Journal of Universal Computer Science, vol. 20, no. 5 (2014), 720-745
submitted: 28/7/13, accepted: 15/2/14, appeared: 1/5/14 © J.UCS

Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections

Rafael Maiani de Mello
(COPPE-Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
rmaiani@cos.ufrj.br)

Eldânae Nogueira Teixeira
(COPPE-Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
danny@cos.ufrj.br)

Marcelo Schots
(COPPE-Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
schots@cos.ufrj.br)

Cláudia Maria Lima Werner
(COPPE-Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
werner@cos.ufrj.br)

Guilherme Horta Travassos
(COPPE-Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
ght@cos.ufrj.br)

Abstract: Software Product Line Engineering (SPL) should ensure the correctness, completeness and consistency of its artefacts and related domain to prevent the propagation of defects in derived products. Software inspection techniques are effective in detecting defects in software artefacts and avoiding their propagation throughout the software development process. However, the results of a *quasi*-systematic review of the technical literature reported in this paper pointed to a lack of such techniques to support the inspection of SPL artefacts, including techniques to support the inspection of feature models (FMs) that are largely used in domain modelling. Therefore, a checklist-based inspection technique (FMCheck) has been developed to support the detection of defects on FMs. FMCheck is configurable and can be applied to the original feature model notation (the FODA approach) and its extensions, including the Odyssey-FEX notation. The inspection technique was empirically evaluated, having indicated its feasibility and effectiveness. It is possible to see that inspectors applying FMCheck to inspect FMs can be more effective than those applying *ad-hoc* techniques, regarding four distinct domains.

Keywords: Feature Model, Software Inspection, Domain Engineering, Software Reuse, Software Product Line, Experimental Software Engineering.
Categories: D.2.1, D.2.2, D.2.4, D.2.13

1 Introduction

The systematic reuse of software artefacts has been used in the last decades, and amongst its observed benefits, an increase of quality and productivity in the software

development process seems to be produced by its use [Lung et al., 97] [Jones, 00]. In this context, a Software Product Line (SPL) is a key approach to support software reuse, aiming to support its systematic accomplishment in all software development stages. SPL represents a group of software-intensive systems sharing a common, managed set of features. Software products meet the specific needs of a particular market or mission and are developed from a common set of core assets in a prescribed way [Northrop, 02].

SPL Engineering can be divided into two stages: Domain Engineering (DE), focused on the development *for* reuse, and Application Engineering (AE), focused on the development *with* reuse. Software development *for* reuse includes a set of specific activities aimed at converting knowledge into reusable components [Arango and Prieto-Díaz, 91]. The DE stage consists of Domain Analysis, Domain Design, and Domain Implementation [Atkinson et al., 02]. Both Domain Analysis and Domain Design can be modelled by using different kinds of representations, which includes UML models, architectural models [Gomaa and Shin, 07] (commonly using UML component diagrams) and one of the representation techniques most used in SPL approaches, i.e., feature models (FMs) [Kang et al., 90]. A FM intends to express domain requirements as features, that can be specified as prominent or distinctive, and user-visible aspects, qualities, or characteristics of a software system [Kang et al., 90]. Several notations have been derived to represent feature models based on the original one (FODA), including the Odyssey-FEX notation [Blois et al., 06].

Although SPL Engineering can use conventional Software Engineering models, the addition of a reuse perspective can expose SPL artefacts to a new range of anomalies, including semantic defects which detection could be supported by software inspections. IEEE [IEEE, 08] defines software inspection as the visual exam of an artefact to find defects, a concept introduced by Fagan [Fagan, 72] and considered an approach for performing software reviews [Wong, 06]. Inspections on software artefacts can be supported by *ad-hoc* techniques or more elaborated ones such as checklists [de Mello et al., 10] and reading techniques [Travassos et al., 99; Shull et al., 00]. Through inspections, a considerable rework on software development can be avoided, because defects can be identified and consequently fixed on early stages of software development [Shull and Seaman, 08]. In this context, specialized literature suggests that the inspection of SPL artefacts from a reuse perspective can reduce the efforts in redundant verification of software products [Denger and Kolb, 06], benefiting all the products derived from it [McGregor, 01].

Thus, to better understand the state-of-the-art of SPL inspections, we carried out a *quasi*-systematic review (secondary study) [Travassos et al., 08] of the technical literature. Results indicated that there is a lack of technologies concerned with SPL inspection. In particular, we could not identify any inspection technique to support the semantic reading of FMs. Although some approaches for detecting anomalies in FMs are common [Benavides et al., 10], their heuristics are typically based on syntactic and automated model-checking, not supporting the identification of semantic defects. These approaches are important to avoid the incorrect modelling of FMs and support the development of SPLs, but are unable to support the verification of whether a given FM, correctly modelled, is best suited to represent a particular domain. Thus, aiming at filling this gap, we developed FMCheck, a checklist-based inspection technique to

support the semantic verification of FMs. Its feasibility and effectiveness in comparison to *ad-hoc* inspections were empirically observed [de Mello, 12].

This paper is an extension of [de Mello et al., 12] that only described FMCheck and its evaluation without presenting how the mentioned gap of inspection techniques for supporting the verification of SPL artefacts, including FMs, was investigated. Thus, this paper describes the *quasi*-systematic review conducted in 2011 before the development of FMCheck, discussing its results and showing new information produced in an additional search trial in 2013. Besides, the paper also brings additional information regarding the checklist and its empirical evaluation that was not completely described in [de Mello et al., 12]. Therefore, the paper is organized in seven sections, including this introduction. Section 2 presents the planning and results from the *quasi*-systematic review undertook to identify inspection techniques for SPL, including new results acquired in a supplementary search trial. Section 3 discusses the basis for establishing FMCheck, a checklist based inspection technique to support the verification of FMs, which is presented in Section 4. Section 5 describes the planning and results of the studies conducted to evaluate FMCheck, and Sections 6 and 7 present, respectively, our conclusions and some paths for future work.

2 The *quasi*-Systematic Review

The *quasi*-Systematic review aimed at identifying inspection technologies concerned with SPL in the technical literature. Its research protocol was based in [Biolchini et al., 05] and the used control paper was [Vasconcelos and Werner, 07].

Our research question was: '*What are the existing techniques for inspecting software artefacts developed for reuse?'* To conduct this review, we defined specific criteria and procedures for article pre-selection (see Section 2.4) and selection (see Section 2.5), involving four researchers from the Software Engineering Group at COPPE/UFRJ. The PICO approach supported the definition of the search string (see Section 2.1) based on the established research question:

- *Population:* articles describing inspection techniques concerned with SPL artefacts.
- *Intervention:* techniques to support defect detection on reusable artefacts and/or on their reuse.
- *Comparison:* none.
- *Outcome:* the techniques, their heuristics and applicability.

The next subsections describe the review plan and its results.

2.1 Search String (SCOPUS syntax)

TITLE-ABS-KEY (inspect OR review* OR verif* OR validat* OR evaluat* OR assess* OR read* OR check*) AND TITLE-ABS-KEY("domain model" OR "domain analysis" OR "domain design" OR "feature model" OR "architectural element" OR "architectural model" OR "reusable component" OR "software component" OR "variability model" OR "software architecture" OR "decision model") AND TITLE-ABS-KEY("Domain Engineering" OR "software product line" OR "software product family" OR SPL OR "software reuse" OR "variability management" OR "software*

factory")

2.2 Source Selection

For search tools, the following selection criteria were defined:

- The search tool should retrieve a unique result for a specific search string;
- The search tool should allow the application of the search string only on the title, abstract, and keywords;
- The search tool should support the whole search string in a single query.

