Research Journal of Applied Sciences, Engineering and Technology 7(2): 275-281, 2014
DOI:10.19026/rjaset.7.251
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2014 Maxwell Scientific Publication Corp.
Submitted: March 29, 2013 Accepted: April 22, 2013 Published: January 10, 2014

Research Article

Using a Rule-based Method for Detecting Anomalies in Software Product Line

Abdelrahman Osman Elfaki, Sim Liew Fong, P. Vijayaprasad,
Md Gapar Md Johar and Murad Saadi Fadhil
Faculty of Information Science and Engineering, Management and Science Universiti, Malaysia

Abstract: This study proposes a rule based method for detecting anomalies in SPL. By anomalies we mean false-optional features and wrong cardinality. Software Product Line (SPL) is an emerging methodology for software products development. Successful software product is highly dependent on the validity of a SPL. Therefore, validation is a significant process within SPL. Anomalies are well known problems in SPL. Anomiles in SPL means dead feature, redundancy, wrong-cardinality and false-option features. In the literature, the problem of false-option features and wrong cardinality did not take the signs of attentions as a dead feature and redundancy problems. The maturity of the SPL can be enhanced by detecting and removing the false-option features. Wrong cardinality can cause problems in developing software application by preventing configuration of variants from their variation points. The contributions of this study are First Order Logic (FOL) rules for deducing false-option features and wrong-cardinality. Moreover, we provide a new classification of the wrong cardinality. As a result, all cases of false-option features and wrong variability in the domain-engineering process are defined. Finally, experiments are conducted to prove the scalability of the proposed method.

Keywords: Domain engineering, software product line, variability

INTRODUCTION

Software Product Line (SPL) has been proven to be an effective strategy to benefit from software reuse allowing many organizations to reduce development costs and duration, meanwhile increasing product quality (Bosch, 2002). SPL has two main processes. The first process is the domain-engineering that represents domain repository and it's responsible for preparing domain artifacts. The second process is the application engineering that aims to consume specific artifact concerning the desired application specification. Feature Model (FM) (Kang *et al*., 1990) and Orthogonal Variability Model (OVM) (Pohl *et al*., 2005) are the useful techniques for representing variability in the SPL. A particular product-line member is defined by a unique combination of features (if variability modeled using FM) or a unique combination of variants (if variability modeled using OVM). The set of all legal features or variants combinations defines the set of product line members.

Variability is the ability of a system to be efficiently extended, changed, customized, or configured to be used in a particular context (Svahnberg *et al*., 2005). According to Beuche *et al*. (2004) variability management proposals should be simple (easy to understand), universal, able to manage variability at all levels of abstraction and the introduction of a new variability expression should be as easy as possible. These conditions for successful variability modeling methods are applicable in our proposed method. Moreover, the proposed method can be used for validating SPL.

The principal objective of SPL is to configure a successful software product from the domain engineering process by managing SPL artifacts using variability modeling technique. Recently, validation of SPL has been discussed as an important issue concentrating on the maturity of SPL (Benavides *et al*., 2009, 2008; Eisenecke *et al*., 2012; Heymans *et al*., 2011). Validating SPL intends to ensure the correctness of artifacts in domain engineering and to produce error-free products including the possibility of providing explanations to the modeler so that errors can be detected and eliminated. Usually, a medium-size SPL contains thousands of features. Therefore validating SPL represents a challenge. The validation of SPL is a vital process and not feasible to be done manually.

The challenging of automated validation of SPL that has been mentioned in Batory *et al*. (2006), Benavides *et al*. (2009, 2008), Eisenecke *et al*. (2012), Heymans *et al*. (2011), Maßen and Lichter (2005) and Wang *et al*. (2007) is motivated our work in this study . The lack of a formal semantics and reasoning support of FM has hindered the development of validation methods for FM (Wang *et al*., 2007). Moreover, the

Corresponding Author: Abdelrahman Osman Elfaki, Faculty of Information Science and Engineering, Management and Science Universiti, Malaysia
This work is licensed under a Creative Commons Attribution 4.0 International License (URL: http://creativecommons.org/licenses/by/4.0/).

automated validation of SPL was already identified as a critical task in Batory *et al*. (2006), Kang *et al*. (2002) and Maßen and Lichter (2005).

