Verification of Software Process Line Models: A Checklist-based Inspection Approach

[1]Eldânae Nogueira Teixeira, [1]Rafael Maiani de Mello, [1]Rebeca Campos Motta
[1]Cláudia L. M. Werner, [2]Aline Vasconcelos

[1]COPPE-Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
[2]Federal Fluminense Institute, Campos dos Goytacazes, Brazil
{danny, rmaiani, rmotta, werner}@cos.ufrj.br, apires@iff.edu.br

Abstract. A Software Process Line (SPrL) approach aims to support systematic process reuse by exploring the benefits of common aspects that exist in the process domain and managing its diversity, i.e., its variability. In this context, a SPrL must ensure the correctness, completeness and consistency among its artifacts and its related domain, in order to prevent the spreading of defects in its derived processes. For supporting quality assurance activities in SPrL Engineering, inspection is considered a relevant tool to detect defects in related artifacts through visual exam. However, the results from a recently conducted literature review pointed out the lack of approaches on supporting SPrLs inspections. In this paper, we present a checklist-based inspection technique (PVMCheck) for supporting the detection of defects on SPrL models, especially in process feature models represented using *OdysseyProcess-FEX* notation. An evaluation of PVMCheck is also presented, which results allowed us to identify its feasibility.

Keywords: Software Process Reuse, Software Process Line, Software Inspection.

1 Introduction

A great concern in organizations is related to the quality improvement of their products and services and meeting their customers' needs is a continuous challenge. This challenge includes the Software Development context in which a well-defined and followed development process is frequently considered as crucial for delivering high-quality products [1] [2]. Satisfying quality requirements entails that software processes must produce the expected results, be correctly defined, and any improvements made should be in accordance with the objectives of the enterprise, which may change quite often in highly competitive companies [3].

However, establishing a specific process from scratch for each project may not be effective and even the establishment of software processes is frequently considered a complex task. Process definition can be time consuming, error prone and cause negative consequences such as unnecessary activities that lead to a waste of time, the omission of necessary activities and the failure to comply with the organizational or international standards which may affect the quality of the final software product [4].

In this context, tailoring Software Reuse concepts [5] for supporting software processes reuse can be considered helpful on bringing similar benefits such as those produced when reusing software product artifacts [6]. Thus, Software Process Line (SPrL) [7] has emerged as an approach for software process reuse, based on the concepts of Software Product Lines (SPL) [8]. A SPrL can be defined as *"a set of processes in a particular domain or for a particular purpose, having common characteristics and built based upon common, reusable process assets"* [9]. The set of activities regarding the construction of SPrL typically includes *characterizing*, *managing* and *modeling* both similarities and differences between process families and process family members, *i.e.*, their *variability*. However, one can see that the addition of a reuse perspective in software processes modeling exposes them to a new range of anomalies, especially semantic defects.

Validation and Verification activities such as Software Inspections represent an important approach for ensuring models' quality before its implementation, simulation or execution, helping to deliver software quality [10]. Software Inspection can be defined as a visual exam of a software artifact in order to detect anomalies [11], especially preventing the introduction of semantic defects in the early stages of software projects, being considered one of the most efficient techniques for quality assurance [12]. Since the first version of a software inspection process [13], it has been evolved [14], and many *inspection techniques* have been developed for many types of software artifacts. These techniques are performed individually and supported by instruments such as *checklists* [15] [16] and guidelines as *reading techniques* [17] [18].

In order to support our research, a brief literature review was performed, searching for techniques of software process model verification. As a result, a lack of verification techniques concerned with SPrL inspection was observed, since the few approaches identified for detecting anomalies on software process models are not tailored for supporting software process reuse [19][20][21]. In fact, these approaches are typically designed for the syntactic model checking. Thus, it was observed that these approaches were unable to support the verification of whether a given SPrL model, even when correctly modeled from a syntactic perspective, correctly represents the semantic content of a specific process domain, see Section 3.

Therefore, this work proposes a checklist-based inspection technique named PVMCheck (Process Variability Modeling Checklist) for supporting the detection of semantic defects in SPrL variability models. PVMCheck was developed mainly based on the two technologies: *OdysseyProcess-FEX* [22], an SPrL meta-model and its notation, addressing process domain variability representation in process elements and relations and *FMCheck* [15], a checklist-based inspection technique for supporting the detection of semantic defects in SPLs represented through feature models.

After a proof of concept, PVMCheck feasibility was evaluated through a *quasi*-experiment. As a result, it was observed its feasibility, although it was not identified an equivalence of the effectiveness and the efficiency between the inspections performed by subjects having more experience and less experience in the study context.

