely used to illustrate the concepts of several different SPL approaches (as well as SMarty), case studies and SPL architectures evaluation.

Observing Fig. 1, the use case **Play Selected Game** is a mandatory variation point. Every variation point has one or more variants associated. Such use case has the following variants: **Play Brickles**, **Play Pong**, and **Play Bowling**. These variants are inclusive (**<<alternative_OR>>**) and may be chosen to derive specific products.

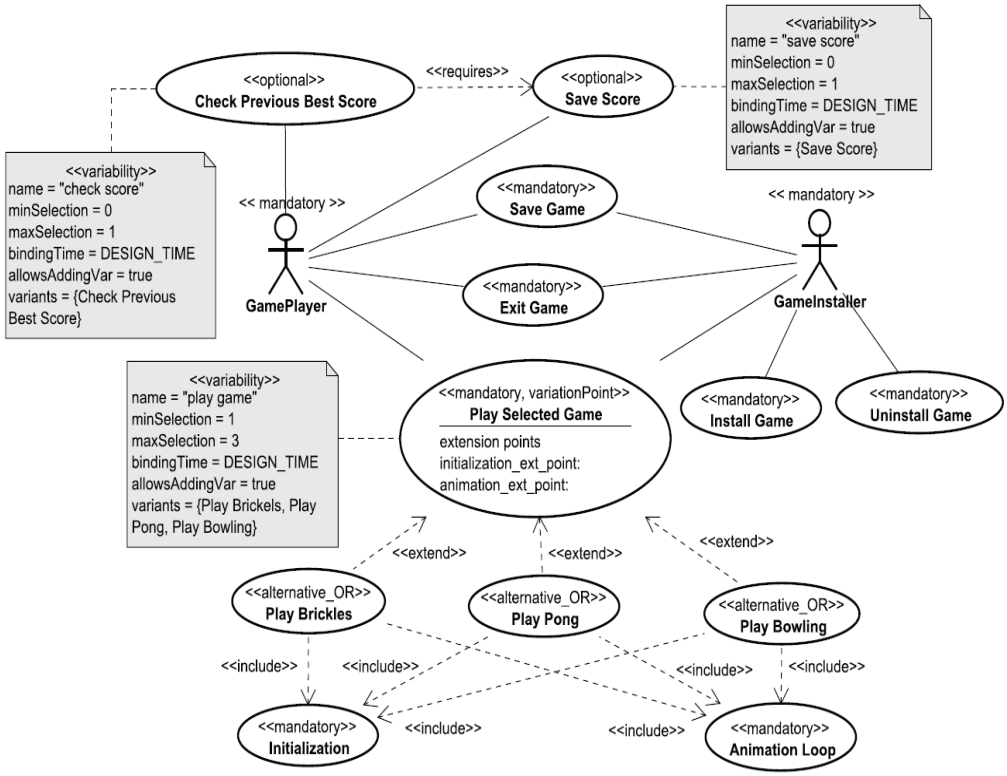


Fig. 1. AGM SPL Use Case Diagram according to SMarty [26].

2.3. Software Inspection

The principle of software inspection is to detect artifact defects in a systematically and planned manner from a software, which does not meet its specifications [12], [20]. Inspections can detect from 60% to 90% of major defects with an average of 80% [5]. There are recent proposals of defect types taxonomies, reading techniques and classifications of defect types which obtained effective results to optimize defects detection in different software artifacts [1], [22]. Therefore, a set of defect categories should be combined with standards, such as the IEEE 1012-2012 - System and Software Verification and Validation [18].

After establishing defect types taxonomy for inspection, reading comprehension techniques is required. The main ones are Checklist-Based Reading (CBR) [1], [2], [27], Perspective-Based Reading (PBR) [16], [20], [32], [34] and Ad hoc technique [31]. CBR uses a document in a checklist format with clear items (yes/no) stated in the form of questions. As opposed to CBR, the Ad hoc technique does not have systematic criteria to conduct inspections, depending on the inspector's knowledge and experience. Specific techniques for SPL, as well as PBR, were not identified in literature (Section 2.4) to be compared to SMartyCheck due to difference of the target artifacts and defect types inspected. Thus, we excluded the possibility of comparing SMartyCheck with a PBR technique for specific artifacts that are not SPL UML diagrams.

[1] Software Engineering Institute (SEI) - Arcade Game Maker (AGM) Product Line: *http://www.sei.cmu.edu/productlines/ppl*

2.4. Related Work

We carried out a systematic mapping study[2] until January 2017 on defect types and software inspection techniques taking into account digital databases as IEEE Xplore, ACM Digital Library, ELSEVIER ScienceDirect, Scopus, Compendex and Google Scholar. The following steps were defined: **Filter #1**, identify primary studies based on defined search strings = #2096; **Filter #2**, analysis, classification and aggregation of the retrieved studies based on inclusion and exclusion criteria = #86; and **Filter #3**, fully reading of primary studies = #32. We identified important different defect types in experiments and taxonomies, proposed techniques and approaches from the current literature.

It is important to highlight this paper did not investigate related work on defect types and code-reading techniques, such as Object-Oriented Code Reading and Object-Oriented Framework Reading, discussed by Zhu [40]. Although SMartyCheck is currently being automated, such a technique can be adapted in the future (Section 5) to detect code level defects.

Two main works are related to ours: Software Product Line Inspection Techniques (SPLIT) [10] and FMCheck [27] techniques. The main difference between such works and SMartyCheck relies in the inspected artifacts and defect types. Unlike the techniques mentioned, focused on feature models, SMartyCheck (Section 3) performs inspections on UML use case and class diagrams based on SMarty.

