Establishing Software Product Lines from Existing Products Based on Feature Model Recovery and Merging

VALENTINO VRANIĆ and MICHAL GRANEC, Institute of Informatics, Information Systems and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava

This paper introduces an approach to establishing software product lines from a set of existing products based on a reversed Lee–Kang design feature based approach to deriving program code from features, Van den Broek's feature model merging combined with partial feature models, and aspect-oriented refactoring strategies cataloged by Monteiro and Fernandes for implementing the features as modules that can be plugged in or out as needed. The approach interlinks the feature model merging cases with the actual implementation of features and specific aspect-oriented refactoring strategies used for this. The approach was successfully applied to JForum, establishing a product line from three existing products prepared out of the original, publicly available JForum code.

1. INTRODUCTION

Software development endeavors typically start with one specific product. Following a successful deployment, another order of a similar product often occurs. The time is always short and what might have been taken from the first product is reused with the necessary customization performed on-the-fly. Documenting is typically postponed for some less rush times. Then comes another order, and yet another one... The company grows, people change, further development teams are formed... The rush times never provide a break for documenting. Many similar versions of the product linger around. Features are being unnecessarily reimplemented because no one has an overview of what is available in this unrecognized software product line.

Software product lines are a well-known approach to software reuse based on explicit and up-front treatment of variability within one domain. This is performed at the level of features, which simply represent whatever properties of the software systems in a given domain are identified as important. Their variability indicates appropriate implementation mechanisms.

Establishing a software product line from a set of existing products may be achieved evolutionary or revolutionary [Bosch 2000]. The approach proposed in this paper works either way. It is based on a reversed Lee–Kang design feature based approach to deriving program code from features [Lee and Kang 2013], Van den Broek's feature model merging [van den Broek 2012] combined with partial feature models [Menkyna and Vranić 2012], and aspect-oriented refactoring strategies cataloged by

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-15-0508 and Research & Development Operational Programme for the project Research of Methods for Acquisition, Analysis and Personalized Conveying of Information and Knowledge, ITMS 26240220039, co-funded by the ERDF.
The authors' addresses: Valentino Vranić, Ilkovičova 2, 84216 Bratislava, Slovakia; email: vranic@stuba.sk.

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2018: 7th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Novi Sad, Serbia, 27–30.8.2018. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

Monteiro and Fernandes [Monteiro and Fernandes 2005; Monteiro 2004] for implementing the features as modules that can be plugged in or out as needed.

The rest of the paper is organized as follows. Section 2 presents the overall approach. Section 3 explains how product feature models are recovered. Section 4 explains the process of merging product feature models into a product line feature model. Section 5 describes the feature module implementation strategy. Section 6 discusses the evaluation. Section 7 compares the approach proposed in this paper with related work. Section 8 concludes the paper.

2. THE OVERALL APPROACH

The overall approach to establishing software product lines from existing products is depicted in Figure 1. As can be seen from the figure, the input to the approach are existing software products, including their documentation, if available. After a series of activities, the approach produces its final outputs: the feature model of a newly established software product line and feature modules, the modules that implement features [Apel et al. 2008].

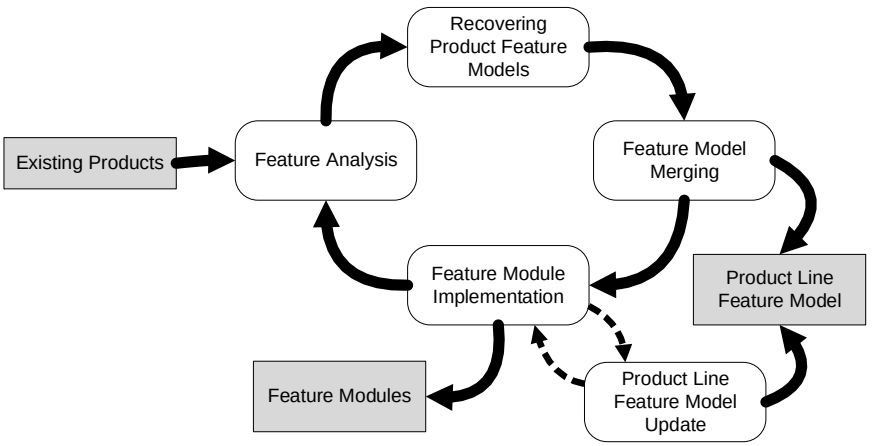

Fig. 1. The overall approach.