This paper aims to present PVMCheck and its empirical evaluation. It is organized into five sections including this Introduction. Section 2 introduces the concepts of SPrL and process variability modeling, focusing on the *OdysseyProcess-FEX*. Section

3 presents the literature review performed in order to identify inspection approaches for detecting anomalies in software processes models in the context of process reuse. Section 4 describes the proposed inspection technique. Section 5 summarizes the results from a proof of concept conducted as a first evaluation of PVMCheck and the experimental study performed in order to evaluate its feasibility. Section 6 presents the conclusions and proposes steps in order to improve the proposed approach.

2 SPrL and Software Process Variability Modeling

In order to achieve effective process reusability, efficient methods for gathering the common and variable elements of specific processes and developing process definitions that can be applied in a variety of situations are necessary [23]. SPrL is an approach proposed for supporting the systematic reuse of processes applying similar principles established for SPL [8]. Thus, SPrL research aims at providing techniques and mechanisms for: (i) modeling existing similarities and variability in a software processes family; and (ii) supporting the customization of software processes according to specific needs of the software process domain [24]. An initial set of requirements for SPrL representation was identified and used in the development of the *OdysseyProcess-FEX* [22] meta-model and notation. The elements of the meta-model were defined through the analysis of different process models, such as OpenUP, RUP (Rational Unified Process), SPEM 2.0 meta-model [18], the process variability modeling literature [24][26][27][28] and also the feature modeling presented in SPL approaches, more specifically the Odyssey-FEX notation [29], which was developed based on well-known software feature model notations as FODA [30]. *OdysseyProcess-FEX* meta-model is composed by three packages: Main, Relationships, and Composition Rules [22]. The Main package describes the elements taxonomy, defining reusable process element categories (Work Units – Activity and Task, Role, Work Product and Tool) and their properties. The variations are described through the variability and optionality concepts. The first one distinguishes the domain elements between configurable (variation points and variants) and fixed (invariants). The optionality concept treats the mandatory or not presence of elements in the domain.

A group of relationships having relevant semantic concepts to processes representation is provided. The Alternative is the relationship that describes the relation between the configuration points and their alternatives (variation points and each of their variants). Other relationships provided are Aggregation, Composition and relations among work units, roles, work products and tools. Additionally, dependency and mutual exclusivity relationships between process elements are represented through inclusive and exclusive composition rules, respectively. A Composition Rule is composed by expressions that can be literal, an elementary expression that designates one single feature or boolean, which denotes a combination between features through the use of boolean operators: AND, OR, XOR, NOT.

A set of restrictions and properties were defined that, together, compose well-formed rules, the basis for syntactic model consistency verification.

Fig. 1 shows an excerpt from an example of the process feature model concerning a Project Management domain, integrating individual reference models of Scrum, XP,

OpenUP, and RUP [31]. One of the domain activities is Project Planning, which is composed by two mandatory tasks - Plan Project and Plan Iteration. The Plan Project task is a domain configuration point (variation point), which can be implemented by one of its optional variants: Plan Release or Plan the Entire Project. The Plan Project task produces the Project Plan and is performed by the Project Manager. Analyst and Stakeholders are additional and optional roles to support the task execution.

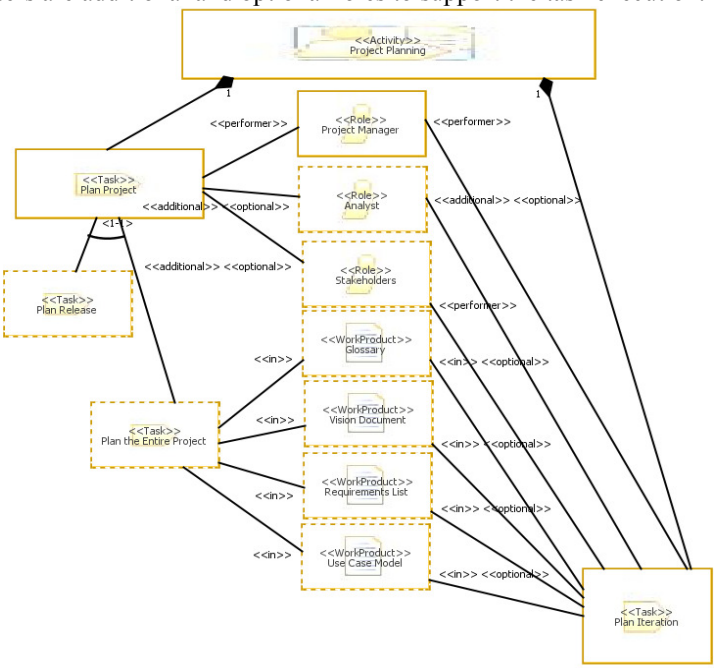

Fig. 1. Example: Project Planning Activity of Project management SPrL

3 Investigating approaches for inspecting software processes models designed for reuse

Software inspection is an effective practice for supporting detecting semantic defects in Software Engineering artifacts. Specialized literature presents a variety of inspection techniques for supporting this activity in diverse artifacts, such as requirements specifications [18], UML models [16][17] and Software Product Lines (SPL) [15].