In our previous work (Elfaki *et al*., 2008, 2009a, b, c), First Order Logic (FOL) rules are suggested to validate SPL. In addition, these FOL rules satisfied constraint dependency checking, optimization and explanation. Moreover, implementation examples are discussed. In Elfaki *et al*. (2009a, b, c) the scalability of the proposed method is illustrated. In this study, we complete our work by defining FOL rules for detecting anomalies in SPL, specifically in the domain engineering process. By anomalies we mean false-optional features and wrong cardinality.

In SPL, configuration has been defined as a process for producing software product that satisfying the constraint dependency rules. With huge number of software artifacts and constraint dependency rules, the configuration process could be a complicate process. As our main contribution of this study is validating domain-engineering direct without any need for configuration, then we can conclude that our approach minimizes the cost of the validation in SPL.

LITERATURE REVIEW

A knowledge-based product derivation process is suggested in Hotez and Krebs (2003) and Hotz *et al*. (2003). A knowledge-based product derivation process is a configuration model that includes three entities of knowledge base. The automatic selection provides a solution for complexity of product line variability. In contrast to the proposed method, the knowledge-based product derivation process does not provide explicit definition of variability notations or for the configuration process. In addition, knowledge-based product derivation process is not focused on validating variability.

Mannion (2002) was the first to connect propositional formulas to FM. Mannion's model did not concern cross-tree constraints (Require and Exclude constraints) and has not been used for supporting validation operations. Zhang *et al*. (2004) defined a meta-model of FM using Unified Modeling Language (UML) core package and took Mannion's proposal as foundation and suggested the use of an automated tool support. In Zhang *et al*. (2004), the proposed model does not deal with the anomalies in SPL. Batory (2005) proposed a coherent connection between FM, grammar and propositional formulas. Batory's study represented basic FM using context-free grammars plus propositional logic. Batory's proposal allows arbitrary propositional constraints to be defined among features and enables off-the-shelf satisfiable solvers to debug FM. Robak and Pieczynski (2003) described a system based on a feature diagram tree, annotated with weighted variant features in the basis of fuzzy logic for modeling variability. Robak did not describe how to validate the SPL. Sun *et al*. (2005) proposed first-order logic to model the FM and used Alloy Analyzer (the Alloy analyzer is a tool for analyzing models written in Alloy) to automate consistency checking in the configuration process. The proposal in Fan and Zhang (2006) does not deal with the validation operations. Fan and Zhang (2006) used description logic for reasoning in FM. The work in Sun *et al*. (2005) and Fan and Zhang (2006) did not mention other validation operations. Czarnecki and Antkiewicz (2005) proposed a general template-based approach for mapping FM. Czarnecki and Pietroszek (2006) used Object-Constraint Language (OCL) to validate constraint rules.

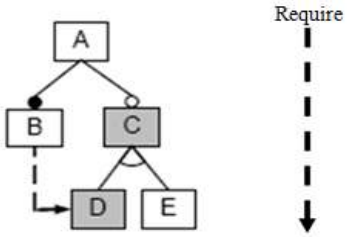

Fig. 1: Example of false-optional feature

Trinidad *et al*. (2006) defined a method to detect dead features based on finding all products and search for unused features. Trinidad *et al*. (2008a) extended this CSP technique to identify false-option features. The aiming of CSP searching is to find all solutions that satisfy the constraints. Finding all solutions is not practical with the huge-size of SPL. Our work is different because we detect false-option features in domain-engineering process.

FAMA framework (Trinidad *et al*., 2008b) defined a deductive operation for wrong cardinality. Our work is different because we classified wrong cardinality into four types and provide auto-support to detect each type.

OPERATIONS FOR DETECTING FALSE-OPTIONAL FEATURES AND WRONG CARDINALITY

In this section, how the proposed method can be used to define and provide auto-support for detecting false-optional features and wrong cardinality are illustrated. Prolog (Segura, 2008) is used for implementing the proposed operations.