The approach is iterative and incremental in its nature, which is indicated by a circular flow between its main activities. This means that the individual steps should be treated within a reasonable time frame and proceed to the next steps as soon as possible rather than refining them indefinitely. The next steps will quickly confirm or refute the assumptions from previous steps forcing a re-iteration if necessary.

The approach will be explained on a study performed on JForum, an open source discussion forum implemented in Java [The JForum Team 2018]. Out of the original version, which will be referred to as *JForum Original*, two other versions were created denoted *JForum Simple* and *JForum Different Base*. *JForum Simple* offers only elementary features, mimicking a situation when this is explicitly required by a customer or when a company wants to offer a simpler and cheaper version. Uncoordinated, these two versions will continue to be developed separately. At some point, their merge extended by some further features might be created. This is what the third product denoted as *JForum Different Base* represents.

3. RECOVERING PRODUCT FEATURE MODELS

Constructing a product line feature model as a common feature model of a set of related products is a complex task. Here, it consists of recovering feature models out of individual products, discussed in this section, followed by merging them together, discussed in Section 4.

The most important and relevant artifact for recovering a feature model out of a given product is its source code. Models and documentation may be consulted, but should be used with caution, as they rarely fully correspond to code.

Since features in feature modeling, which should be both comprehensible and usable for configuration purposes, tend to be at a higher level of abstraction than code, it's convenient to first create their approximation closer to the actual code modules and then make appropriate abstraction out of this initial *design feature model*. This is exactly how the Lee–Kang *design feature based approach to deriving program code from features* [Lee and Kang 2013] works. Here, for the purposes of recovering feature models from software products, the Lee–Kang method has been reversed within the process of creating *partial feature models* [Menkyna and Vranić 2012], which will be explained on an example.

Consider JForum Simple. Figure 2 shows selected features of its initial partial feature model. The basic Czarnecki–Eisenecker feature modeling notation [Czarnecki and Eisenecker 2000] is used. The root node represents the JForum Simple system as such. Within the first iteration, the identified design features are attached directly to the concept node (the root) as optional features (indicated by an empty circle) without trying to determine precise feature relationships. The square brackets around the feature name mean that the feature is open, i.e., further subfeatures are expected.

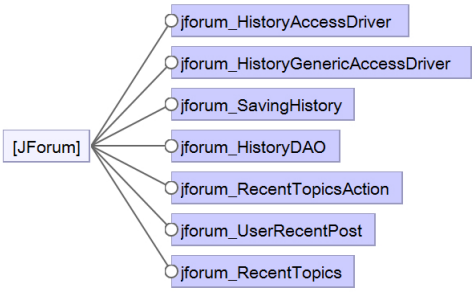

Fig. 2. Initial partial feature model of JForum Simple.

Afterwards, the work goes towards organizing the discovered features and correcting their variability and also towards discovering further features, some of which are abstract (indicated by a lighter background), i.e., no feature modules correspond to them. Figure 3 shows selected features of the resulting JForum Simple feature model.

Figure 4 shows the feature model constructed for JForum Original. The JForum Different Base feature model is very similar: it only lacks the *Cache* feature (and, consequently, its subfeature, *TopicCache*), as well as the additional restriction. Additional restrictions are textually defined relationships between features. They are typically used to express relationships whose inclusion would compromise the tree structure of a feature diagram. Logical operators are used to form additional constraints [Vranić 2005].

In general, the approach proposed here focuses on variable features, which are essential to defining a software product line, and does not capture thoroughly all the common (mandatory) features.