In this sense, a literature review on inspection approaches in process reuse area was performed, especially concerned with SPrL, aiming at answering the following search question: "*What are the available techniques for inspecting software process artifacts designed for reuse?*". Due to the comprehensiveness of SCOPUS digital library observed when performing other systematic literature reviews (SLRs) [16][17], the authors established this search engine as the only source. In fact SCOPUS agglutinates publications from different and relevant sources in Software Engineering, such as ACM, IEEE and others. The search string was derived from previous knowledge and

experience on performing SLRs [15] [16] for the inspection context and it was reviewed by a SPrL specialist, in order to address the body of knowledge regarding software process reuse.

After the execution, 340 publications were retrieved. From this set, 22 were selected for a comprehensive evaluation, based on full paper reading. In other to evaluate the pertinence of these selected works for the research context, they were analyzed according to the following requirements (previously established by the researchers):

- Only approaches based on the visual examination of artifacts [11] are considered as inspection approaches, which excludes those specifying rules or heuristics without their formalization and /or execution, or supporting the detection using automated tools such as *model checkers*;
- The type of artifact inspected should be related to process, designed for reuse and possibly in the SPrL context;
- It is desirable that the inspection approaches had been evaluated.

After full papers reading, it was concluded that none of the selected papers fulfilled the two first criteria and also did not meet the third desirable one, showing the limitation in current approaches, especially related to SPrL inspection. For instance, Akbar et al. [19] present a process tailoring framework, not an inspection technique, whereas Martínez-Ruiz et al. [27] propose and evaluate a set of new variability constructs for software process engineering meta-model (SPEM) and no reference to possible defects in the notation are presented. Lopez-Herrejon et al. [20] present the concern of fixing inconsistencies in Software Product Line models, not addressing how to find them. Travassos et al. [21] use inspections for supporting the design and maintenance of software products through UML diagrams.

A previous work [15], developed by three authors involved in this work, was also retrieved in the SLR. It presents FMCheck, a checklist-technique to support detection of semantic defects in feature models describing SPLs. In this work, a *quasi*-systematic review (secondary study) was conducted to better understand the state-of-the-art of SPL inspections. It was concluded that there is a lack of technologies concerned with SPL semantic inspection, where the approaches identified were concerned with detecting anomalies in feature models using heuristics typically based on syntactic and automated model checking. These approaches are important to avoid the incorrect modelling, but are unable to support the verification of whether a given feature model is best suited to represent a particular domain. Then, a first version of FMCheck was evaluated through a proof of concept and then through a feasibility study *in vitro*, in which significant evidence regarding its feasibility was observed.

4 PVMCheck

For better interpreting the defects detected on inspections, a clear categorization of them is needed. The specialized literature presents different taxonomies for categorizing defects, and the one adopted by the presented approach is commonly applied for classifying defects detected in software models [17] [18] (Table 1.). Modeling SPrLs

requires a deep understanding of the domain. The quality of the domain artifacts is essential for the project success, since any non-conformity will be propagated to their derived processes. Into this context, the verification of the semantic adherence between the domain description and the SPrL models through inspections can be considered a relevant contribution on promoting this quality.

Table 1. Defect Types, adapted from [**Erro! Fonte de referência não encontrada.**].

Defect Category	Description
Omission	Necessary information about the domain that was omitted in the artifact.
Incorrect Fact	Some information in the artifact that contradicts information requirements specification or general knowledge domain.
Inconsistency	The information in certain part of the artifact is not consistent with other information.
Ambiguity	Some information is not clear, allowing multiple interpretations.
Extraneous Information	Some information in the model is out of scope.

In order to avoid defects propagation from SPrL models to its derived processes, early detection of defects in SPrL models is crucial. This section presents the steps followed for developing the first version of PVMCheck, a checklist based inspection technique for supporting the detection of defects on SPrL models. Since many issues suggesting defects can be identified on performing such inspections, the following subsection (Subsection 4.1) presents a first set of mapped discrepant cases, i.e., issues suggesting defects [18]. Subsection 4.2 presents the first version of PVMCheck.