3. The SMartyCheck Technique

SMartyCheck is in accordance to the inputs and outputs of each sub-process of the Domain Engineering stage (Pohl's *et al*. [30] engineering framework) by containing variability in the produced core artifacts. The Checklist-Based Reading (CBR) technique is the basis of the SMartyCheck conception. SMartyCheck consider CBR according to the related works and the carried out systematic mapping study (Section 2).

The SMartyCheck checklist was developed based on the analysis and adaptation of defect types identified from the systematic mapping. From the analysis and classification based on **Filter #2**, several different studies with distinct defect types taxonomy proposals, most of them empirically evaluated, could be adapted by existing software inspection techniques, such as SMartyCheck. Thus, we made adaptations of existing studies to the SPL context toward the SMartyCheck checklist.

Therefore, we proposed the checklist of SMartyCheck specifically to inspect use case and class UML diagrams of SPLs. Several studies from the mapping study adapted their checklists based on the definitions of defect types from IEEE Standard 830-1998 [17], Standard 1012-2012 [18] and requirements engineering.

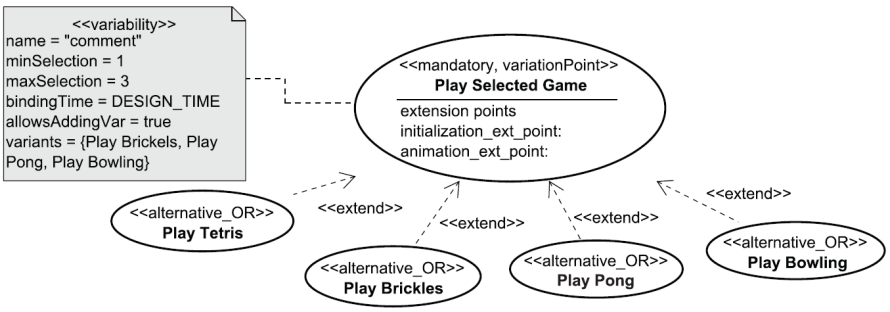
In order to perform inspections, inspectors should read the checklist answering each question from a list of assertive questions, based on their prior knowledge with relation to the SMartyCheck checklist defect types [14]. For the illustration purpose, Fig. 2 presents a fragment of the AGM SPL with examples of defect types by means of use case or class elements with artificially incorporated defects based on diagrams applied to the AGM SPL.

Sample defect types adopted from IEEE Standard 1012-2012 [18] and Travassos *et al*. [35]:

- **Inconsistency**: refers to the lack of internal consistency, in which a subset of individual elements has conflicts with use case or class diagrams. E.g., an use case or class element does not meet the constraints of SMarty diagrams modeling. The use case **Play Selected Game** has four inclusive (**<<alternative_OR>>**) variants (**Play Tetris**, **Play Brickles**, **Play Pong** and **Play Bowling**), whereas should have three as the meta-attribute **maxSelection = 3**, specifies in Fig. 2(a);
- **Incorrect Fact**: the names of one or more use cases or class elements are contradictory. E.g., the name of one or more use cases or class elements in diagram is incorrect. According to Fig. 2(b), the use case **Play Pong** has the name **Pl Pon**;
- **Extraneous Information**: the use case or class elements are redundant. E.g., this defect type detects over specified or duplicated elements in use cases or classes. Fig. 2(c) presents the class **Velocity**

[2] Our systematic mapping study paper is accepted for publication in the International Journal of Software Engineering and Knowledge Engineering (IJSEKE). Preprint is not available yet.

duplicated.



(a) Inconsistency.

(b) Incorrect Fact. (c) Extraneous Information.

Fig. 2. Smartycheck defect types examples.

Each SMartyCheck checklist (use cases or class) have a taxonomy with 11 defect types and 17 items and can be used in two categories of inspection: (i) to inspect possible versions of an SPL which may be modeled and compared according to an original SPL (oracle); and (ii) to inspect artifacts of an SPL containing possible defects.

The SMarty stereotypes were applied from existing SMarty guidelines for adapting the defect types applied in checklists of SMartyCheck. The stereotypes help improving indeed the quality of UML diagrams inspections when properly specified [33].

The checklist of SMartyCheck was improved iteratively after analyzing and interpreting the results of these two studies (Section 4). The following checklist is the final version, named SMartyCheck 2.2, after such studies.

***Comparison Category*:** *Compares use case or class elements with defects to use case or class elements with no defects (oracle)*.

1) **Business Rule (BR)**

   BR.1 The use case/class is not clear with the purpose and the desired functionalities based on defined domain.

2) **Inconsistency (Incons)**

   Incons.1 Is there any use case/class specified with the stereotype **<<variationPoint>>** from which the number of specified variants is major or minor than defined in **maxSelection** or **minSelection** the variability (**<<variability>>**) associated?

   Incons.2 Is there any use case/class specified with the stereotype **<<optional>>** from which the number of specified variants is major or minor than defined in **maxSelection** or **minSelection** the variability (**<<variability>>**) associated?

3) **Incorrect Fact (IF)**

   IF.1 Is there any use case/class with incorrectly name in SPL?

   IF.2 Is there any use case/class that it is not be compare name in SPL?

   IF.3 Is there any use case/class with stereotype **<<variationPoint>>** associated with use case/class in the SPL which not **<<alternative_OR>>**?

4) **Non-modifiable (Nm)**

   Nm.1 Is there any use case/class specified with the stereotype **<<variationPoint>>**, in which variants associated (**<<optional>>**, **<<alternative_OR>>** or **<<alternative_XOR>>**) can not be combined or selected in accordance with variants already specified in the meta-attribute variants in SPL?

5) **Omission (Om)**

   Om.1 Is there any use case/class specified as mandatory (required) through the stereotype

<<mandatory>> that is not specified on the SPL?

6) **Extraneous Information (EI)**

EI.1 Is there any use case/class specified besides use cases/class existing in the SPL?