False-optional feature detection: A false-optional feature is a feature included in any product but not assigned as a common feature, i.e., a common feature without a common label (Maßen and Lichter, 2005). Figure 1 illustrates an example of a false-optional feature. Feature 'B' is a common feature, which means 'B' must be included in any product. Feature 'B'

Table 1: Rules for detecting false-optional features

Definitions	
type (V_1, variant), type (V_2, variant), type (VP_1, variationpoint), type (VP_2, variationpoint) variants (VP_1, V_1), variants (VP_2, V_2), common (V_1, yes) and common (VP_1, yes).	
$\forall VP_2, V_1, V_2$: requires_v_v (V_1, V_2) $\wedge$common (VP_2, no) $\wedge$common (V_2, yes) $\Rightarrow$false_option (V_2)	(1)
$\forall VP_2, VP_1$: requires_vp_vp (VP_1, VP_2) $\wedge$common (VP_2, no) $\Rightarrow$false_option (VP_2)	(2)
$\forall VP_2, V_1$: requires_v_vp (V_1, VP_2) $\wedge$common (VP_2, no) $\Rightarrow$false_option (VP_2)	(3)
$\forall VP_2, V_2$: false_option (VP_2) $\wedge$common (V_2, yes) $\Rightarrow$false_option (V_2)	(4)

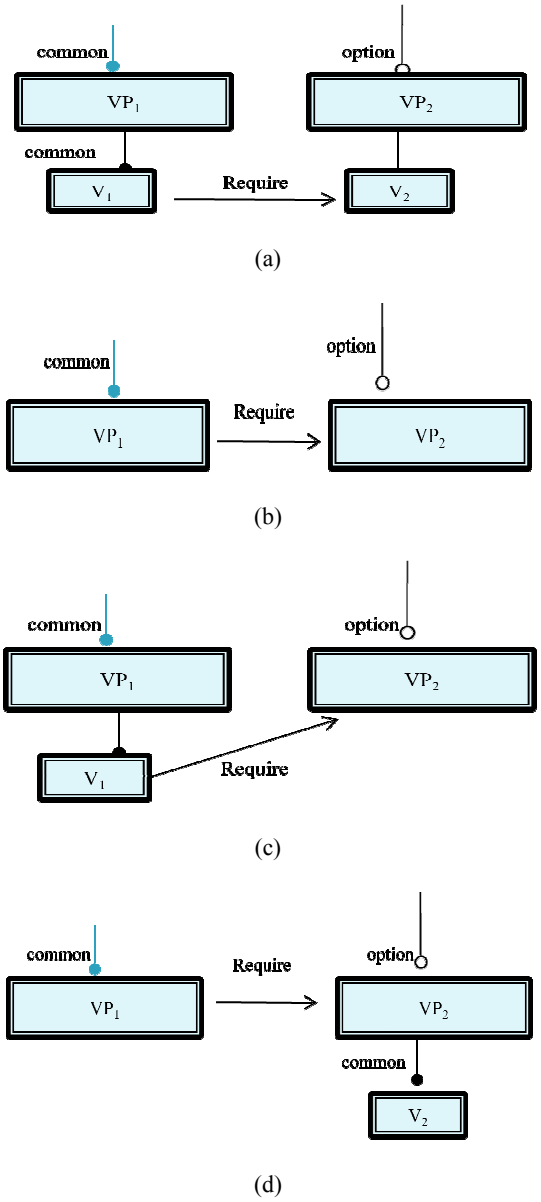

Fig. 2: Illustration of false-optional features detection rules

requires feature 'D', which means that feature 'D' must be included in all products. This property formulates feature 'D' as a common feature. Thus, feature 'D' has the same behavior as a common feature but is not labeled as a common feature, which means feature 'D' is a false-optional feature.

The general pattern to describe false-option feature detection is:

Feature f_1 is common and requires feature f_2 and feature f_2 not common.