Three search engines were considered: SCOPUS (http://www.scopus.com/), EI Compendex (http://www.engineeringvillage.com), and IEEEXplore (http://ieeexplore.ieee.org). The intention on using such search engines was as regards to the high coverage they offer. For instance, SCOPUS indexes articles from different sources including ACM, IEEE and others, besides its well-known stability, reliability and interoperability with different referencing systems. Additionally, we included the proceedings of Brazilian Software Engineering conferences, available from BDBComp (http://www.lbd.dcc.ufmg.br/bdbcomp/) as a source of knowledge for this research. The three selection criteria were applied to the search engines. Based on them, the IEEEXplore was discarded due to the third selection criterion. Theoretical papers, reports of application in industry, experimental studies and/or combinations between them would be accepted if they had been published in journals or conferences.

2.3 Paper Selection Procedures (Experimental Design)

Two reviewers carried out the search and individually rated each returned paper as "pre-selected", "excluded" or "undefined", following the pre-selection criteria described in the section below. If a paper got rated as "undefined" by both reviewers, it would be pre-selected. All the pre-selected papers were fully read by the first and by a third reviewer, ranking each paper as "selected", "excluded" or "undefined", after the selection criteria described in subsection 2.5.

2.4 Paper Pre-Selection Criteria

- The paper should meet the search string in its abstract, title, and keywords.
- The abstract should suggest the characterization and/or application of an approach to evaluate artefacts related to software reuse, excluding explicit citations to testing and measurement approaches.
- The full paper should preferably be available for download. If not, an email for two of the authors should be sent with an answer expectation of 20 days, after which the paper would be discarded.
- The title and abstract should be written in English or Portuguese due the inclusion of BDBComp.

2.5 Selection Criteria

- Papers should be written in English or Portuguese;
- The full reading of the papers should allow the identification of at least one approach to support the inspection of reusable artefacts;
- Papers that do not characterize (partially or totally) the inspection approach used, without even citing an existing one, should be excluded;
- Papers describing approaches that do not present inspections as a visual exam of an artefact (according to the definition of IEEE 1028 standard [IEEE, 08]), such as approaches based on automated test tools, should be excluded;
- Since we are looking for evaluating already developed inspection techniques, papers that only mention rules or heuristics to identify software defects without organizing it in a technique should be excluded, although they could be considered to support the construction of a new inspection technique;
- Papers that do not present an approach to identify defects in software artefacts should be excluded.

2.6 Execution

After two test rounds, the final search string as presented on subsection 2.1 was executed on May, 2011 and re-executed on July, 2013 for the selected sources (SCOPUS and EI Compendex). Some registers were discarded due to not being papers, but only abstracts from proceedings. In the first search trial (2011), from the 350 distinct papers returned, 304 were referred to as "Computer Science" papers, suggesting a good calibration of the search string. In its current re-execution (2013), 165 more distinct papers were returned, as shown in Table 1. Numbers in brackets show the number of results found only for the second search trial and the numbers outside brackets show the number of results only for the first search trial.

Search Tool	#Registers	#Papers	#Distinct	#Pre-Selected	#Selected
SCOPUS	375(178)	348(165)	350(165)	100(34)	4(1)
EI Compendex	207(60)	193(0)			

Table 1: Summary of returned publications by search tools.

At the end of the first search trial, it was a consensus amongst the reviewers that only four papers, referring to three distinct inspection approaches, should be selected, as summarized in Table 2. Also, by applying the same criteria to the BDBComp proceedings, only one paper [Vasconcelos and Werner, 08] was found, having also been mentioned in Table 2. Finally, after the second search trial (2013), only the paper describing the Inspection Technique presented in Section 4 [de Mello et al., 12] was added to the list of selected papers.

In [Ortega et al., 07], a small checklist is provided as an example of how to inspect requirements in an approach to certify the quality of reusable components, but it neither describes its application nor refers to any further study. The extended ArqCheck [Vasconcelos and Werner, 07] consists of a configurable inspection

technique to support the detection of defects in architectural models, independent from the modelling notation [Barcelos and Travassos, 06]. In the extended version of ArqCheck, five evaluation items related to check the compliance between the architectural model and non-functional requirements were added in a reusability perspective, as shown in Table 3. The papers related to ArqCheck describe two *quasi*-experiments conducted to evaluate it. The first one identified evidence on the feasibility of ArqCheck as an inspection technique, and the second describes the evaluation of the second and current version of ArqCheck.

Paper	**[Ortega et al., 07]**	**[Kim et al., 08]**	**[Vasconcelos and Werner, 07;08;11]**
Approach Name	No name	No name	ArqCheck (extended version)
Reference describing the approach	Own paper (partially)	Own paper (partially)	Own papers (extended version) and [Barcelos and Travassos, 06]
Inspected artefacts	Requirements Specification	Architectural models	Architectural models
Specific for reuse?	Undefined	Yes	Yes
Type of technique	Checklist	Checklist	Checklist
Review Focus	Semantic	Semantic	Semantic
Reporting of application?	No	Proof of Concept	Academic Projects
Approach Evaluated?	Yes, specialists evaluation	Yes, specialists evaluation	Yes. *quasi*-Experiments

Table 2: Summarized data extracted from the selected papers (first search trial).

The technique proposed in [Kim et al., 08] is based on the Families' Evaluation Framework (FEF) [Schimid and van der Linden, 07], a specific architectural framework based on PuLSE^{TM} (the Product Line Software Engineering method) [Schimid and Widen, 00] approach. This technique aims to verify if the practices established in these frameworks are being applied to a model. The evaluation items of this technique were grouped in four checklists, each one focusing on a desirable architectural attribute of SPLs: *commonality and variability*; *layered architecture*; *abstraction mechanism*; *and default interface*. Table 4 provides the checklist for standard interface evaluation. At the end of this evaluation, the approach leads to the counting of non-conformities for each checklist, aiming to establish a grade of achievement for each attribute, as follows: not achieved (<25%), partially achieved (between 25% and 50%), largely achieved (between 50% and 85%) and fully achieved (>85%). Although the authors do not mention terms such as 'inspection' or 'defects', they present evaluation items for the detection of semantic defects in the referred context.

Items that evaluate consistency in the architectural representation
Do the responsibilities of the internal modules (i.e., classes) of a reusable element belong to the same context, i.e., do they intend to achieve the same goal or are they used in the same use case scenarios?
Is it possible to identify groups of reusable architectural elements with similar responsibilities or that share some common implemented functionalities that should be grouped to form a component?
From the point-of-view of the concept that the reusable architectural element represents, are there modules (i.e., classes) that should be allocated in it, considering their responsibilities or functionalities, but that are allocated in another architectural element?
Are there couplings between a reusable architectural element and other elements that hinder its reuse?
Considering the coupling amongst reusable architectural elements, are there couplings that justify their clustering into one component?

Table 3: Excerpt from the extended ArqCheck [Vasconcelos and Werner, 07]

Classification	Items
Definition of standard interface	Are there standard interfaces defined between layers and components?
	Are there rules for interface standardization? For instance, interface documenting guide and naming convention.
Design of standard interface	Are inner functions and outer functions (interface) separated?
	Are interface parts separated from their implementation parts?
Implementation of standard interface	Are function calls between components made with only standard interfaces (i.e., no direct access)?

Table 4: Checklist for Standard Interface Evaluation [Kim et al., 08].

So far, the data extracted from the selected papers suggest that only architectural models had inspection techniques in the SPL context, whereas only one was experimentally evaluated.