4.1 Discrepant Cases

Based on the set of requirements for process variability representation identified in [22], Discrepant Cases (DCs) [16] were developed considering the defect types described in [**Erro! Fonte de referência não encontrada.**] (see Table 1.). DCs are sceneries that configure a discrepancy, which means a generic situation where a defect could be detected. Through the SPrL representations analysis, specially using *OdysseyProcess-FEX* notation, and its application in process domain modeling, the considered discrepancies were basically related to:

- Consistency – between the elements in the model;
- Clarity – regarding the interpretation;
- Correctness – when compared to the textual description in the notation;
- Completeness – when compared to the domain.

A total of 159 DCs were identified. Some of them can be considered in a broader context, addressing multiple notations. Other DCs are more specific to the *OdysseyProcess-FEX* meta-model and notation. *Inconsistencies* – cases such as misplacing antecedent with consequent in the model Composition Rules, *Incorrect Facts* – cases such as mistake mandatory with optional classification, and *Omissions* – cases such as an element or relationship were omitted – are examples of the most common defect

types that can be observed when comparing SPrL models with the domain description. This set of DCs identified doesn't intend to cover all possible scenarios of semantic defects that could be related to inspecting SPrL models. They can be organized in the following groups:

A. Process Elements: 57 DCs related to clarity and completeness of process element description and the correctness of process element category. These DCS are related to the individual analysis of each process element and consistency between them;
B. Relationships: 63 DCs refering to the representation of relationships between elements. This category checks if the relationships specified in the base document have been properly represented in the model;
C. Composition Rules: 33 DCs related to semantics rules of dependency and mutual exclusivity between process elements. The *OdysseyProcess-FEX* notation describes these restrictions as textual description of logical definitions involving two or more elements. So, these DCs verify the correct participation of all process elements involved and the correct representation as a compostion rule;
D. Optionality: 2 DCs related to the optionality / mandatory classification considering the whole domain as it is described in the base document; and
E. Variability: 4 DCs related with anomalies on the alternatives represented.

4.2 The Process Variability Model Checklist

Based on the DCs presented in Section 4.1 PVMCheck, a checklist-based inspection technique to support inspectors detecting defects in SPrL models was developed. This technique was tailored for supporting individual inspections performed by professionals that do not necessarily have previous domain knowledge to apply it, since textual description, such as domain specification, is defined as a base document (oracle) to be compared with the model during the inspection. It was developed based on software process variability representation requirements, also including particular items related with *Odyssey-ProcessFEX* meta-model and notation.

One challenge while developing a checklist concerns the amount of questions, because an excessive amount can lead inspectors to rework or weariness [16], but they still have to be representative in order to accomplish the inspection task. Each DC was created treating a specific situation of the meta-model. Some of them are very similiar, being covered by the same question in the checklist. Thus, trying to overcome the excessive amount of information of the checklist and considering its practical application, 34 checklist items were compiled from the 159 DCs. These items were distributed in the following verification groups: *Process Elements Verification*, *Relationships Verification* and *Composition Rules* V*erification*. The checklist items related to variability and optionality are transversal properties and were covered by verification items present in the three groups.

The checklist was structured in a spreadsheet with some orientation of how to apply the checklist for a group of verification items. It was described by the item number, the item description as questions and the possible answer alternatives: "Yes", "No" or "N.A" (*not applicable*, specifically for the model inspected).

Process elements verification is supported by 12 items, as presented in Table 2. These items cover the group A of DCs (Subsection 4.1). It aims detecting defects concerned with the clearness of the process elements attributes and descriptions and the correctness and completeness of the process elements that compose the domain scope.

Table 2. Items for process elements verification.

Id	Process Elements Verification Items
1	Are all process elements clearly described?
2	Is there only necessary and sufficient information describing each process element from the model, *i.e.,* there isn't unnecessary information?
3	Is there any process element, although correct, out of the domain scope?
4	Is there any process element from the domain scope that has been omitted from the model?
5	Is there any attribute representing the description of a process element that has been omitted from the model or has been incompletely described?
6	Are there process elements in the model that represent the same element in the domain?
7	Is there any process element that has been modeled isolated from the other elements?
8	Is the set of abilities and competences from each process element classified as role clearly described and in conformance with the domain?
9	Are the optionality/ mandatory of the process elements from the model compliant with the domain description?
10	Is there any incorrect comment associated to a domain element?
11	Is there any comment that does not add relevant information to the domain?
12	Is there any comment that should be represented as process element or attribute?