As mentioned earlier, there are three implementations for the require relation: variant requires another variant, variant requires variation point and variation point requires another variation point. Table 1 shows the rules for detecting the false-optional variants and false-optional variation points. In Rule 1, V_1 is a common variant belonging to a common variation point VP_1. V_1 requires V_2 which means V_2 must be included in any product, but V_2 labeled as a not common feature. Therefore, V_2 is a false-optional feature. Rules 2 and 3 detect false-optional variation points. In this case, it is clear that all the common variants belonging to the false-optional variation point are false-optional variants. Rule 4 explains how to detect the common variants belonging to the false-optional variation point. Figure 2 shows an illustration of the false-optional features detection rules. Figure 2a illustrates rule 1, Fig. 2b illustrates rule 2, Fig. 2c illustrates rule 3 and Fig. 2d illustrates rule 4.

Wrong cardinality detection: Cardinality is wrong if the maximum or minimum number allowed to select from a variation point cannot be implemented (Maßen and Lichter, 2005). In this subsection, we introduce four types of the wrong cardinality as one of our contributions in this study: (Examples are based on Fig. 3 to 6). In these figures the letter Y represents variation point; the letter R denotes require relation and the letter E denotes exclude relation. The numbers between brackets represent cardinality; minimum and maximum number allowed to be selected from the variation point:

- **Maximum wrong cardinality:** In this type, maximum cardinality cannot be implemented by any means. In Fig. 3, X1 requires X2, X2 requires X3 and X3 requires X1. In Fig. 3, maximum cardinality is defined as 2. According to the require constraints, this maximum cardinality cannot be implemented.
- **Minimum wrong cardinality:** In this type, minimum cardinality cannot be implemented by any means. In Fig. 4, X1 excludes X2, X2 excludes X3 and X3 excludes X1. Minimum cardinality is defined as 2. According to the exclude constraints, this minimum cardinality cannot be implemented.

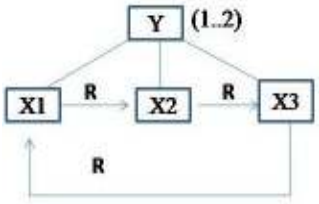

Fig. 3: Maximum wrong cardinality

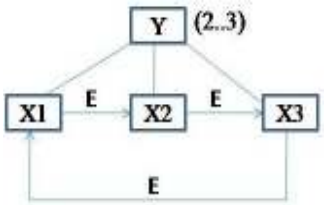

Fig. 4: Minimum wrong cardinality

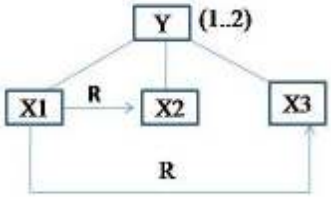

Fig. 5: Max possibly wrong cardinality

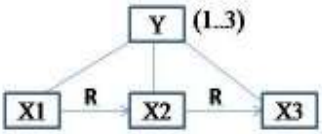

Fig. 6: Min possibly wrong cardinality

- **Maximum possibly wrong cardinality:** In this type, maximum cardinality cannot be implemented in some cases, i.e., not all selections are true. In Fig. 5, regarding to the maximum cardinality, selection of (X2, X3) is correct but selection of (X1, X2) and (X1, X3) are incorrect.
- **Minimum possibly wrong cardinality:** The minimum wrong cardinality occurs in some cases, i.e., not all selections are true. As an example, in Fig. 6, a selection of (X3) is true but selections of (X1) or (X2) is not true (because X1 requires X2 and X2 requires X3, therefore any selection of X1 followed by selections of X2 and any selection of X2 followed by selection of X3).

Detection rules: Although the wrong cardinality occurs in domain engineering, the detection rules work in the configuration process. Comparing the number of selected variants with the cardinality for each variation point is the main issue in the deducing process. We define a special predicate (no-selected (x)) for counting the number of selected variants.

Table 2: Wrong-cardinality detection rules

Definition type (x, variant) ∧type (y, variationpoint) ∧variants (y, x)
∀y, x, n, m: ∧ (no_selected (x) >max (y, n)) ⟹ max-wrong-cardinality. (5)
∀y, x, n, m: (no_selected (x) <min (y, n)) ⟹ min-wrong-cardinality. (6)

Table 2 shows the detection rules for wrong-cardinality.