2.7 Threats to Validity

The search string used cannot be considered as complete, as the approaches for product lines found in the technical literature may cite a specific artefact instead of mentioning a generic term related to SPL. Also, we could not have previously know all the distinct names that could be used to identify each stage of the SPL process and used generic terms in the second part of the search string. However, as mentioned in the previous subsection, we observed a good calibration of the search string in the first trial, which also happened in the second search trial.

3 Developing an Inspection Technique for Feature Models

Considering the results of the *quasi*-systematic review described in Section 2, which indicated the lack of inspection techniques focusing on artefacts produced for Domain Engineering (DE), and our experience on developing reading techniques for different problem domains, we assert the following research initiatives to start the organization of a body of inspection techniques to support verification activities of SPL:

- Evolve the support for architectural model inspection, which can be done by developing more specific heuristics for detecting defects based on [Kim et al., 08] to the extended version of ArqCheck;
- As observed for ArqCheck, extend other inspection techniques such as OORTs (Object Oriented Reading Techniques) [Travassos et al., 99] and PBR (Perspective-Based Reading) [Shull et al., 00] to lead to reusability. For example, OORTs deal with the inspection of UML models such as class diagrams and state machine diagrams, both identified as models for DE [Gomaa and Shin, 07];
- Develop an inspection technique to support defect detection in FMs, since we did not find any inspection technique to support the detection of defects for this relevant SPL artefact.

Overlooked defects are an unavoidable aspect of any software development. As the SPL approach is used as the basis for deriving many products, the early detection of defects is crucial to avoid their propagation in subsequent development stages to different applications derived from them. The FM is created as a result of Domain Analysis, the first step in the DE stage. Thus, considering the relevance of this representation for DE and its high level of abstraction, we decided to follow the third initiative, developing a checklist-based inspection technique. The following subsections describe the body of concepts that represent the basis from which the verification items of the checklist were derived. These include the common concepts related to recent notations of a FM and the set of possible defects that can be detected in a FM when it is compared to its respective domain description.

3.1 Feature Model Notations

One of the ways to specify the acquired domain knowledge (also known as variability modelling) is by feature modelling, a high-level abstraction that aims to describe the domain requirements based on the concept of features, gathering its commonality and variability. Software variability is the ability of a software system or artefact to be changed, customized, or configured for use in a particular context [Gurp et al., 01]. Variability can be defined as points in the core assets of the domain where it is necessary to differentiate individual characteristics of software products, i.e., configuration points in the domain. This concept is represented in a FM using the following elements: (1) variation points, which establish the need of decision-making related to one feature, regarding which variants will be used; (2) variants, available choices for a variation point; and (3) invariants, fixed elements that are not configurable in the domain. Although variability could also be modelled by

conventional approaches such as extended UML models, the state-of-the-practice of SPL shows that feature modelling is largely used due to its representational power concerned with the aforementioned elements.

However, there are many notations in the technical literature for feature modelling [Reibisch et al., 02] [Cechticky et al., 04] [Czarnecki et al., 04] [Gomaa, 04] [Czarnecki et al., 05], all of them based on the original notation designed for the Feature-Oriented Domain Analysis (FODA) method [Kang et al., 90]. The literature also has some notations designed for specific approaches, such as FeatuRSEB [Griss et al., 98], FORM [Kang et al., 02] [Lee et al., 02], and Odyssey-FEX [Blois et al., 06]. Amongst these notations, some concepts have the same semantics, regardless of the provided graphical representations or different nomenclatures, forming a set of basic concepts from which each notation could be extended, including: optionality, variability, and dependency relations. Thus, some notations present a richer structure of components to support feature modelling, as it can be seen in the Odyssey-FEX notation that offers a larger set of relationships and a more comprehensive taxonomy of features when compared to other notations [Teixeira et al., 09], including its own categorization of features, reflecting the different stages of the software product lifecycle. In this notation, a feature can be categorized as a *functional or conceptual feature*, categories of domain features, and *entity feature*, for domain analysis. Also, a feature can be categorized as an *operational environment*, *domain technology*, or *implementation technique* for domain design. Domain features are related to the core domain functionalities and concepts. Entity features are the model actors. Operational environment features represent attributes of an environment that a domain application can use and operate. Domain technology features represent technologies used to model or implement a specific domain requirement. Finally, implementation techniques features represent technologies used to implement other features.

In addition to the *alternative* relationship, which is used among variation points and their variants, features of an Odyssey-FEX model can be connected by UML relationships, such as association, aggregation, and composition. The Odyssey-FEX notation also supports relationships of dependencies between features and mutual exclusion.

3.2 Discrepant Cases

Discrepant Case (DC) could be defined as a generic situation that can be found in software artefacts configuring a discrepancy, i.e., where there could be a defect after the inspection meeting [de Mello et al., 10]. By analysing components and examples from FODA notation [Kang et al., 90] and other notations previously mentioned, we tried to extract what could be considered discrepancies basically related to consistency, clarity, correctness, and completeness of a FM in comparison to its corresponding domain textual description. Thus, after the study of various FM notations and analysis of models from DE projects, a first set of 48 discrepant cases was identified. It is important to point that these discrepant cases are not meant to be the unique possibilities of semantic defects that could be related to inspecting FMs.

The 48 DC identified for FMs were organized into the following groups:

A. *Feature description:* this category verifies the clarity of a feature description. Five DC were related to this individual analysis of each feature. This category

also checks if the feature is considered to be inside the scope of the domain according to the domain textual description (base document);

B. *Feature Category:* this category checks the proper classification of a feature into one of the seven categories established by the Odyssey-FEX notation [Blois et al., 06];
C. *Optional/Mandatory features:* this category checks if the optionality property of each feature was correctly applied. Three DCs were defined, as related to the classification of a feature as optional or mandatory, considering the whole domain. This category also checks if the optionality/mandatory property of a feature could actually be defined from the base document;
D. *Variability and Association Relationships:* this category checks if the relationships specified in the base document were properly represented in the model. Sixteen DCs were defined, as related to the establishment of relationships between features. The classification of a feature into its variability property is evaluated by the analysis of the relationship between each variation point feature and its alternative configuration features. This category also evaluates the relationships of *aggregation, composition, generalization* and *'implemented by'*, when applied by the adopted notation;
E. *Dependency and mutually exclusive relationships:* this category includes six DCs related to the possibility of the joint features selection (dependency) or the unfeasibility of two or more features being selected together (mutual exclusion) for product development;
F. *Rationale and Composition Rules:* five DCs related to the textual description of logical definitions and model composition rules;
G. *Overview of the model:* six DCs related to the clarity and consistency of the model as a whole. The features are evaluated to verify their pertinence to the domain scope and to check their adequacy for the abstraction level used to understand the model and future application in the domain implementation.

Each discrepant case, which shall be identified as defect or false positive in the inspection meeting, is related to one of the following defect categories, according to the classification adopted by Shull [Shull et al., 00] for defects in software requirements and extended for UML models [Travassos et al., 01]: *omission, incorrect fact, inconsistency, ambiguity,* and *extraneous information*. Thus, this classification was adopted to guide the identification and categorization of discrepant cases in feature models, as shown in Table 5.

4 FMCheck

Based on the discrepant cases mentioned in the last section and based on the experience obtained with the development of ActCheck – an inspection technique for UML activity diagrams [de Mello et al., 11], a feature model checklist (FMCheck) was developed. FMCheck is a checklist-based inspection technique to support individual inspectors to detect defects in feature models designed under notations based on FODA, also including particular items related to Odyssey-FEX notation. This technique was developed for software processes based on individual inspections. The inspector does not need to have previous domain knowledge to apply FMCheck, since some valid textual description, such as requirements specification or domain

specification, is defined as a pre-requisite (base document) to be used as basis for comparison during the application of the technique.