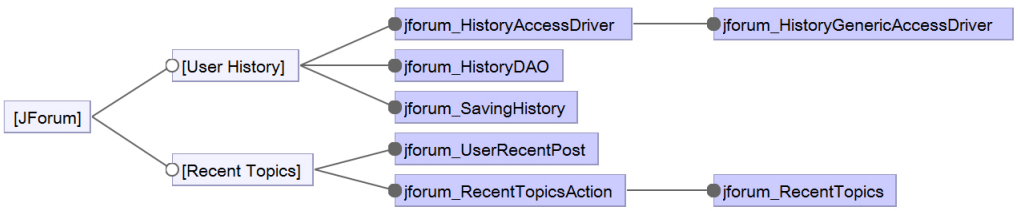

Fig. 3. Complete partial feature model of JForum Simple.

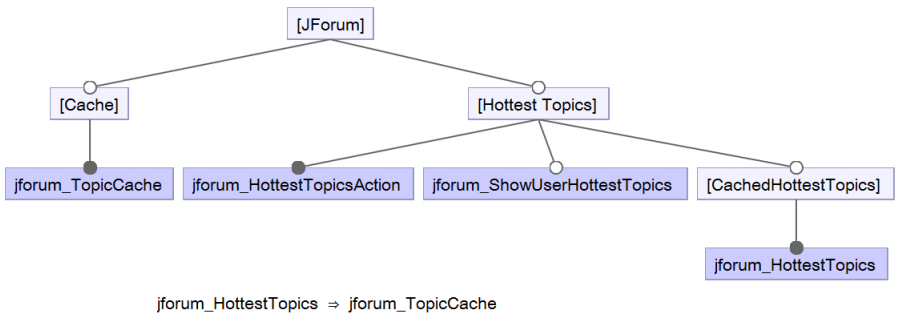

Fig. 4. Complete partial feature model of JForum Original.

4. FEATURE MODEL MERGING

The overall feature model reflecting the contemporary state of the software product line is obtained by merging feature models of individual products. Feature model merging is performed using Van den Broek's approach [van den Broek 2012]. This approach operates upon a set and predicate logic representation of feature models, such as this one of JForum Simple:

FE(JForumSimple) = {JForum, User History, Recent Topics, HistoryAccessDriver, HistoryGenericAccessDriver, HistoryDAO, SavingHistory, UserRecentPost, RecentTopicsAction, RecentTopics}

RO(JForumSimple) = JForum

PA(JForumSimple, User History) = JForum

PA(JForumSimple, Recent Topics) = JForum

PA(JForumSimple, HistoryAccessDriver) = User History

PA(JForumSimple, HistoryDAO) = User History

PA(JForumSimple, SavingHistory) = User History

PA(JForumSimple, HistoryGenericAccessDriver) = HistoryAccessDriver

PA(JForumSimple, UserRecentPost) = Recent Topics

PA(JForumSimple, RecentTopicsAction) = Recent Topics

PA(JForumSimple, RecentTopics) = RecentTopicsAction

PC(JForumSimple, JForum) = {{}, {User History}, {Recent Topics}, {User History, Recent Topics}}

PC(JForumSimple, User History) = {{HistoryAccessDriver, HistoryDAO, SavingHistory}}

PC(JForumSimple, HistoryGenericAccessDriver) = {{}}

PC(JForumSimple, HistoryDAO), PC(JForumSimple, SavingHistory) = {{}}

PC(JForumSimple, UserRecentPost), PC(JForumSimple, RecentTopics) = {{}}

PC(JForumSimple, RecentTopicsAction) = {{RecentTopics}}
RE(JForumSimple) a EX(FM2) = {}
AF(JForumSimple) = {JForum, User History, Recent Topics}

Here, JForumSimple is a name of a feature model. The rest of the identifiers represent features. The feature model is defined using the predicates for retrieving the set of the features (FE), concept[1] (RO), feature parent (PA), possible sets of feature children in feature model configurations (PC), require constraints (RE), and mutual exclusion constraints (EX). An additional predicate has been introduced to indicate retrieving of abstract features (AF).