In Rule 5, in all cases, the number of selected variants (x) belonging to the variation point y is greater than the maximum number (n) allowed to be selected from y. This is a maximum wrong cardinality. In Rule 6, in all cases, the number of selected variants (x) belonging to the variation point y is less than the minimum number (n) allowed to be selected from y. This is a minimum wrong cardinality.

Replacing all (∀) sign by there exist (∃) sign allows the same rules to be used for detecting the maximum and minimum possibly wrong cardinality.

SCALABILTY TESTING

Scalability is a key factor in measuring the applicability of the techniques dealing with variability modeling in domain engineering (Segura, 2008). The wrong cardinality detection rules work within the configuration process (application engineering process). Scalability is not a critical issue in the configuration process. Therefore, in this section, we discuss the experiments related to redundancy detection. Testing the output-time is our objective from these experiments. In the following, we describe the method of our experiments:

- **Define the assumptions:** We have two assumptions:
  - Each variation point and variant has a unique name.
  - All variation points have the same number of variants.
- **Generate data set to represent the domain engineering:** Domain engineering is generated in terms of predicates (variation points and variants). We generated four sets containing 1000, 5000, 15000 and 20000 variants. Variants are defined as numbers represented in sequential order. For example, in the first set (1000 variants) the variants are: 1, 2, 3,…, 1000. In the last set (20000 variants) the variants are: 1, 2, 3, …, 20000. The number of variation point in each set is equal to number of variant divided by five, which means each variation point has five variants. As an example in the second set (5000 variants), the number of variation points equal 1000. Each variation point defined as sequence number having the term vp as postfix, e.g., vp12.
- **Set the parameters:** the main parameters are the number of variants and the number of variation points. The remaining eight parameters (common

Table 3: Snapshot of experiment dataset

type (vp1, variationpoint). type (1, variant).
Variants (vp1, 1).
Common (570, yes).
Common (vp123, yes).
requires_v_v (7552, 2517).
requires_vp_vp (vp1572, vp1011).
excludes_vp_vp (vp759, vp134).
excludes_v_v (219, 2740).
requires_v_vp (3067, vp46).
excludes_v_vp (5654, vp1673).

Table 4: A prolog program to detect the false-optional features

```
det:-
caseone, casetwo, casethree, nl, told.
caseone:-
variants (Y, X), common (Y, yes), common (X, yes), requires_v_v
(X, N), variants (Z, N), not (common (Z, no)), common (N, yes),
write ('1....False Option ....'), write (N), nl, fail, det.
caseone:- true.
casetwo:-
common (Y, yes), requires_vp_vp (Y, Z), not (common (Z, yes)),
variants (Z, N),
common (N, yes), write (' 2.... False Option ....'), write (N),
nl, fail, det.
casetwo:- true.
casethree:-
variants (Y, X), common (Y, yes), common (X, yes), requires_v_vp
(X, Z),
not (common (Z, no)), variants (Z, N), common (N, yes), write
('3....False Option ....'), write (N),
nl, fail, det.
casethree:- true.
```

variants, common variation points, variant requires variant, variant excludes variant, variation point requires variation point, variation point excludes variation points, variant requires variation point and variant excludes variation point) are defined as a percentage of the number of variants or variation points. Three ratios are defined: 10, 25 and 50% respectively. The number of the parameters related to variant (such as common variant, variant requires variant, variant excludes variant, variant requires variation point and variant excludes variation point) is defined as a percentage of the number of the variants. The number of parameters related to variation point (such as; variation point requires variation point) is defined as a percentage of the number of variation points. Table 3 represents snapshots of an experiment dataset, i.e., the domain-engineering in our experiments.

- **Calculate output:** For each set, we made thirty experiments and calculated the execution time as average. The experiments were done with the variant range (1000-20000) and percentage range of 10, 25 and 50%, respectively of constraint dependency rules.