Name	Description
Omission	Some information from the domain was not properly included in the feature model.
Incorrect Fact	Some information or behaviour from the feature model contradicts its domain specification.
Inconsistency	Some feature model element is not consistent with another element from the same feature model.
Ambiguity	Some Information from the feature model is not clear, allowing multiple interpretations for the specified domain.
Extraneous Information	Some Information in the feature model is outside the domain scope.

Table 5: Defect Types, adapted from [Travassos et al., 01].

The application of FMCheck consists of three activities. First, the domain analyst or the domain designer should fill a *model characterization questionnaire*. This questionnaire aims to provide the basis for the checklist configuration, discarding evaluation items from FMCheck that do not need to be applied in the specific verification scenario. The main goal is to avoid the extra effort of checking unnecessary verification items, as FMCheck provides items that could be applied in a broader context, addressing multiple notations. The questions deal with several kinds of information, such as the DE stage in which the model was designed (Domain Analysis or Domain Design), the feature model notation chosen to represent the model, the modelling capabilities provided by the applied notation, and so on. It is important to point that the questionnaire should be filled based on what is provided by the adopted FM notation, and not only on the characteristics presented in the model to be inspected.

The second activity consists of the *checklist configuration*, to be done by the inspection moderator, supported by a traceability table relating each answer from the model characterization questionnaire to specific evaluation items from the checklist (see Section 4.1). Finally, the third activity consists of conducting one or more individual inspections, producing one or more discrepancy reports, describing each discrepancy, its defect category and location.

4.1 The Checklist

In a checklist, an excessive amount of divisions can lead inspectors to massive rework, as already observed in the development of ActCheck [de Mello et al., 11]. Thus, as the organization of an inspection technique should consider its practical application, the 48 DCs originally divided into seven groups were analysed and regrouped, allowing the construction of the checklist. Then, the verification items of FMCheck were split into three verification groups: *individual verification of each feature*; *verification of relationships between features*; and *verification of composition rules*. As a result the first version of FMCheck, consisting of 34 verification items

organized into the mentioned verification groups, was established. For each verification item, there are three possible answers: "Yes", "No" or "N.A." (*not applicable*). The "N.A." option means that the item is not applied to the inspected model, although it could be when adopting the same feature modelling notation for other models.

4.1.1 Individual Verification of Features

The 13 verification items from this group (see Table 6) aim to ensure that each feature has a correct, clear and objective description. They also check if a feature belongs to the domain. These evaluation items were designed based on the DC categories (A, B, C, and G) described in Section 3.2.

#	Verification Items
1	Are all the features clearly and correctly described?
2	Is the described optionality of each feature (optional/ mandatory classification) in accordance with the domain specification?
3	Is it possible to identify the feature category by its description on the domain?
4	Are the features representing conceptual aspects of the domain properly classified as *Conceptual Features*?
5	Are the features representing functional aspects of the domain properly classified as *Functional Features*?
6	Are the features representing a real entity (actor) of the domain properly classified as *Entity Features*?
7	Are the features representing attributes of an environment related to the domain properly classified as *Operational Environment* features?
8	Are the features representing some technology used to model or implement the domain properly classified as *Domain Technology* features?
9	Are the features representing some technology used to implement other domain features properly classified as *Implementation* features?
10	Are the features that do not have a concrete relation with the domain, but help in its understanding, represented as *organizational features*?
11	Is there some feature in the model that, although correct, is out of the domain scope?
12	Are there different features in the model that represent the same domain concept?
13	Is there any domain concept that has been omitted from the model?

Table 6: Verification items for each model feature.

4.1.2 Verification of Relationships between Features

The 16 verification items from this group guide the inspector in the verification of the relationships between features (see Table 7). These items aim to verify how the representation of the relations between features renders the model understandable, deployable, and compliant with the domain. The DC categories (D and F) mentioned in Section 3.2 were the basis for the definition of these items.

4.1.3 Verification of Composition Rules

The five items from this group (see Table 8), guide the inspector in checking the clarity, completeness, correctness, relevance, and consistency of the model composition rules as established in FODA [Kang et. 90]. This set of items was based on the F category of discrepant cases, as described in Section 3.2.

#	Verification Items
14	Are the variabilities of the domain adequately represented as groups of alternatives (variation point and its variants)?
15	Are the cardinalities of the variation points correct?
16	Are the variation points clearly described, reflecting the meaning of their variants?
17	Are there two or more features having a relationship in the model without defining this relationship in the domain?
18	Is there some relationship described in the domain that has not been informed in the model?
19	Is the established hierarchy between each feature compliant with the domain?
20	Is there some feature in the model that has been incorrectly classified as a *generalization* of another feature?
21	Are the features identified in the model as *implemented by* another feature present this relationship in the domain?
22	Are the *aggregation* and *composition* relationships between domain features in the model consistent with the reality of this domain?
23	Is there any *dependency* or *mutually exclusive* relation between features that is not applied to the described domain?
24	Is there any *dependency* or *mutually exclusive* relation between features that is not represented in the model?
25	Is there any feature in the model contradicting other features?
26	Does the *root* feature help to understand the meaning of the domain?
27	From a general perspective, is it possible to understand the domain from the features represented in the model?
28	Does the model describe the domain in an appropriate level of detail to be understood from the intended perspective?
29	Does the model have the sufficient features to guide the domain implementation?

Table 7: Verification of the relationships between the features from a model.

#	Verification Items
30	Are all the composition rules clearly and objectively described, being in compliance with the domain description?
31	Is there any composition rule that contradicts another one in the same model?
32	Is there any composition rule that is not applied to this domain, although it is correct?
33	Are all domain composition rules adequately represented in the model?
34	Does the model present sufficient composition rules to guide its implementation?

Table 8: Verification items for composition rules between model features.

5 Evaluation of FMCheck

The Evaluation of FMCheck feasibility was made of two activities: a proof of concept and a first experimental study. After the development of the first version of FMCheck, two developers (one Doctoral student – P1, more experienced and one Master student – P2, less experienced) from the Reuse Group at COPPE/UFRJ were invited to apply the technique in a specific domain (i.e., mobile devices). FMCheck was presented to these participants, but they did not have access to the feature models before their inspection. Each participant received an email containing the necessary artefacts for the inspection, including a spreadsheet containing the checklist. As the FM of this domain was modelled using the Odyssey-FEX notation, the checklist for these inspections was set-up to contain all the 34 items proposed by FMCheck.

Table 9 shows the results for each participant. Effectiveness was defined as the ratio between the number of defects found by an inspector and the total amount of defects, whereas efficiency indicated the average time (in minutes) the inspector needed to detect a single defect. Comparing the results of the two inspections, it was observed that the most experienced inspector (P1) found more defects than the less experienced participant (P2), although P1 needed much more time to accomplish the same task. Moreover, one can see the low incidence of false positives and the high incidence of identical defects found by the two participants.

Part.	#Defects	#Repeated Defects	#False Positives	Time (min.)	Effectiveness	Efficiency
P1	12	6	2	80	92.31%	6,67
P2	7		1	12	53.85%	1,71

Table 9: Summarized Proof of Concept results.