The relationship verification is composed by 14 items presented in Table 3 representing group B of DCs. It aims to verify the correct association between elements verifying the restrictions imposed by the meta-model and the domain scope and the correct classification according to the elements categories and the semantic of the relationship. Composition rules verification is composed by eight items presented in Table 4 supporting the group C of DCs.

Table 3. Items for Relationship Verification

Id	Relationship Verification Items
13	Is there any relationship between process elements that has not been included in the model?
14	Is there any relationship between process elements that has not been compliant with the domain description?
15	Are all relationships and their attributes represented in the model clearly described and compliant with the domain?
16	Are all aggregation and composition relationships between process elements correctly represented in the model?
17	Are all domain configurations that can be represented by a set of alternative relationships (relations between variation point and its variants) correctly represented in the model with the elements correctly connected and their variability classification correctly defined?
18	According to the domain, are the cardinalities (minimum and maximum values) of the variation points correctly represented in the model?
19	Is the classification of optionality from a relationship between process elements compliant with the participation type established by the domain?

Id	Relationship Verification Items
20	Is there any relationship in which the variation point (origin) and its variants (destiny) are not classified with the same process element category?
21	Is there any relationship in which its origin does not have the destiny represented by an adequate category?
22	Is there any role, work product or tool not related to a work unit?
23	Is the optionality classification of each relationship compliant with the domain?
24	Is there any role in which its responsibility is not compliant with its participation in work execution established by the domain?
25	Is there any work unit without a work product declared as its output?
26	Are the work products associated to a work unit correctly classified as inputs (in), outputs (out) or modified (in/out) compliant with the domain?

Table 4. Items for Composition Rules Verification.

Id	Composition Rules Verification Items
27	Are all domain composition rules represented in the model?
28	Is there any composition rule or expression in the model that is out of the domain scope?
29	Are all composition rules clearly described and compliant with the domain description?
30	Is there any inclusive composition rule that has been represented as an exclusive composition rule (or vice-versa)?
31	Is there any composition rule inconsistent in the model?
32	Is there any composition rule with an antecedent expression represented as a consequent expression (or vice-versa)?
33	Are all the expressions having inclusive composition rule compliant regarding their mandatory?
34	Is there any exclusive composition rule in which its consequent expression includes process elements classified as mandatory?

5 Evaluation of PVMCheck

The Evaluation of PVMCheck feasibility was performed through two activities: a *proof of concept* and a *quasi-experiment*. Two members (P1 and P2) from the Experimental Software Engineering Group at COPPE/UFRJ participated in the proof of concept. Both subjects declared relevant experience on developing and applying software processes, but limited knowledge regarding software process reuse and variability modeling. After a brief introduction to *OdysseyProcess-FEX* and PVMCheck, these subjects were invited to apply this proposed checklist for individually inspecting the same SPrL model.

Analyzing the results, it was observed that the subject with more experience on inspections (P1) detected fewer defects (5) than the other subject (P2, 10 defects), although the time dedicated for each inspection was similar (P1, 120; P2, 127). One reason for these unexpected results could be found in the subjects answers to a follow up questionnaire, in which it was observed that P2 considers that PVMCheck contributed significantly for its performance and for its learning on software inspection while P1 has considered its performance was significantly hampered by the insufficient content presented during the preparation regarding the technologies involved. However, P2 also pointed out the need of improving the description of our checklist items

and also the need of improving the training session. Thus, based on the results observed in the proof of concept, improvements to the training session and the description of checklist' items were performed.

Then, an experimental study (*quasi*-experiment) was planned for evaluating the feasibility of PVMCheck. Based on [32] the following study goal was defined: to analyze the inspection of SPrL models using PVMCheck in order to characterize, *with respect to* its effectiveness (defects identified/ total existing defects) and efficiency (identified defects/ time) in identifying defects and the opinion of the inspectors *from the perspective of* Software Engineering researchers *in the context of* software developers (represented by undergraduate and graduate students from a Software course at NES/IFluminense and Software Reuse Group at COPPE/UFRJ) inspecting SPrL models in two different application domains. Considering the restriction of applying a single task to each subject and the reduced influence of the inspector experience on his/her effectiveness and efficiency when using a checklist [33], the following null hypotheses were established:

$H_0$1: There is no equivalence between the *effectiveness* of SPrL inspections performed by the group composed by more experienced subjects and the group composed by less experienced subjects.

$H_0$2: There is no equivalence between the *efficiency* of SPrL inspections performed by the group composed by more experienced subjects and the group composed by less experienced subjects.