EI.2 Is there any use case/class with its functionality duplicated in SPL?

7) **Intentional Deviation (ID)**

ID.1 Is there any use case/class which requires the selection of another (**<<requires>>**) and this another is not specified in the SPL?

ID.2 Is there any use case/class not which requires selection of another (**<<mutex>>**) and this another is not specified in the SPL?

Non-comparison Category: *Inspect only use case or class diagram with defects*.

8) **Ambiguous (Am)**

Am.1 Is there any use case/class in that its name is equal to the another use case/class in a manner to owning a duplicate interpretation?

9) **Anomaly (An)**

An.1 Is there any use case/class specified as **<<alternative_OR>>** which extend **<<extend>>** another use case/class that is not specified **<<variationPoint>>**?

An.2 Is there any use case/class specified as **<<alternative_XOR>>** which extend **<<extend>>** another use case/class that is not specified **<<variationPoint>>**?

10) **Unstable (Uns)**

Uns.1 Is there any use case/class specified with the stereotype **<<variationPoint>>**, in which has the stereotype associated **<<variability>>** from which the meta-attribute name equals to other use cases/class elsewhere specified with the stereotype **<<variationPoint>>**?

11) **Infeasible (Inf)**

Inf.1 Is there any use case/class specified with stereotype **<<variationPoint>>** which not allows you to add new variants as defined in the meta-attribute **allowsAddingVar = false**?

12) (*see in 6*). **Extraneous Information (EI)**

EI.2 Is there any use case/class with its functionality duplicated in SPL?

4. SMartyCheck Empirical Evaluation

We previously carried out a qualitative empirical study[3] aiming at analyzing the SMartyCheck feasibility. According to feedback of the participants of this study, checklist items were considered inconsistent, ambiguous and difficult to understand. Thus, this qualitative study contributed for the improvement of the descriptions and checklist items supported by the stereotypes of SMarty.

We adopted the Sequential Exploratory Strategy according to mixed-methods [9] (Fig. 3).

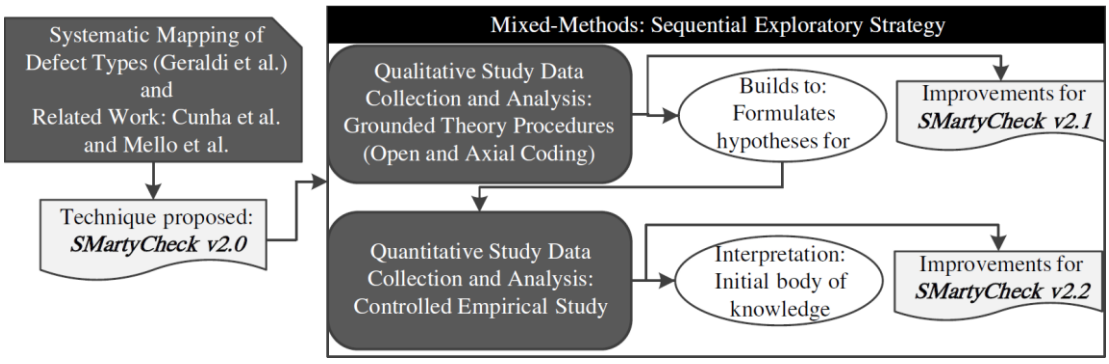


Fig. 3. The smartycheck sequential exploratory strategy.

[3] Geraldi et al. [14] presented a detailed version of this study at the International Conference on Enterprise Information Systems (ICEIS 2015).

This research method allowed us to plan and conduct a quantitative study to evaluate the effectiveness of SMartyCheck with regard to other types of inspection technique, as suggested by the experts from the qualitative study, in this case to Ad hoc inspections (see instrumentation and results[4]). Thus, this section presents planning, execution and analysis of data from the quantitative study following the guidelines by Jedlitschka *et al*. [19] and Wohlin *et al*. [36].

4.1. Research Objectives

Analyze SMartyCheck, **for the purpose of** characterize, **with respect to** its efficiency, efficacy, and effectiveness compared to the Ad hoc technique, **from the point of view** of SPL artifact inspectors, **in the context of** undergraduate and graduate students and industrial practitioners of the Software Engineering area. This study takes advantage of the benefits of taking into consideration students to perform experiments as pointed out by Carver *et al*. [38].

4.2. Experimental Design

To collect evidence, the following **Research Question (R.Q.1)** was defined according to the research objectives: *Has SMartyCheck efficiency (**i**), efficacy (**a**) and effectiveness (**e**) on inspecting SMarty use case and class diagrams compared to the Ad hoc technique?*

Specific techniques were not identified in literature to be compared to SMartyCheck due to difference of the artifacts and defect types inspected. We, then, choose an Ad hoc technique. The following hypotheses were defined/tested based on R.Q.1:

- **Efficiency Null Hypothesis** *(H_i0): μ(i(SMartyCheck)) = μ(i(Adhoc));*
- **Efficiency Alternative Hypothesis** *(H_i1): μ(i(SMartyCheck)) <> μ(i(Adhoc));*
- **Efficacy Null Hypothesis** *(H_a0): μ(a(SMartyCheck)) = μ(a(Adhoc));*
- **Efficacy Alternative Hypothesis** *(H_a1): μ(a(SMartyCheck)) <> μ(a(Adhoc));*
- **Effectiveness Null Hypothesis** *(H_e0): μ(e(SMartyCheck)) = μ(e(Adhoc));*
- **Effectiveness Alternative Hypothesis** *(H_e1): μ(e(SMartyCheck)) <> μ(e(Adhoc)).*