Table 4 shows Prolog software for detecting false-optional features.

The wrong cardinality detection is happened in configuration time. Thus, the scalability is not an issue. In the following parts of this subsection, the results of false-optional detection operation are presented. The results show the execution time compared with number

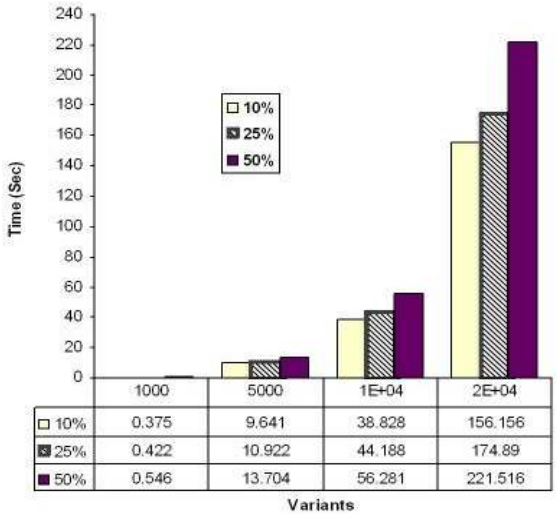

Fig. 7: Results of false-optional detection scalability test

of variants, number of variation points and the eight parameters (the six dependency constraint rules, common variants and common variation points).

Figure 7 shows the result of scalability test for false-optional detection. Our results show that the proposed operations can deal with large number of features (20,000) in reasonable time.

DISCUSSION AND COMPARISON WITH PREVIOUS WORK

Generally, the problem of the current research is that the checking of the software product's correction only happen after it has been developed as in the process of application engineering. This is not feasible to ensure the correctness of the SPL because the medium-size SPL can contain huge number of software products.

The proposed rules detect false-optional in the domain-engineering process which is a novel. According to Benavides *et al*. (2010) our method is the first method that detects false-optional in the domain engineering which makes the validation process is feasible and practicable for industrial SPL.

In respect of the scalability results of the domain engineering, we conducted experiments for SPLs with ranges of up to 20,000 variants and up to 50% of constraint dependency rules and we were able to obtain results in a good time. In White *et al*. (2008) the scalability is done by 5,000 features in one minute. In Segura (2008), the execution time for 200-300 features is 20 min after applying atomic sets to enhance the scalability. When compared to the literature, it can be seen that our proposed method is scalable. The scalability of our approach is good enough when compared with the literature because we first define special patterns and later the system searches only for these patterns. As a consequence, the searching time is acceptable.

FAMA framework (Trinidad *et al*., 2008a) defined a deductive operation for wrong cardinality. Our proposed approach defined four types of wrong cardinality. Moreover, the proposed approach defines a detective rule for each type of wrong cardinality, which is novel.

Generally, researchers (in the literature) used solvers for validating SPL by generating all products and detect errors in each product. The process of generating all products is a very tough process and almost impossible with large-size SPL (Benavides *et al*., 2009). Our approach is based on validating SPL by detecting errors in domain-engineering. We proposed a methodology based on defining a general pattern for each error. Later, all the implemented cases of this general pattern are defined. Using this general pattern all cases of redundancy and wrong cardinality could be detected. This methodology could be used to solve other problems in SPL.

The proposed approach enhance the searching process (in SPL) by predefine specific cases (each rule represent case) and search only for them. Therefore, our scalability experiments show good results.

CONCLUSION

The proposed method deals with the complexity of detecting anomalies in domain engineering. Two types of anomalies are discussed in this study: false-optional and wrong cardinality. Deducing rules are presented to deduce redundancy and wrong cardinality. The proposed method is based on modeling variability using predicates, then defining a general form for each type. These definitions formulate the problems and allow the FOL rules to deduce the results from predefined cases.

The problems that are discussed in this research could be found in any SPL regardless of the technique used for modeling variability. Wrong-cardinality and false-optional features could occur in an SPL due to the wrong usage of constraint dependency rules. Although these problems are very clear in both the FM and the OVM, it still could occur in all types of variability modeling techniques. Since any SPL has a group of features (by which we mean software assets) that are collected in the domain-engineering process, wrong usage of the dependency rules leads to these problems.