The study sample was composed by 18 subjects, 17 students a Software course at NES/IFluminense and one student from Reuse Group at COPPE/UFRJ, who signed a consent form and filled in a characterization form. The experience level from each subject was evaluated considering the following attributes: *academic degree; experience with software processes* (developing, applying and reviewing); *experience with reuse* (feature models, Odyssey-FEX, SPrL); *experience with software inspections in general*. After analyzing the distribution of experience level between all subjects, it was identified the opportunity of amalgamating these subjects into two distinct groups normally distributed (Saphiro-Wilk test): *group A*, composed by the more experienced subjects and *group B*, composed by the less experienced subjects. Then, it was identified that these groups are significantly different (Student t- test, p-value=0.99). This composition is especially relevant in the context of the alternative hypotheses investigated in this study (equivalence of effectiveness and efficiency between groups).

All subjects were trained in software inspection and domain description through process feature models using *OdysseyProcess-FEX* notation and in the application of PVMCheck. Two domain specifications and their respective SPrLs' model regarding the Project Planning activity designed using *OdysseyProcess-FEX* notation were used as instruments: D1, a more complex domain combining *Scrum* and *OpenUP* (D1) and D2, a simpler domain combining of *OpenUP Basic* and *OpenUP* (D2). These domains are variations of the one described in Section 2, Fig. 1. Then, nine subjects were randomly selected to inspect each domain. However, after the execution, it was observed that only 13 reports had the identification of the related subjected.

The discrepancies reported by the 13 subjects were reviewed by two researchers that classified each one as *defect* or *false positive.* The type of each reported defect

was also reviewed and reclassified, when needed. As can be seen at Table 5, none of the subjects caught the higher level of effectiveness (1) and three of them didn't detect any defect. It is important to highlight that the domain distribution between subjects was random.

Table 5. Execution Results

Grp.	Subj.	Domain	Exp. Level	Time	#Defects	Efficiency	Effectiveness
A	JCC	D2	0.604	90	0	0	0
	GCBC	D2	0.479	91	7	0.077	0.5
	RMS	D2	0.458	53	2	0.038	0.143
	RSM	D1	0.396	88	4	0.045	0.308
	RNG	D2	0.271	98	0	0	0
	ROM	D2	0.271	54	7	0.13	0.5
B	JGA	D1	0.208	68	0	0	0
	RVS	D1	0.188	50	2	0.04	0.154
	TCZ	D2	0.167	79	4	0.051	0.286
	TSC	D2	0.167	74	1	0.014	0.071
	LPD	D2	0.167	74	2	0.027	0.143
	LFS	D1	0.146	55	4	0.073	0.308
	PAS	D2	0.125	92	3	0.033	0.214

It was observed that the distribution of *effectiveness* and *efficiency* by groups (**Erro! Fonte de referência não encontrada.**) were normal (Saphiro-Wilk test) and no outlier in any distribution was detected. However, it was also identified that the variance in group A for both variables was significantly different from group B. In addition, it was not observed a significant influence regarding the domain complexity over the results.

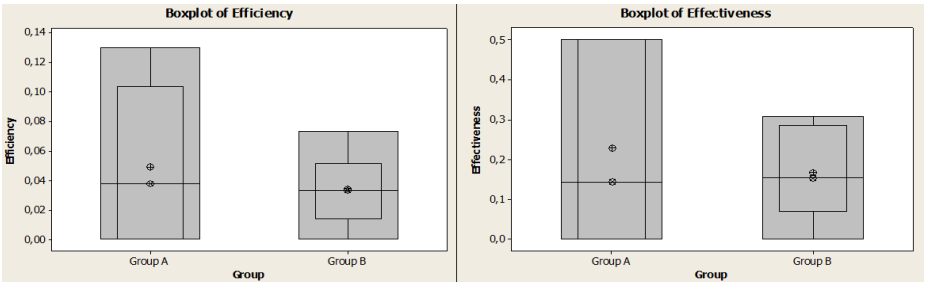

Fig. 2. Distribution of Efficiency and Effectivenes in groups A and B

By applying the Equivalence test with a threshold of non-similarity of 10% (JMP tool), it was not observed equivalence between the distributions of effectiveness and efficiency in groups A and B. Thus, it was not possible to reject **$H_0$1** and **$H_0$2.**

Considering the inspections' results as a whole, it was observed that the inspectors were not able to detect all known defects in both domains (69% of D1 and 71% of D2). However, it is important to highlight that the simpler domain (D2) was inspected by nine subjects and five of them are from group (A) while the more complex domain (D1) was inspected by only four subjects, three of them from group (B). These unex-

pected results can be explained due to the fact that only one, from the nine defects detected in D1, was reported by more than one subject while eight, from the 10 defects detected in D2, were reported by more than one inspector.