We adopted several metrics from the studies of Gopalakrishnan Nair *et al*. [15] and Karg *et al*. [21]. These metrics allowed us to calculate the dependent variables: efficiency ***(i)*** and efficacy ***(a)*** of inspections carried out by the participants of each technique ***(t)***. In summary, efficiency ***(i)*** calculates the number of detected defects with relation to productivity/time of inspections (1). Efficacy ***(a)*** calculates the quality of inspections measured by a percentage (2). The dependent variable effectiveness ***(e)*** was calculated according to the metric based on the studies of Basili and Selby [3] and Marcolino *et al*. [24], which is the number of defects detected correctly ***(nDefC)*** minus incorrectly ***(nDefI)*** (3). The effectiveness ***(e)*** calculates the number of hits ***(nDefC)*** and errors ***(nDefI)*** of each technique. Therefore, the following metrics adopted:

$$i(t) = \frac{Total\ number\ of\ detected\ defects}{Total\ inspection\ time} \tag{1}$$

$$a(t)(\%) = \frac{Total\ number\ of\ detected\ defects}{Total\ existing\ defects} * 100 \tag{2}$$

$$e(t) = \begin{cases} nDefC, & if\ nDefI = 0 \\ nDefC - nDefI, & if\ nDefI > 0 \end{cases} \tag{3}$$

Two independent variables were defined: the inspection technique, which is a factor with two treatments

[4] The learning experience in applying these studies shared with researchers. Instrumentation and results for the qualitative study is available at: *http://bit.ly/28SZ6OI*; and for the quantitative study is available at: *http://bit.ly/28SxzgC*.

SMartyCheck and Ad hoc, and the AGM SPL, which is pre-fixed value variable. Dependent variables efficiency, efficacy and effectiveness are calculated for each technique based on SMarty use case and class diagrams.

Most participants are from the software engineering area: three undergraduates (21.5%), one graduate (7.5%) and 10 master students (71%). These participants are academics and practitioners who work directly with the area or who have knowledge of software verification and validation over the last few years in different universities and companies in Brazil. Selection of participants did not occur randomly due to universe of candidates be very restricted. One group of N = 7 participants performed inspections on the SMartyCheck checklist. Random capacity was carried out with regard to study objects distribution (SMarty use cases and class diagrams) and SMartyCheck and Ad hoc techniques distribution to participants using AGM SPL for inspections to response the R.Q.1.

All participants received a set of documents: a copy of the Consent Term, a Characterization Questionnaire (to indicate knowledge and experience), a document on Software Product Line (SPL) and a document with the description of the AGM SPL. Electronic forms were created, distributed and sent via email randomly to the participants. Specific forms were created for each technique, containing two SMarty use cases and two SMarty class diagrams. Each participant received two forms (one form with two use cases and one form with two class diagrams) of a single technique at random. All forms were created using LimeSurvey[5].

A pilot project was performed aiming at evaluating the instrumentation used in order to adapt it. A PhD lecturer with knowledge on software engineering evaluated this instrumentation. The data obtained in this pilot project was not taken into consideration as results in data analysis of this study.

4.3. Execution

The 14 participants were divided into two groups and trained before performing the study. It was presented to participants examples of inspection of SMarty use cases and class diagrams through the explanation of defect types contained in the SMartyCheck using the SPL Mobile Media [37]. For the explanation of the Ad hoc technique participants were also trained on defect types through inspection using the SPL Mobile Media. After training, the electronic forms were sent via email to the participants to be answered over a period of up to seven days.

4.4. Analysis and Results Discussion

The appropriate statistical tests were applied to analyze the collected data performed. Thus, it was possible to calculate the efficiency, efficacy and effectiveness of the inspections performed by each participant using: total number of defects, total inspection time, total existing defects, defect numbers detected correctly and incorrectly. Obtained results are shown in Table 1 and plotted in box-plots according to Fig. 4 and Fig. 5.

Table 1. SMartyCheck and ad Hoc Observed Values

SMartyCheck (SPL AGM) - Use Case Diagram				Ad hoc (SPL AGM) - Use Case Diagram			
#Id Subject: Group 1	Efficiency	Efficacy	Effectiveness	#Id Subject: Group 2	Efficiency	Efficacy	Effectiveness
1	0.34	100%	4	8	0.26	56%	1
2	0.28	100%	8	9	0.13	67%	2
3	0.72	100%	8	10	0.23	44%	-1
4	0.09	100%	1	11	0.17	22%	-5
5	0.47	100%	9	12	0.14	83%	4
6	0.18	83%	1	13	0.18	33%	-3
7	0.19	78%	5	14	0.05	67%	3
Mean	0.32	0.94	5.14	Mean	0.17	0.53	0.14
Standard Deviation	0.20	0.09	3.09	Standard Deviation	0.06	0.20	3.04
Median	0.28	1.00	5.00	Median	0.17	0.56	1.00

[5] LimeSurvey: *http://www.limesurvey.org*.

SMartyCheck (SPL AGM) - Class Diagram			
#Id Subject: Group 1	Efficiency	Efficacy	Effectiveness
1	0.52	100%	5
2	0.37	100%	7
3	0.48	100%	7
4	0.09	83%	1
5	0.46	100%	7
6	0.13	50%	-3
7	0.13	86%	2
Mean	0.31	0.88	3.71
Standard Deviation	0.17	0.17	3.57
Median	0.37	1.00	5.00

Ad hoc (SPL AGM) - Class Diagram			
#Id Subject: Group 2	Efficiency	Efficacy	Effectiveness
8	0.38	57%	1
9	0.05	50%	0
10	0.17	43%	-1
11	0.42	44%	-1
12	0.66	67%	2
13	0.30	17%	-5
14	0.08	89%	7
Mean	0.29	0.52	0.43
Standard Deviation	0.20	0.21	3.37
Median	0.30	0.50	0.00