Positive comments on the completeness of FMCheck and its applicability for the intended activity were made by the two participants concerning the support provided by the inspection technique. Also, P1 mentioned a possible negative influence factor on performance: the limited technical resources of the environment used to perform the whole activity (both participants used a desktop with a single and small screen)

may have made it difficult for both the completion of the spreadsheet and the artefacts analysis. Despite this, as the focus of this pilot study was to observe the feasibility of using the technique, issues related to a possible automated or semi-automated support were left out for future investigations.

5.1 The *quasi*-Experiment

The positive results observed in the proof of concept led to the submission of the first version of FMCheck to an experimental study aimed at increasing the capacity of observation of its feasibility. Thus, a *quasi*-experiment was planned to be analysed under two perspectives, quantitative and qualitative. The following subsections present a summary of the study planning, its results, and the analysis performed.

5.1.1 Specific Goals

Based on the GQM template [Basili et al., 94], the goal of this study was defined as follows:

Analyse: the conducting of feature model inspections by using *ad-hoc* techniques and FMCheck

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: undergraduate and graduate students from a Software Reuse course at PESC-COPPE/UFRJ (representing as much as possible software developers) inspecting feature models in four different application domains (i.e., mobile devices, hospitality, context-aware mobile applications and library).

5.1.2 Questions and Metrics

- *Question:* How much time was dedicated to the inspections?
- *Metrics:* time dedicated to the inspection (in minutes), and efficiency of each inspection (as defined in the previous study).
- *Question:* Which inspection technique (FMCheck or ad-hoc) allows the inspectors to detect more defects?
- *Metrics:* number of defects detected, effectiveness of the inspection (as defined in the previous study).

5.1.3 Hypotheses

$H_0$1: There is no difference between the efficiency of feature model inspections conducted with FMCheck and with *ad-hoc* inspections.

H_A1: The efficiency of feature model inspections conducted with FMCheck is greater than the efficiency of *ad-hoc* ones.

$H_0$2: There is no difference between the effectiveness of feature model inspections conducted with FMCheck and *ad-hoc* inspections.

H_A2: The effectiveness of feature model inspections conducted with FMCheck is greater than that of *ad-hoc* ones.

5.1.4 Variables

- *Independent variables:* application domains textually described and represented through feature models using the Odyssey-FEX notation, participant's experience in inspections, participant's previous knowledge of the domains used in the study.
- *Dependent variables:* Amount of defects and false positives, time spent in performing the inspection, efficiency, and effectiveness.

5.1.5 Experimental Design

The study participants consisted of 14 students (four undergraduate students and 10 graduate students) from a Software Reuse course at COPPE/UFRJ, who signed a consent form and filled in a characterization form. These participants were organized into three groups (A, B and C), based on the following criteria: (i) academic and industrial experience with Software Engineering; (ii) academic degree (undergraduate, master, doctoral student); and (iii) previous experience with software inspections in general and experience with inspection of feature models. In this study, considering the sample, only the first criterion was needed to group the participants. Thus, Group A had four participants with greater experience in the industry, group B had six participants with some experience in the industry and group C had four participants with only academic experience.

For the first round, the participants were trained in software inspection and domain description through feature models, to prepare them for the execution of two *ad-hoc* inspections. After that, in the second round, the participants were first trained in the application of FMCheck; then, each participant performed inspections of two other artefacts (that had not been inspected by them in the first round) applying FMCheck. In the first round, due to the absence of the training session, two participants were withdrawn from the study. In the second round, three participants dropped out the study. Thus, only nine participants participated in both rounds.

5.1.6 Instrumentation

For the inspections, the domain specification and feature model artefacts from four different domains were selected. Figure 1 shows an excerpt from the feature model concerning the hospitality domain.

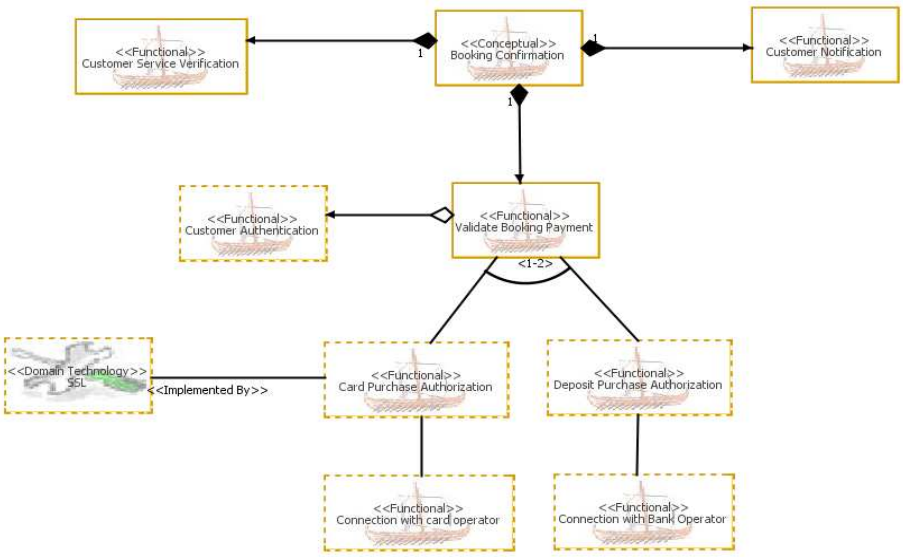

Figure 1: Excerpt (translated) of the Hospitality Domain (Odyssey-FEX notation).

Besides hospitality, the other three domains selected were: mobile devices, context aware mobile applications, and library. Every round had each participant inspecting two distinct domains, which were assigned according to: (i) the complexity of each feature model, (ii) the sample size and its groups (A, B, or C).

In order to establish the complexity of each feature model, we did a comparative analysis between the four models, applying the following criteria: number of features, maximum depth between features, and amount of variability. Thus, two domains were considered simplest – mobile devices and library (S01 and S02, respectively), with the other two models considered more complex, i.e., context-aware mobile applications and hospitality (C01 and C02, respectively). The final distribution of the models among the participants over both rounds can be seen in Table 10. Also, a follow-up questionnaire sent by email to all participants collected their impressions about FMCheck, including possible contributions to streamline it.

5.1.7 Analysis Mechanism

To undertake the *quasi*-experiment, the following analysis mechanisms for the collected data were adopted:

- Comparison between results from *ad-hoc* and FMCheck inspections.
- Comparison between the performance of each participant in both rounds.
- Calculation of defects' variance and standard deviation.
- Calculation of the time spent with the inspections.
- Elimination of outliers and verification of data normality (*Shapiro-Wilk*) and homoscedasticity (*Levene*).
- Application of a nonparametric test (*Wilcoxon*) or a parametric test (*Student's t*), according to each case.

Group	Participant	Round 1 (*ad-hoc*)	Round 2 (FMCheck)
A	P1	C01 C02	S01 S02
	P2	S01 S02	C01 C02
	P3	S01 C01	S02 C02
	P4	S02 C02	S01 C01
B	P5	S01 C02	S02 C01
	P6	S02 C01	S01 C02
	P7	S01 S02	C01 C02
	P8	C01 C02	S01 S02
	P9	S01 C01	S02 C02
	P10	S02 C02	S01 C01
C	P11	S01 C02	S02 C01
	P12	S02 C01	S01 C02
	P13	S01 S02	C01 C02
	P14	C01 C02	S01 S02

Table 10: Planned distribution of inspectors and artefacts for the two rounds.