The subjects filled an evaluation form after their inspection. Most of subjects agree or partially agree that the given instruments were helpful to support the activities and both *OdysseyProcess-FEX* and PVMCheck items were easy to understand. Most of them also agree or partially agree in applying PVMCheck for future inspections since the inspection technique helped them to detected defects. However, most of them disagree or partially disagree that their time were better applied using PVMCheck. Five subjects also suggested improving the training session while five of them suggested improvement in the checklist, including reorganizing its questions and reducing the amount of questions.

5.1 Threats to Validity

As threats to the validity of this *quasi*-experiment, the small sample size and the limited experience of the subjects in inspection and software process reuse should be considered. In addition, the limited number of inspected domains is another aspect that can be pointed out. However, it is worth noting that no participant inspected the same domain more than once. The absence of a complete list of defects specified prior to study execution can also be considered as a threat to the validity. Some defects were considered as 'known defects', which were detected during this quasi-experiment. This directly affects the calculation of the inspections efficacy. It is also important to note that this in vitro study was performed synchronously, with the participants using the same resources (printed artifacts) to conduct their inspections. As an internal threat, one can consider the fact that the same group that has developed the technology evaluated it, even based on objective experimental practices. As an external threat, it can be considered the definition of the study sampling. The population was defined by convenience, typical in *quasi*-experiments [34].

6 Conclusion

Considering the lack of approaches observed in the specialized literature for supporting the detection of semantic defects on software process models, including SPrLs, this paper proposed PVMCheck, a checklist-based inspection technique for supporting individual inspections on SPrLs. A preliminary version of the technique was submitted to a proof of concept and then an improved new version of the checklist was submitted to a *quasi*-experiment, in which its feasibility was observed, although an equivalence of the effectiveness and the efficiency between the inspections performed by subjects having more experience and less experience was not identified in the study context. As next step, PVMCheck will be improved based on the feedback provided by the subjects and then a new trial of the presented study will be performed. We also intend to compare the effectiveness and efficiency of PVMCheck with *ah hoc* inspections and to evolve the Odyssey environment to support SPrL verification

activities based on PVMCheck. Other future works are related to the improvement of the whole SPrL verification context, automatizing its syntactic verification.

Acknowledgements

We would like to thank the proof of concept and *quasi*-experiment participants and CAPES, CNPq and FAPERJ for financial support. We also thank to the Software Reuse Group and Experimental Software Engineering Group at COPPE/UFRJ and NES/IFluminense for supporting this work.

References

1. Osterweil, L., 1987, “Software Processes Are Software Too”. In: Proceedings of the 9th International Conference on Software Engineering, pp. 2-13, Monterey, USA.
2. Fuggeta, A., 2000, "Software process: a roadmap". In: Proceedings of the Conference on The Future of Software Engineering, pp. 25-34, Limerick, Ireland.
3. García, F. et al. (2006). FMESP: Framework for the modeling and evaluation of software processes. *Journal of Systems Architecture*, *52*(11), 627-639.
4. Pedreira O., Piattini, M., Luaces, M.R., et al.: A systematic review of software process tailoring. SIGSOFT Software Engineering Notes. 32, 1–6 (2007).
5. Frakes, W.B., Kyo Kang, K., 2005, “Software Reuse Research: Status and Future”, Journal of IEEE Transactions on Software Engineering, v. 31, n.7, July.
6. Barreto, A., Murta, L., Rocha, A.R., 2011, “Software Process Definition: a Reuse-Based Approach”. Journal of Universal Computer Science 17(13), 1765–1799.
7. Rombach, D., 2013, “Integrated Software Process and Product Lines”. In Perspectives on the Future of Software Engineering, pp. 359-366. Springer Berlin Heidelberg.
8. Northrop, L., 2002, “SEI’s Software Product Line Tenets”, IEEE Software, v.19, n.4, pp. 32-40, July/August.
9. Washizaki, H., 2006, “Building software process line architectures from bottom up”. In: Proceedings of the 7th International Conference on Product-Focused Software Process Improvement, pp. 415-421, Amsterdam, Netherlands, June.
10. Collofello, J. S., 1988, “Introduction to software verification and validation”, No. CMU/SEI-CM-13-1-1, Carnegie-Mellon University, Pittsburgh, Soft. Eng. Institute.
11. IEEE., 2008, “STD 1028-2008: IEEE Standard for Software Reviews and Audit”.
12. Denger, C., Kolb, R, 2006, “Testing and inspecting reusable product line components: First empirical results”. Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering, Rio de Janeiro, Brazil.
13. Fagan, M. E., 1976, “Design and Code inspections to reduce errors in program development”. IBM Systems Journal 15 (3): pp. 182–211.
14. Laitenberger, O., 2002, “A survey of software inspection technologies”. Handbook on Soft. Eng. and Knowledge Engineering, Vol. 2, pp. 517–555. World Scientific Publishing.
15. De Mello et al., 2014, “Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections”. Journal of Universal Computer Science, 20(5), 720-745.
16. De Mello, R. M., Pereira, W. M., Travassos, G. H., 2010, “Activity Diagram Inspection on Requirements Specification”. In Brazilian Symposium on Soft. Eng, pp. 168-177, IEEE.