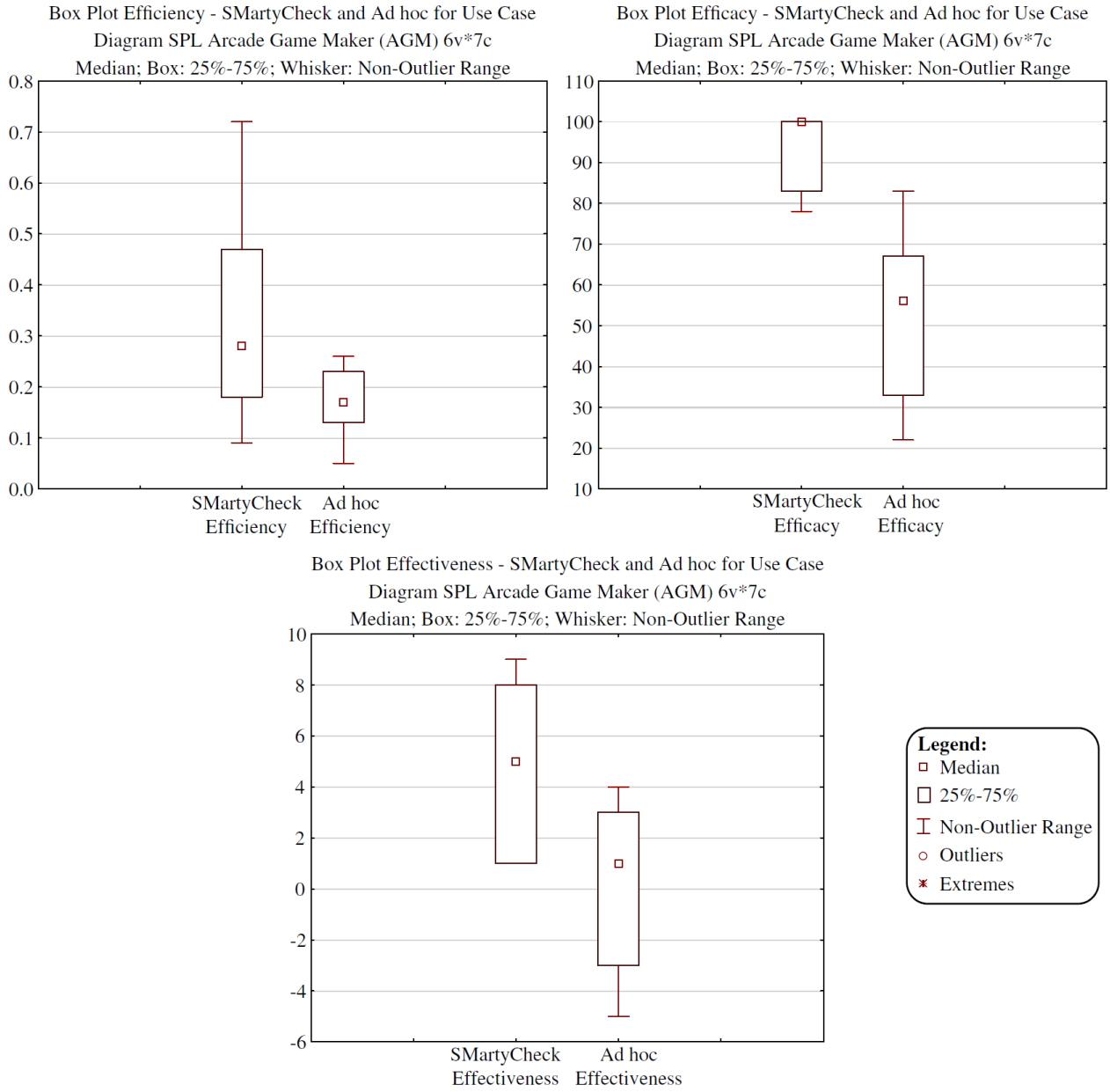


Fig. 4. Box plots of efficiency, efficacy and effectiveness representing smartycheck and Ad hoc inspections for use case diagram on AGM SPL.

Analyzing the results obtained in inspections, the following steps allowed to answer the R.Q.1 and test the hypotheses: (i) analysis and interpretation of data collected for the SMartyCheck and Ad hoc techniques (Table 1, Fig. 4 and Fig. 5), through Shapiro-Wilk normality test, T test and Mann-Whitney Wilcoxon test [36] to evaluate and compare techniques (see results in http://bit.ly/28SxzgC); and (ii) analysis and interpretation of the correlation between the efficiency, efficacy and effectiveness of techniques (SMartyCheck e Ad hoc) and the answers provided by the participants by assigning weights using Likert Scale in the Characterization Questionnaire, by applying the Spearman correlation test [36]. Therefore, we observed that the participants generally have a moderate knowledge on UML, basic knowledge on SPL and software inspection.