Many methods are applying empirical results to test scalability by generating random FMs (Segura, 2008; Trinidad *et al*., 2008a, b; Yan *et al*., 2009). Comparing the literature, our test range (1000-20,000 variants) is sufficient to test scalability. The proposed method is limited to work only in certain environment, i.e., where constraint dependency rules are well known in all cases.

We now are developing a software tool that allows users to model their SPL using the proposed method. The tool provides direct link between the two layers and implements our proposed validation operation. Moreover, it provides scalability tests.

Our approach is limited to work only in a certain environment, i.e., where constraint dependency rules are well known in all cases. In some SPL, constraint dependency rules are different from product in product. We called these types of SPL as uncertain SPL environments. As a future work, our approach could be extended to handle uncertain SPL environments using case-based reasoning. In domain engineering, our approach is used to detect wrong cardinality and false-optional features. As a future extension of this study, some new rules could be developed for auto-correction of these errors.

REFERENCES

Batory, D., 2005. Feature models, grammars and propositional Formulas. Proceeding of the the 9th International Software Product Lines Conference (SPLC05). Rennes, France.

Batory, D., D. Benavides and A. Ruiz-Cortes, 2006. Automated analysis of feature models: Challenges ahead. Commun. ACM, 49(12): 45-47.

Benavides, D., A. Metzger and U. Eisenecker, 2009. Main introduction of the proceeding of Third International Workshop on Variability Modeling of Software-intensive systems. Spain.

Benavides, D., S. Segura and A. Ruiz-Cortés, 2010. Automated analysis of feature models 20 years later: A literature review. Inform. Syst., 35(6): 615-636.

Benavides, D., A. Ruiz-Cort´es, D. Batory and P. Heymans, 2008. Main introduction of the proceeding of First International Workshop on Analyses of Software Product Lines (ASPL'08). Limerick, Ireland.

Beuche, D., H. Papajewski and W. Schröder-Preikschat, 2004. Variability management with feature models. Sci. Comput. Program., 53(3): 333-352.

Bosch, J., 2002. Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization. In: Chastek, G. (Ed.), Software Product Lines. Springer, Berlin, Heidelberg, pp: 257-271.

Czarnecki, K. and M. Antkiewicz, 2005. Mapping features to models: A template approach based on superimposed variants. Proceeding of the the 4th International Conference on Generative Programming and Component Engineering (GPCE'05). Tallinn, Estonia.

Czarnecki, K. and K. Pietroszek, 2006. Verifying feature-based model templates against well-formedness OCL constraints. Proceeding of the 5th International Conference on Generative Programming and Component Engineering (GPCE'06).

Eisenecke, U., S. Apel and S. Gnesi, 2012. Main introduction of the proceeding of Sixth International Workshop on Variability Modelling of Software-intensive Systems. Germany.

Elfaki, A., S. Phon-Amnuaisuk and C.K. Ho, 2008. Knowledge based method to validate feature models. Proceeding of the 1st International Workshop on Analyses of Software Product Lines (ASPL'08), Collocated with SPLC08. Limerick, Ireland.

Elfaki, A., S. Phon-Amnuaisuk and C. Ho, 2009a. Investigating Inconsistency Detection as a Validation Operation in Software Product Line. In: Lee, R. and N. Ishii (Eds.), Software Engineering Research, Management and Applications. Springer, Berlin, Heidelberg, pp: 159-168.

Elfaki, A., S. Phon-Amnuaisuk and C. Kuan Ho, 2009b. Using first order logic to validate feature model. Proceeding of the the 3rd International Workshop on Variability Modeling of Software-Intensive Systems. Sevilla, Spain.

Elfaki, A.O., S. Phon-Amnuaisuk and C.K. Ho, 2009c. Modeling variability in software product line using first order logic. Proceeding of the 7th ACIS International Conference on Software Engineering Research, Management and Applications (SERA'09). Haikou, Hainan Island, China, pp: 227-233.