17. Travassos, G., Shull, F., Fredericks, M., Basili, V. R., 1999, “Detecting defects in object-oriented designs: using reading techniques to increase software quality”. In ACM Sigplan Notices, Vol. 34, No. 10, pp. 47-56.
18. Shull, F., Rus I., Basili, V., 2000, “How Perspective-Based Reading can Improve Requirements Inspections”, IEEE Computer, vol. 33, no. 7, pp. 73-79.
19. Akbar, R., Hassan, M. F., Abdullah, A., 2012, “A framework of soft-ware process tailoring for small and medium size IT companies”. In: Computer & Information Science, International Conference on, Malaysia, IEEE, pp. 914-918.
20. Lopez-Herrejon, R. E., Egyed, A., 2012, "Towards fixing inconsistencies in models with variability." Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems. ACM.
21. Travassos, G. H., Shull, F., Carver, J., 2001, "Working with UML: A software design process based on inspections for the unified modeling language." Advances in Computers Book Series, Volume 54, n.1, pp. 35-97.
22. Teixeira, E.N., 2014, “A Component-Based Software Process Line Engineering with Variability Management in Multiple Perspectives”, In: 18th International Software Product Line Conference Doctoral Symposium, Florence, Italy.
23. Hollenbach, C., Frakes, W., 1996, “Software Process Reuse in Industrial Setting”. In 4th International Conference on Software Reuse, Orlando, Florida, USA , pp. 22-30.
24. Junior, E. A. O. et al., 2013, “SMartySPEM: A SPEM-Based Approach for Variability Management in Software Process Lines”. In Product-Focused Software Process Improvement, pp. 169-183, Springer Berlin Heidelberg.
25. Martínez-Ruiz, T., et al., 2012, “Requirements and constructors for tailoring software processes: a systematic literature review”. In: Software Quality Journal, 20, 1, 229–260.
26. OMG, 2008: “Software Process Engineering Meta-model”, In: http://www.omg.org/technology/documents/formal/spem.htm
27. Martínez-Ruiz, T. et al. 2011, "Modeling software process variability: an empirical study", In: Software, IET, 5, 2, 172,187.
28. Alegría, J.A.H., Bastarrica, M.C., 2012, “Building software process lines with CASPER”, In: Proceedings of International Conference on Software and System Process, Zurich, Switzerland, IEEE, pp. 170 - 179.
29. Fernanders, P., Werner, C., 2008, “Ubifex: Modeling context aware software product lines”. In 2nd International Workshop on Dynamic Soft. Product Line Conf., pp. 3-8.
30. Kang, K. C. et al., 1990, A. S. Feature-Oriented Domain Analysis (FODA) Feasibility Study”. Technical Report CMU/SEI-90-TR-21/ ESD-90-TR-222.
31. Magdaleno, A. M., Araujo, R. M., Werner, C. M. L., 2012, “COMPOOTIM: An Approach to Software Processes Composition and Optimization”. In: Congresso Ibero-Americano em Engenharia de Software, Buenos Aires, Argentina, pp. 1-14.
32. Basili, V., Caldiera, G., Rombach, H., 1994, “Goal Question Metric Paradigm”, Encyclopedia of Software Engineering, v. 1, John Wiley & Sons, pp. 528-532.
33. Biffl, S. and Halling, M. “Investigating the influence of inspector capability factors with four inspection techniques on inspection performance.” Eighth IEEE Symposium on Software Metrics, 2002. IEEE, 2002.
34. De Mello, R. M., Travassos, G. H.. “An ecological perspective towards the evolution of quantitative studies in Software Engineering”. In 17th Intl. Conf. on Evaluation and Assessment in Software Engineering (EASE), pp. 216-219, 2013.