5.1.8 Execution

The study was executed in April/ May 2012, starting with the completion of the consent form and the characterization form by 14 participants. Then, participants were trained in feature modelling and in the Odyssey-FEX notation. Also, an introductory training (one hour) in software inspection was done, including the guidelines for the execution of the first round. Each participant received an email containing an inspection package, and 12 participants answered until the given deadline, reporting the defects detected in each inspection.

Then, the participants were trained (one hour) in FMCheck, by presenting and explaining each evaluation item and examples of defects that could be detected with these items. In the second round, as mentioned earlier, only nine participants maintained their participation in the study, answering to the forwarded packages.

Table 11 summarizes the results obtained in the first round, and Table 12 summarizes the results obtained in the second round. To calculate the efficiency (ratio between defects detected and the total of defects), the total of defects corresponds to the sum of distinct defects detected in a FM in both rounds. The sum of defects for each FM is shown in Table 13.

Group	Part.	Domain	Time (min.)	# Defects	Efficiency	Effectiveness
A	P2	S01	31	3	10.33	23.08%
		S02	19	3	6.33	6.98%
	P3	C01	50	8	6.25	18.18%
		S01	50	6	8.33	46.15%
	P4	C02	40	2	20.00	6.90%
		S02	32	3	10.67	6.98%
B	P5	C02	138	15	9.20	51.72%
		S01	43	9	4.78	69.23%
	P6	C01	44	11	4.00	25.00%
		S02	50	6	8.33	13.95%
	P7	S01	34	6	5.67	46.15%
		S02	41	6	6.83	13.95%
	P8	C01	32	4	8.00	9.09%
		C02	43	2	21.50	6.90%
	P9	C01	25	5	5.00	11.36%
		S01	30	3	10.00	23.08%
C	P11	C02	94	16	5.88	55.17%
		S01	65	3	21.67	23.08%
	P12	C01	50	8	6.25	18.18%
		S02	35	14	2.50	32.56%
	P13	S01	30	1	30.00	7.69%
		S02	30	4	7.50	9.30%
	P14	C01	65	6	10.83	13.64%
		C02	45	6	7.50	20.69%

Table 11: Results for the first round: Ad-hoc inspections.

5.1.9 Quantitative Analysis

The results obtained were analysed from several perspectives, including time, discrepancies, defects, efficiency, and effectiveness. In addition, differences in these perspectives among groups and their inspected artefacts in each distribution were also observed. Statistical tests were performed with the support of the JMP 4.0 tool (http://www.jmp.com/), using α=0.05. All distributions were normally distributed after the removal of 14 outliers (11 *ad-hoc* inspections and three inspections applying FMCheck). Table 14 shows the behaviour of the distributions and test results, considering 13 observations for *ad-hoc* samples and 15 for FMCheck samples.

Group	Part.	Domain	Time (min.)	# Defects	Efficiency	Effectiveness
A	P2	C01	83	25	6.92	27.27%
		C02	78	21	9.75	27.59%
	P3	C02	60	12	6.00	34.48%
		S02	60	19	4.00	34.88%
B	P5	C01	142	32	6.17	52.27%
		S02	138	31	4.76	67.44%
	P6	C02	55	12	6.11	31.03%
		S01	36	6	9.00	30.77%
	P7	C01	63	20	4.50	31.82%
		C02	54	16	4.15	44.83%
	P8	S01	20	2	10.00	15.38%
		S02	45	5	22.50	4.65%
C	P11	C01	104	21	6.12	38.64%
		S02	90	23	5.63	37.21%
	P12	C02	85	16	7.73	37.93%
		S01	50	11	7.14	53.85%
	P13	C01	60	7	10.00	13.64%
		C02	60	9	8.57	24.14%

Table 12: Results for the second round: inspections applying FMCheck.

Domain	C01	C02	S01	S02
#Defects	44	29	13	43

Table 13: Total of distinct defects detected in each domain.

Distribution	**Normality(*Shapiro-Wilk, p-value*)**		**Homoscedasticity (*Levene, p-value*)**
	Ad-hoc	**FMCheck**	
Time	0.3966	0.6756	0.2078
Discrepancies	0.1225	0.7162	0.0021
Defects	0.0606	0.9247	0.0047
Efficiency	0.2683	0.3128	0.7176
Effectiveness	0.3056	0.7809	0.1833

Table 14: Normality and homoscedasticity of the distributions.

Only in Time, Efficiency and Effectiveness distributions homoscedasticity was observed. Over the normal and homoscedastic distributions, the Student's t test was applied. Distributions that did not show homoscedasticity were submitted to the Wilcoxon nonparametric test. Table 15 summarizes the results of both tests, in which it can be seen that it was not possible to reject the null hypothesis regarding efficiency. However, the null hypothesis regarding effectiveness was refuted. In fact, according to the results, the time required to undertake inspections with FMCheck is greater than with *ad-hoc*. Even so, it was observed that, although having similar efficiency, FMCheck inspections identified 51.3% more defects than *ad-hoc* ones,

which can be seen through the boxplots shown in Figure 2, showing that the first quartile of effectiveness with FMCheck are up to the third quartile of *ad-hoc* effectiveness, suggesting a significant difference between the samples. Also, Figure 2 shows that the distribution of time spent to detect a defect (efficiency) with FMCheck inspections is more diverse than *ad-hoc*.

Distribution	Analysis Result	Statistic test(*p*-value)	
		t-test	Wilcoxon
Time	ad-hoc <FMCheck	0.0008	-
Discrepancies	ad-hoc <FMCheck	-	0.0118
Defects	ad-hoc <FMCheck	-	0.0018
Efficiency	ad-hoc = FMCheck	0.2229	-
Effectiveness	ad-hoc <FMCheck	0.0001	-

Table 15: Statistical test for each distribution.

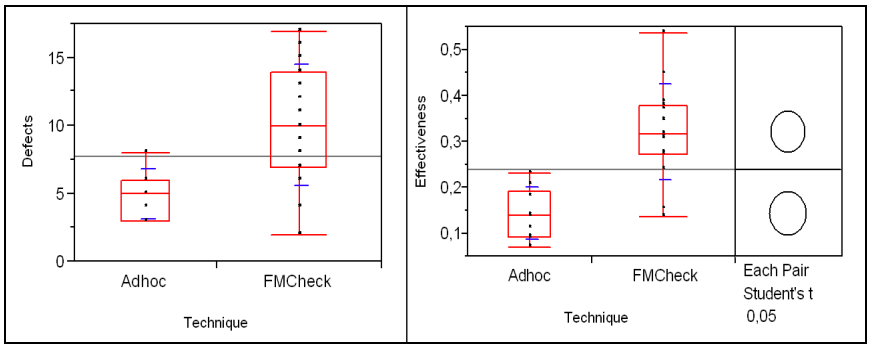

Figure 2: Distribution of number of defects for each sample and distribution of Effectiveness for each distribution and corresponding Student's t test.

Aiming to better understand the behaviour seen in both samples, the performance of each group (A, B and C) was analysed individually. The individual analysis showed that all three groups presented a significant increase in their effectiveness when applying FMCheck ($\alpha = 0.05$). The efficiency analysis of each group also produced the same conclusion from the sample as a whole, i.e., the groups performed inspections with FMCheck as efficiently as when undertaking *ad-hoc* inspections.

Statistical tests were also applied to observe the influence of each inspected domain (S01, S02, C01 and C02) over the sample as a whole. In this sense, it was seen that none of the four domains represented any confounding factor for the conclusion on effectiveness. Also, FMCheck inspections had a more homogeneous behaviour (less variation) in effectiveness than *ad-hoc* inspections for all the domains. Looking at the individual performance of the nine participants who acted in both rounds, it was seen that seven of them had an increase in their effectiveness by applying FMCheck, and six of them (i.e., two thirds of the participants in both rounds) showed greater than 40% improvement of their effectiveness.