Fan, S. and N. Zhang, 2006. Feature Model Based on Description Logics. In: Gabrys, B., R. Howlett and L. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems. Springer, Berlin, Heidelberg, pp: 1144-1151.

Heymans, P., K. Czarnecki and U. Eisenecker, 2011. Main introduction of the proceeding of Fifth International Workshop on Variability Modelling of Software-intensive Systems. Namur, Belgium.

Hotez, L. and T. Krebs, 2003. A knowledge based product derivation process and some idea how to integrate product development. Proceeding of the the Software Variability Management Workshop. Groningen, the Netherlands.

Hotz, L., A. Gunter and T. Krebs, 2003. A knowledge-based product derivation process and some ideas how to integrate product development. Proceeding of Software Variability Management Workshop, pp: 136-140.

Kang, K.C., J. Lee and P. Donohoe, 2002. Feature-oriented product line engineering. IEEE Software, 19(4): 58-65.

Kang, K.C., S.G. Cohen, J.A. Hess, W.E. Novak and A.S. Peterson, 1990. Feature-oriented Domain Analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, ESD-90-TR-222, Software Engineering Institute, Carnegie Mellon University.

Maßen, T. and H. Lichter, 2005. Determining the Variation Degree of Feature Models. In: Obbink, H. and K. Pohl (Eds.), Software Product Lines. Springer, Berlin, Heidelberg, pp: 82-88.

Mannion, M., 2002. Using first-order logic for product line model validation. Proceeding of the 2nd International Conference on Software Product Lines.

Pohl, K., G. Bockle and F. Van Der Linden, 2005. Software Product Line Engineering Foundations Principles and Techniques. Springer, Verlag Heidelberg, Germany.

Robak, S. and A. Pieczynski, 2003. Employing fuzzy logic in feature diagrams to model variability in software product-lines. Proceeding of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS03), pp: 305-311.

Segura, S., 2008. Automated analysis of feature models using atomic sets. Proceeding of the 1st International Workshop on Analyses of Software Product Lines (ASPL'08), Collocated with (SPLC08). Limerick, Ireland.

Sun, J., H. Zhang, Y. Fang and L.H. Wang, 2005. Formal semantics and verification for feature modeling. Proceeding of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pp: 303-312.

Svahnberg, M., J. Van Gurp and J. Bosch, 2005. A taxonomy of variability realization techniques. Software Pract. Exp., 35(8): 705-754.

Trinidad, P., B. David and A. Ruiz-Cort´es, 2006. Isolated features detection in feature models. Proceeding of the the Advanced Information Systems Engineering (CAiSE'06), Luxembour.

Trinidad, P., D. Benavides, A. Dura´n, A. Ruiz-Cortes and M. Toro, 2008a. Automated error analysis for the agilization of feature modeling 2008. J. Syst. Software, 81(6): 883-896.

Trinidad, P., D. Benavides, A. Ruiz-Cort´es, S. Segura and A. Jimenez, 2008b. FAMA framework. Proceeding of the 12th Software Product Lines Conference (SPLC08).

Wang, H.H., Y.F. Li, J. Sun, H. Zhang and J. Pan, 2007. Verifying feature models using OWL. Web Semantics: Sci. Serv. Agents World Wide Web, 5(2): 117-129.

White, J., D. Schmidt, D. Benvides, P. Trinidad and A. Ruiz-Cortes, 2008. Automated diagnosis of product line configuration errors on feature models. Proceeding of the 12th International Conference of Software Product Line. Limerick, Irland.

Yan, H., W. Zhang, H. Zhao and H. Mei, 2009. An Optimization Strategy to Feature Models' Verification by Eliminating Verification-Irrelevant Features and Constraints. In: Edwards, S. and G. Kulczycki (Eds.), Formal Foundations of Reuse and Domain Engineering. Springer, Berlin, Heidelberg, pp: 65-75.

Zhang, W., H. Zhao and H. Mei, 2004. A propositional logic-based method for verification of feature models. Proceeding of the the 6th International Conference on Formal Engineering Methods (ICFEM 04).