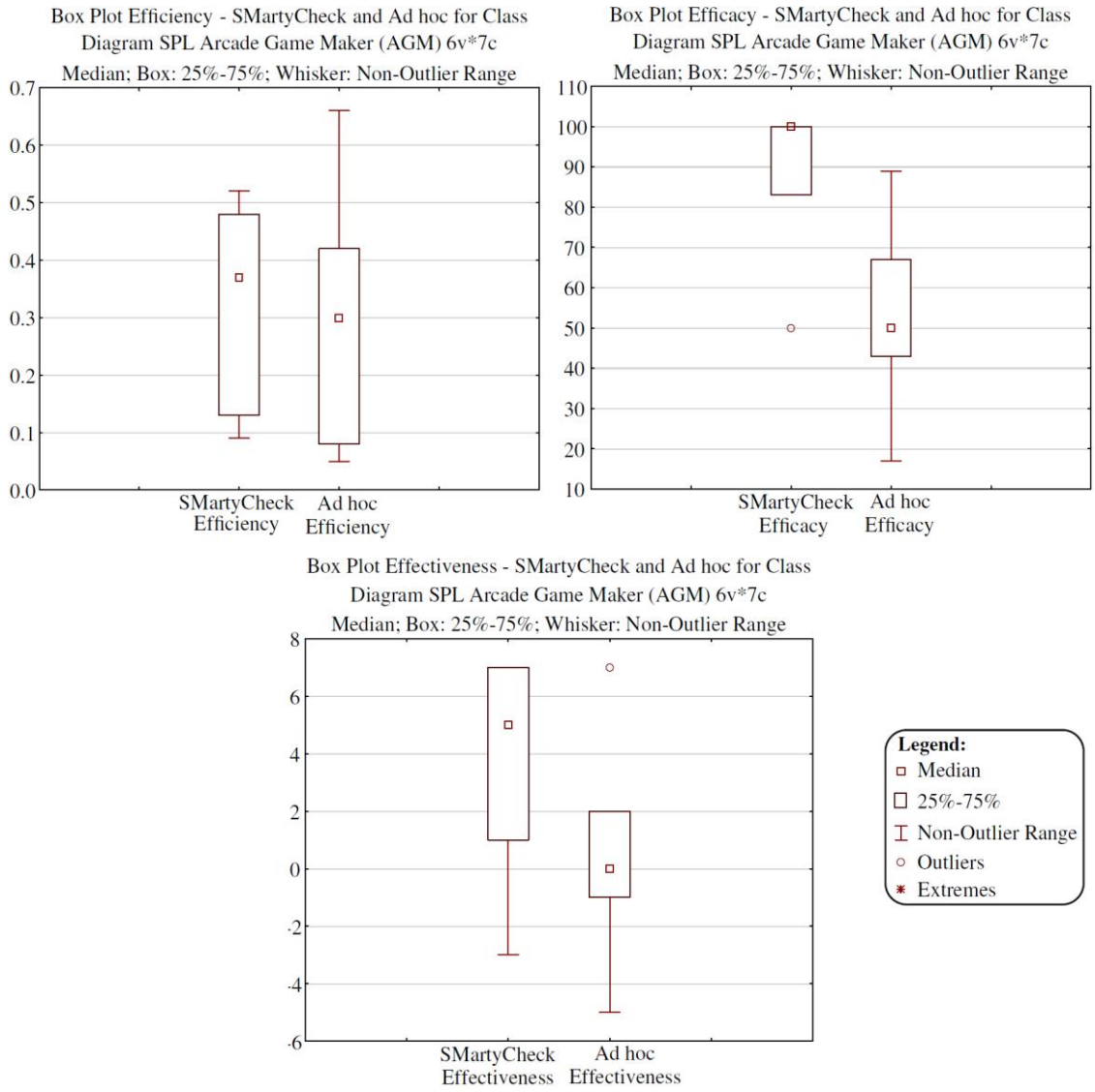


Fig. 5. Box plots of efficiency, efficacy and effectiveness representing smartycheck and Ad hoc inspections for class diagram on AGM SPL.

Analyzing the results obtained on interpretations performed over Table 1 and Fig. 4 and Fig. 5 we can infer the following: (i) according to the evidence provided for this study, the SMartyCheck technique is more efficient, efficacy and effective compared to the Ad hoc inspection; (ii) the statistical tests were applied according to the distribution of samples, in order to ensure the acceptance or rejection of the hypothesis set for this study correctly; (iii) it is believed that the statistical difference in the efficacy obtained for SMartyCheck was due to the Ad hoc does not have systematic criteria for inspections. The evidence obtained of efficacy for the SMartyCheck can be related to effectiveness of SMarty approach, which has a UML profile, SMartyProfile, and a process with

guidelines to the user; and (iv) the Ad hoc technique can be evidenced as a technique that can be efficient and effective, if inspectors have a high degree of experience and knowledge in various areas (UML, SPL and software inspection). However, this finding needs extra empirical studies.

After applying statistical tests (Shapiro-Wilk normality test, T test and Mann-Whitney Wilcoxon test [36]) we rejected all null hypotheses of this study with regard to: efficiency, efficacy and effectiveness. In all cases SMartyCheck is differently in the inspections than Ad hoc (Fig. 4 and Fig. 5).

Main SMartyCheck improvements were carried out as follows: (i) the organization of the SMartyCheck items by categories; (ii) the definition of a format more suitable to the taxonomy of the technique, as well as the improvement of the checklist description in both items addition (IF.3 and An.2), item change (BR.1, Incons.1, Incons.2, IF.1, IF.2, Nm.1, Om.1, EI.1, EI.2, ID.1, ID.2, An.1, Uns.1, Inf.1) and exclusion of defect types (Incomplete and Incorrect, besides item Am.2). After the qualitative study, we distributed 18 items in the checklist for 13 defect types. With the results of this quantitative study, we distributed 17 items in the final checklist (in different categories) for 11 defect types.

4.5. Validity Evaluation

The main threats to validity are as follows.

4.5.1. Threats to internal validity

- ***Differences among participants***, as the sample of this study was small, a few variations with regard to the skills of the participants were evident. However, the tasks performed by the participants in equal numbers, participants were divided into two groups, where each group received electronic questionnaires specific for a technique;
- ***Training session***, two trainings sessions (one for each technique: SMartyCheck and Ad hoc) were performed to two different groups of participants according to their knowledge. Thus, we considered valid the inspections and the detection of defects on SMarty diagrams;
- ***Fatigue effects***, the training sessions to each group lasted 45 minutes in average. After this, we sent the electronic forms to the participants to be answered over a period of up to seven days. Thus, the fatigue effects were reduced, decreasing the delivery pressure for the answers;
- ***Influence among participants***, we mitigated the influence between the participants as the electronic questionnaires to be answered by the same participant were randomly distributed.