In total, the inspections with FMCheck were able to detect 118 distinct defects, while *ad-hoc* inspections detected only 76 defects. It is important to emphasize that this advantage for FMCheck was observed with a sample that was 25% lower. It is also important to point out that FMCheck detected 53 distinct defects not detected by *ad-hoc* inspections, while *ad-hoc* inspections were able to detect only 11 distinct defects that were not captured by FMCheck (as shown in Figure 3). However, it was found that FMCheck verification items covered 10 out of 11 defects reported only in *ad-hoc* inspections, suggesting a large coverage of the checklist for the scope of this study. As regards to the category of defects, it was observed that FMCheck contributed mainly to detect more omissions, incorrect facts, and ambiguities.

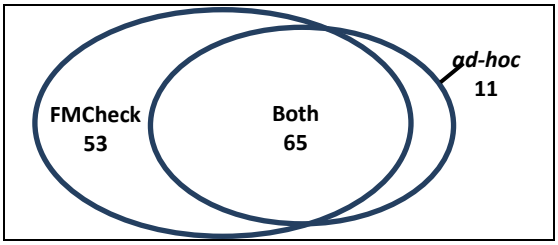

Figure 3: Number of distinct defects detected for both inspection rounds and distinct defects detected in each single round.

5.1.10 Qualitative Analysis

All the nine subjects that took part in the second round were invited by email to answer a follow-up questionnaire on the study, but only six participants replied. We found that the participants agreed (partially or totally) that the training sections helped them to conduct inspections with FMCheck, although, in their comments, some participants presented suggestions to improve future training sections and the material shared with the inspectors. In fact, due to time limitation of the subjects we believe that the training has been prejudiced. In addition, we are looking for more systematic ways to share packages of studies and to improve them, including video lessons as a training alternative that could be retrieved anytime by the subjects.

As regards to the technique, answers show that all respondents agreed (partially or totally) that the FMCheck verification items are well-described and the checklist is useful to detect defects. However, some disagreed that their time was better spent using FMCheck. Still, both previous observations are consistent with effectiveness and efficiency results. Finally, participants would use FMCheck in future inspections.

5.1.11 Threats to Validity

As threats to validity of this *quasi*-experiment, we should point the small number of subjects and the limited number of inspected domains. For instance, the statistical analysis of each group was limited. Moreover, the small number of participants also limited the combination of groups, which may cause some bias. However, it is worth noting that no participant inspected the same domain more than once. The absence of a prior list of known defects can also be considered as a threat to validity, which

directly affects the calculation of the coverage of inspections. The defects considered as 'known defects' were only those detected during this *quasi*-experiment.

Internally, we emphasize the bias regarding the fact that FMCheck was evaluated by the same group that developed it, even based on experimental practices. It is also important to note that this *in vitro* study was performed asynchronously, without the control of environment variables. Thus, the resources used by the participants to conduct their inspections (screen size, printed artefacts) may have positively or negatively influenced the results. We also emphasize the external threats to validity concerned with the population defined by convenience and non-random sampling, which is typical in *quasi*-experiments [de Mello and Travassos, 13]. Also, one can see that, in order to conduct the second round, applying FMCheck, each inspector may have brought some learning bias from the first round (*ad-hoc* inspections).

6 Conclusions

This paper showed, through the results of a *quasi*-systematic review, that there are few approaches to support the inspection of SPL, suggesting a set of alternatives to improve this scenario. Based on the findings, we proposed FMCheck, a checklist-based inspection technique designed to support inspectors in detecting defects in feature models. FMCheck was first presented in [de Mello et al., 12]. With FMCheck, we intend to contribute to the quality assurance in Domain Engineering, helping to prevent the dissemination of domain defects until product development. The inspection technique presented was subjected to a proof of concept and a quasi-experiment, which suggested its feasibility by showing superior effectiveness and roughly equivalent efficiency when compared to *ad-hoc* inspections.

7 Future Work

In the near future, we intend to evolve FMCheck based on the findings of the experimental study, re-applying this study to reinforce our conclusions. To avoid external threats to validity, we intend to conduct a large-scale experiment based on the approach for systematic population establishment and randomly sampling on SE quantitative studies using social networks as source of recruitment, an ongoing research of two authors of this paper [de Mello and Travassos, 13b].

We also intend to evolve the Odyssey environment [Werner et al., 99] by integrating inspection activities based on FMCheck. Other future works are related to other opportunities to improve the whole SPL inspection context, including better support for verifying the textual domain description, which can be served by an extension of the perspective-based reading (PBR) inspection technique [Shull et al., 00], including the new perspective of the domain analyst.

Acknowledgements

We would like to thank Breno França and Karen Nakazato for their support to the systematic review, and the students of the 2011 Special Topics in Software Engineering IV course, and the students of the 2012 Software Reuse course, both at

PESC/COPPE/UFRJ, especially the collaboration of Marcelo Palmieri. We would also like to thank researcher Jobson Massollar for his contributions. Professors Werner and Travassos are CNPq research scholars and Eldânae Teixeira and Marcelo Schots are scholarship grantees from CNPq.

References

[Arango and Prieto-Díaz, 91] Arango, G, Prieto-Díaz, R., "Part 1: Introduction and Overview-Domain Analysis Concepts and Research Directions". Domain Analysis and Software Systems Modelling. IEEE Computer Society Press, 1991.

[Atkinson et al., 02] Atkinson, C. et al. "Component-based product line engineering with UML". Boston, Addison-Wesley Longman Publishing Co., Inc., 2002.

[Barcelos and Travassos, 06] Barcelos, R. F.; Travassos, G. H. "Uma abordagem para inspeção de documentos arquiteturais baseada em checklist". Simpósio Brasileiro de Qualidade de Software (SBQS), 2006, Vila Velha, Brazil, 2006.

[Basili et al., 94] Basili, V., Caldiera, G., Rombach, H., 1994, "Goal Question Metric Paradigm", Encyclopaedia of Software Engineering, v. 1, John Wiley & Sons, pp. 528-532.

[Benavides et al., 10] Benavides, D., Segura, S., Ruiz-Cortés, A. "Automated Analysis of Feature Models 20 Years Later: A Literature Review". Inf. Syst., vl. 35(6), pp. 615-636, 2010.

[Biolchini et al., 05] J. Biolchini, P. G. Mian, A. C. C. Natali, G. H. Travassos, "Systematic Review in Software Engineering". Technical Report RT-ES 679/5, PESC-COPPE/UFRJ, 2005.

[Blois et al., 06] Blois, A. P. T. B., Oliveira, R. F., Maia, N., Werner, C., and Becker, K., "Variability modeling in a component-based Domain Engineering process". In: Reuse of Off-the-Shelf Components, Springer Berlin Heidelberg (2006), 395-398.

[Cechticky et al., 04] Cechticky, V., Pasetti, A., Rohlik, O., et al., "XML-based feature modeling". In: Software Reuse: Methods, Techniques and Tools, 8th International Conference on Software Reuse (ICSR), Proceedings, v. 3107, pp. 101–114, Madrid, Spain, July, 2004.

[Czarnecki et al., 04] Czarnecki, K., Helsen, S., Eisenecker, U., "Staged Configuration using feature models". In: Proceedings of the 3rd International Software Product Line Conference, SPLC 2004, v. 3154, pp. 266-283, Boston, MA, USA, August 30-September 2, 2004.

[Czarnecki et al., 05] Czarnecki, K., Helsen, S., Eisenecker, U. W., "Formalizing cardinality based feature models and their specialization", Software Process: Improvement and Practice, v. 10, n. 1 (March), pp. 7-29, 2005.

[de Mello et al., 10] de Mello, R. M., Pereira, W. M., Travassos, G. H. "Activity Diagram Inspection on Requirements Specification". XXIV Simpósio Brasileiro de Engenharia de Software, 2010.

[de Mello et al., 11] de Mello, R. M., Massollar, J. L., Travassos. G. H., "Técnica de Inspeção Baseada em Checklist para Identificação de Defeitos em Diagramas de Atividades". XX Simpósio Brasileiro de Qualidade de Software, 2011.

[de Mello et al., 12] de Mello, R. M., Teixeira, E. N., Schots, M., Werner, C. M., & Travassos, G. H. "Checklist-based Inspection Technique for Feature Models Review". In 6th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), pp. 140-149, 2012.

[de Mello and Travassos, 13] 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.

[de Mello and Travassos, 13] de Mello, R. M., & Travassos, G. H. "Would Sociable Software Engineers Observe Better?". In Proceedings of the 7th ESEM, 2013.

[Denger and Kolb, 06] Denger, C., Kolb, R. "Testing and inspecting reusable product line components: First empirical results". Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering, pp. 184-193, 2006.

[Fagan, 76] Fagan, M. E., "Design and Code inspections to reduce errors in program development". IBM Systems Journal 15 (3): pp. 182–211, 1976.

[Gomaa, 04] Gomaa, H., "Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures", Addison-Wesley, 2004.

[Gomaa and Shin, 07] Gomaa, H. A., Shin, M. E. B., "Automated Software Product Line Engineering and Product Derivation". Proceedings of the 40th Hawaii International Conference on System Sciences, pp. 285a, 2007.

[Griss et al., 98] Griss, M. L., Favaro, J., D'Alessandro, M., "Integrating feature modelling with the RSEB". In: Proceedings of the 5th International Conference on Software Reuse (ICSR), pp. 76-85 Victoria, British Columbia, Canada, 1998.

[Gurp et al., 01] Gurp, J., Bosch, J., and Svahnberg, M., "On the Notion of Variability in Software Product Lines". In: Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2001), 45-55.

[IEEE, 08] IEEE. "STD 1028-2008: IEEE Standard for Software Reviews and Audit", 2008.

[Jones, 00] Jones, C. "Software assessments, benchmarks, and best practices". Addison-Wesley, Reading, MA, USA Chung-Horng Lung, Joseph E. Urban, 2000.

[Kang et al., 90] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., Peterson, A. S. "Feature-Oriented Domain Analysis (FODA) Feasibility Study". Technical Report CMU/SEI-90-TR-21/ ESD-90-TR-222, 1990.

[Kang et al., 02] Kang, K. C., Lee, J., Donohoe, P., "Feature-Oriented Product Line Engineering", IEEE Software, v. 9, n. 4 (July/August 2002), pp 58-65.

[Kim et al., 08] Kim, K. Kim, H., Kim, S., Chang, G. "A case study on SW product line architecture evaluation: experience in the consumer electronics domain". 3rd International Conference on Software Engineering Advances, pp. 192-197, 2008.

[Lee et al., 02] Lee, K., Kang, K. C., Lee, J., "Concepts and Guidelines of Feature Modelling for Product Line Software Engineering". In: Software Reuse: Methods, Techniques, and Tools: 7th International Conference on Software Reuse (ICSR), pp. 62-77, Austin, USA, April, 2002.

[Lung et al., 97] Lung, C. H., Bot, S., Kalaichelvan, K., Kazman, R. "An Approach to Software Architecture Analysis for Evolution and Reusability". Proc. of CASCON, 1997.

[McGregor, 01] McGregor, J. D. "Testing a Software Product Line. Carnegie Mellon, Software Engineering Institute", ESC-TR-2001-022, 2001.

[Northrop, 02] Northrop, L. M. "SEI's Software Product Line Tenets". IEEE Software July/ August 2002. IEEE Computer Society Press, 2002.

[Ortega et al., 07] Ortega, M., Grimán, A., Pérez, M., Mendoza, L. E., "Reuse strategy based on quality certification of reusable components". IEEE International Conference on Information Reuse and Integration, pp. 140-145, 2007.

[Riebisch et al., 02] Riebisch, M., Böllert, K., Streitferdt, D., et al., "Extending Feature Diagrams with UML Multiplicities". In: Proceedings of 6th Conference on Integrated Design & Process Technology, pp. 1-7, Pasadena, California, USA, June, 2002

[Schimid and Widen, 00] Schmid K., Widen, T. "Customizing the PuLSE™ Product Line Approach to the Demands of an Organization". Proceedings of the 7th European Workshop on Software Process Technology, pp.221-238, 2000.

[Schimid and van der Linden, 07] Schmid, K., van der Linden, F. "Improving product line development with the families evaluation framework (FEF)". Proceedings of the 11th International Software Product Line Conference, Japan, pp. 15-16, 2007.

[Shull et al. 00] Shull, F., Rus I., Basili, V. "How Perspective-Based Reading can Improve Requirements Inspections", IEEE Computer, 33(7), 73-79, 2000.

[Shull and Seaman, 08] Shull, F., Seaman, C. "Inspecting the history of inspections: An example of evidence-based technology diffusion", IEEE Software, v. 25, n. 1, pp. 88-90, 2008.

[Teixeira et al., 09] Teixeira, E., Vasconcelos, A., Werner, C., "An Approach to Support a Flexible Feature Modelling". III Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), 2009, Natal, Brazil, pp. 81-94.

[Travassos et al., 99] Travassos, G. H., Shull, F., Fredericks, M, Basili, V. R. "Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality" Proceedings of the Intl. Conf. on OOPSLA, pp. 47-56, 1999.

[Travassos et al., 01] Travassos, G. H. In Rocha, A. R. C., Maldonado, J. C., Weber, K. C., "Qualidade de Software – Teoria e Prática". Prentice Hall, 2001.

[Travassos et al., 08] Travassos, G. H.; Santos, P. S. M.; Mian, P.; Dias Neto, A. C; Biolchini, J. An Environment to Support Large Scale Experimentation in Software Engineering. In: IEEE Intl. Conf. on Engineering of Complex Computing Systems. Belfast. pp. 193-202, 2008.

[Vasconcelos and Werner, 07] Vasconcelos, A. and Werner, C. "Architecture Recovery and Evaluation Aiming at Program Understanding and Reuse". LNCS 4880, pp. 72-89, 2007.

[Vasconcelos and Werner, 08] Vasconcelos A. and Werner, C. "Refining the Architecture Recovery Approach ArchMine by Incrementally Performing Evaluation Studies". XXII Simpósio Brasileiro de Engenharia de Software, pp. 172-187, 2008.

[Vasconcelos and Werner, 11] Vasconcelos, A. and Werner, C. "Evaluating reuse and program understanding in ArchMine architecture recovery approach". Inf. Sci., pp. 2761-2786, 2011.

[Werner et al., 99] Werner, C. M. L., Mattoso, M., Braga, R, "Odyssey: Infraestrutura de Reutilização Baseada em Modelos de Domínio". In: Seção de Ferramentas do XIII Simpósio Brasileiro de Engenharia de Software, pp. 17-20, 1999.

[Wong, 06] Wong, Y. K. "Modern Software Review – Techniques and Technologies". IRM Press, 2006.