4.5.2. Threats to external validity

- ***Instrumentation***, different pedagogical SPLs used in training (Mobile Media and AGM) might threaten this study. As these SPLs are not commercial, some assumptions can be made with regard to this threat. Additional empirical studies can be conducted in order to obtain different results by using commercial SPLs;
- ***Participants***, obtaining qualified participants were one of the greatest challenges of this study. Thus, participants from academia and practitioners with relevant knowledge in software engineering were invited.

4.5.3. Threats to conclusion validity

The main threat for this study is related to the number of participants (N=14). This number is relatively small, but the knowledge of the participants who contributed to this study was significant. Thus, a larger number of participants would allow a better generalization of the results.

4.5.4. Threats to construct validity

In this study, we defined and tested three dependent variables based on the conducted pilot project. Therefore, the instrumentation was appropriate before being applied to the actual study. It was possible to verify that the metrics applied under these three variables are really in accordance with the quantitative evaluation proposed. We also evaluated the independent variables in the pilot project.

5. Conclusion and Future Work

The main result of this work is the empirical evaluation of SMartyCheck based on the sequential exploratory strategy. We had, thus, initial evidence that SMartyCheck is feasible in terms of efficiency, efficacy and effectiveness for SMarty use case and class diagrams inspections based on the context of this empirical evaluation strategy.

Thus, we achieved the following results: (i) SMartyCheck has coherent defect types; (ii) SMartyCheck allows the correction of diagrams with a well-defined structure supported by SMarty stereotypes; (iii) SMartyCheck has a checklist that systematically follows criteria to conducting inspections, unlike the Ad hoc technique; and (iv) initial evidence from this quantitative study corroborated the previous qualitative study with regard to the relevance of the SMartyCheck quality.

SMartyCheck contributes to improving the SPL software inspection process taking into consideration SMarty diagrams, minimizing defects in artifacts, thus increasing the quality of generated SPL products. In addition, the proposal and empirical evaluation of a new inspection technique for SPL contributes to provide academia and industry an initial body of knowledge.

We identified limitations of SMartyCheck and they should be minimized with new research and the conduction of new empirical studies. Such limitations are: (i) automatic generation mechanism and product configuration are not supported by SMartyCheck yet; (ii) lack of UML-based SPL inspection techniques in the literature to be compared to SMartyCheck; (iii) SMartyCheck supports the inspection of SMarty use case and class diagrams, but not other UML diagrams as component, sequence and activity; (iv) SMartyCheck needs to be evaluated in an industrial set using commercial SPLs.

According to the experience obtained from the development of SMartyCheck and the performing of the empirical studies, we provide directions for future work, as follows: (i) mitigation of more defect types to adapt them to the SMartyCheck taxonomy; (ii) inspection of different SMarty diagrams (activity, sequence and component); (iii) automation of the SMartyCheck inspection process throughout the development of a syntactic analysis tool (which is currently being developed); (iv) extending SMartyCheck for inspecting feature models on the problem space or base models used by the CVL language engine; (v) plan and conduct more empirical studies to improve the initial body of knowledge towards assuring the SMartyCheck quality; and (vi) propose a technique for inspecting SMarty diagrams based on the PBR inspection technique and compare it to SMartyCheck.

Acknowledgments

The authors would like to thank CAPES, a Brazilian funding agency, for supporting this project. They also thank all the participants of both quantitative and qualitative studies and the Informatics Department of the State University of Maringá (UEM) for providing all infrastructure for performing such studies. In addition, the authors would like to thank a lot the Centro de Tecnologia (CTC-UEM) for financially supported the publication of this paper.

References

[1] Alshazly, A., Elfatatry, A., & Abougabal, M. (2014). Detecting defects in software requirements specification. *Alexandria Engineering Journal, 53(3)*, 513–527.

[2] Anda, B., & Sjøberg, D. (2002). Towards an inspection technique for use case models. *Proceedings of the Int. Conf. on Software Engineering and Knowledge Engineering* (pp. 127–134). ACM Press.

[3] Basili, V., & Selby, R. (1987). Comparing the effectiveness of software testing strategies. *IEEE Transactions on Software Engineering, SE-13(12)*, 1278–1296.

[4] Bera, M., OliveiraJr, E., & Colanzi, T. (2015). Evidence-based smarty support for variability identification and representation in component models. *Proceedings of the Int. Conf. on Enterprise Information Systems* (pp. 295–302).

[5] Boehm, B., & Basili, V. (2001). Software defect reduction top 10 list. *IEEE Computer, 34(1)*, 135–137.

[6] Bosch, J., Capilla, R., & Hilliard, R. (2015). Trends in systems and software variability. *IEEE Software, 32(3)*, 44–51.

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

[8] Chen, L., Ali Babar, M., & Ali, N. (2009). Variability management in software product lines: A systematic review. *Proceedings of the Int. Conf. on Software Product Line* (pp. 81–90). Carnegie Mellon University.

[9] Creswell, J., & Clark, V. (2010). *Designing and Conducting Mixed Methods Research* (2nd ed.). SAGE.

[10] Cunha, R., Conte, T., Almeida, E., & Maldonado, J. (2012). A set of inspection techniques on software product line models. *Proceedings of the Int. Conf. on Software Engineering and Knowledge Engineering* (pp. 657–662).

[11] Fagan, M. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal, 15(3)*, 182–211.

[12] Fagan, M. (1986). Advances in software inspections. *IEEE Transactions on Software Engineering, SE-12(7)*, 744–751.

[13] Galster, M., Weyns, D., Tofan, D., Michalik, B., Avgeriou, P. (2013). Variability in software systems - A Systematic literature review. *IEEE Transactions on Software Engineering, 40(3)*, 282–306.

[14] Geraldi, R. T., OliveiraJr, E., Conte, T., & Steinmacher, I. (2015). Checklist-based inspection of smarty variability models - Proposal and empirical feasibility study. *Proceedings of the Int. Conf. on Enterprise Information Systems* (pp. 268–276).

[15] Gopalakrishnan Nair, T., Suma, V., & Kumar, T. P. (2012). Significance of depth of inspection and inspection performance metrics for consistent defect management in software industry. *IET Software, 6(6)*, 524–535.

[16] He, L., & Carver, J. (2006). PBR vs. checklist: A replication in the N-fold inspection context. *Proceedings of the Symp. on Empirical Software Engineering* (pp. 95–104). ACM Press.

[17] IEEE. (1998). Recommended practice for software requirements specifications - standard 830–1998.

[18] IEEE. (2012). System and software verification and validation - standard 1012–2012.

[19] Jedlitschka, A., Ciolkowski, M., & Pfahl, D. (2008). Reporting experiments in software engineering. *Guide to Advanced Empirical Software Engineering*.

[20] Kalinowski, M., & Travassos, G. (2004). A computational framework for Supporting software inspections. *Proceedings of the Int. Conf. on Automated Software Engineering* (pp. 46–55).

[21] Karg, L., Grottke, M., & Beckhaus, A. (2011). A systematic literature review of software quality cost research. *Journal of Systems and Software, 84(3)*, 415–427.

[22] Van, L. A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley.

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

[24] Marcolino, A., OliveiraJr, E., & Gimenes, I. (2014). Variability identification and representation in software product line UML sequence diagrams: Proposal and empirical study. *Proceedings of the Brazilian Symp. on Software Engineering* (pp. 141–150).

[25] Marcolino, A., OliveiraJr, E., Gimenes, I., & Barbosa, E. (2014). Empirically based evolution of a variability management approach at UML Class level. *Proceedings of the Int. Conf. Computers, Software & Applications* (pp. 354–363).

[26] Marcolino, A., OliveiraJr, E., Gimenes, I., & Maldonado, J. (2013). Towards the effectiveness of a variability management approach at use case level. *Proceedings of the Int. Conf. on Software Engineering and Knowledge Engineering* (pp. 214–219).

[27] Mello, R., Teixeira, E., Schots, M., & Werner, C. (2014). Verification of software product line artefacts: A checklist to support feature model inspections. *Journal of Universal Computer Science, 20(5)*, 720–745.

[28] OliveiraJr, E., Gimenes, I., & Maldonado, J. (2010). Systematic management of variability in UML-based

software product lines. *Journal of Universal Computer Science, 16(17)*, 2374–2393.
[29] OliveiraJr, E., Gimenes, I., Maldonado, J., & Masiero, P. (2013). Systematic evaluation of software product line architectures. *Journal of Universal Computer Science, 19(1)*, 25–52.
[30] Pohl, K., Böckle, G., & Linden, F. (2005). *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer.
[31] Rombach, D., Ciolkowski, M., Jeffery, R., Laitenberger, O., McGarry, F., & Shull, F. (2008). Impact of research on practice in the field of inspections, reviews and walkthroughs: Learning from successful industrial uses. *ACM SIGSOFT Software Engineering Notes, 33(6)*, 26–35.
[32] Shull, F., Carver, J., Travassos, G., Maldonado, J., Conradi, R., & Basili, V. (2003). Replicated studies: Building a body of knowledge about software reading techniques. *Lecture Notes on Empirical Software Engineering*, 39–84. World Scientific Publishing.
[33] Staron, M., Kuzniarz, L., &Thurn, C. (2005). An empirical assessment of using stereotypes to improve reading techniques in software inspections. *Proceedings of the Wksp. on Software Quality* (pp. 1–7). ACM.
[34] Travassos, G. (2014). Software defects: Stay away from them. do inspections! *Proceedings of the Int. Conf. on Quality of Information and Communications Technology* (pp. 1–7).
[35] Travassos, G., Shull, F., Fredericks, M., & Basili, V. (1999). Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality. *Proceedings of the ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications* (pp. 47–56). ACM Press.
[36] Wohlin, C., Runeson, P., Host, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer.
[37] Young, T. (2005). *Using AspectJ to Build a Software Product Line for Mobile Devices*, Master's thesis, University of British Columbia.
[38] Carver, J., Jaccheri, L., Morasca, S., &Shull, F. (2003). Issues in using students in empirical studies in software engineering education. *Proceedings of the Software Metrics Symp.* (pp. 239-249). IEEE.
[39] Vitharana, P. (2017). Defect propagation at the project-level: results and a post-hoc analysis on inspection efficiency. *Empirical Software Engineering, 22(1)*, 57–79. Springer.
[40] Zhu, Y.-M. (2016). *Software Reading Techniques: Twenty Techniques for More Effective Software Review and Inspection*. Apress.

Ricardo Theis Geraldi is an assistant professor at the State University of Maringá (UEM) in the Informatics Department (DIN). He holds a master degree in computer science (UEM) (2015). His research experience includes software engineering, software inspection, software product line (SPL), variability management, java (JSE, JEE), web services, human-computer interaction (HCI) and web information retrieval.

Edson OliveiraJr is an assistant professor of software engineering at the State University of Maringá (DIN-UEM), Brazil. He holds a Ph.D. degree in computer science from the University of São Paulo (ICMC-USP), Brazil. He was a visitor scholar at the University of Waterloo (UW), Ontario, Canada in 2009. His research interests include: software product lines, software process lines, variability management, software architecture and evaluation, metrics, model-driven engineering, UML and metamodeling, and empirical software engineering. He acts regularly as an editorial board and a reviewer in peer-reviewed journals, and as a member of the technical program committee of several international peer-reviewed conferences.