Feature models are merged one by one, gradually accumulating the features of the individual products in the product line feature model. Van den Broek's feature model merging is based on making an intersection of the two feature models to be merged after enriching each one of them with the differentiating features occurring in the other one. These features are added as optional features. The feature models being merged are assumed to fulfill so-called parent compatibility, which means that equal features have equal parent features [van den Broek 2012].

With respect to the application of Van den Broek's approach within the approach proposed in this paper, several distinctive cases have been identified. These cases, presented in the following paragraphs, also affect the implementation of the corresponding feature module, which should be considered along.

A feature occurring in a product feature model, but not in the product line feature model should be added to the product line feature model. In implementation, this mostly requires just refactoring the code that corresponds to this feature.

A feature missing in a product feature model, but mandatory in the product line feature model should be transformed into a variable feature. In implementation, aspect-oriented refactoring of the corresponding common base code may be performed.

A feature with varying implementation in different products poses a non-trivial problem. If it's caused by varying behavior being required, it may be resolved by introducing alternative design features. If the behavior is essentially one and variance in the implementation is only accidental, this may be resolved by refactoring. If this fails, the solution should be resorted to introducing alternative design features. The variance might as well be caused by an additional functionality present in one of the products. This functionality may be treated as a new optional feature or it may be incorporated as mandatory to all the products.

A feature with equal implementation in all products indicates a mistake in feature identification, as such a feature should have been identified as mandatory and probably not even introduced into product feature models (recall from Section 3 that the approach proposed here tends to omit mandatory features focusing on variability).

A feature with different names in different products might be tricky to discover during feature model merging as its consequences become apparent only during feature module implementation. Of course, the solution is to name all such features equally.

Figure 5 shows selected features of the merged feature model reconstructed from its set and predicate logic representation.

5. FEATURE MODULE IMPLEMENTATION

In order to keep features pluggable, the corresponding feature modules are implemented using aspect-oriented programming. With an object-oriented code base, aspect-oriented refactoring is required.

[1] denoted by Van den Broek as the root feature

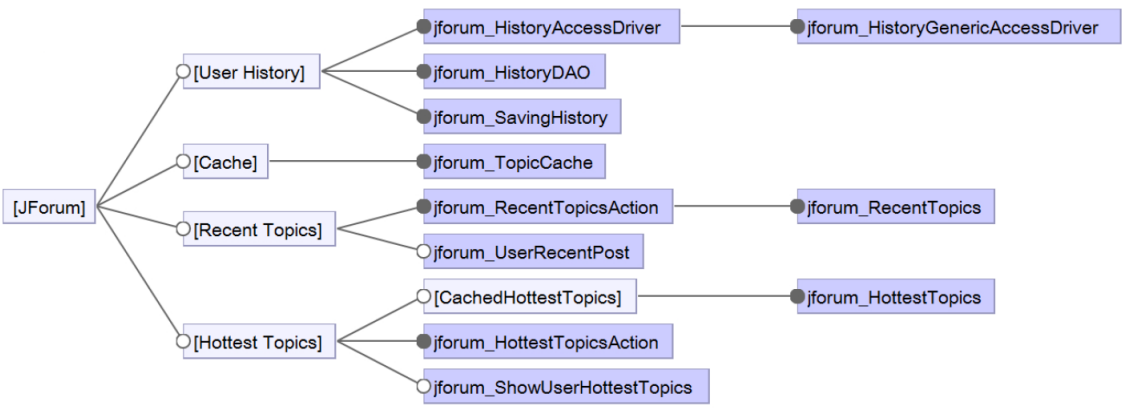

Fig. 5. JForum merged feature model.

Aspect-oriented refactoring strategies cataloged by Monteiro and Fernandes are applied [Monteiro and Fernandes 2005; Monteiro 2004], putting them into the context of product lines. Among these, the Move Field from Class to Inter-Type Declaration and Move Method from Class to Inter-Type Declaration strategies are in particular useful to implement the features occurring in a product feature model, but not in the product line feature model and features missing in a product feature model, but mandatory in the product line feature model.

The Partition Constructor Signature strategy is also applicable to the features occurring in a product feature model, but not in the product line feature model.

The Extract Advice strategy is the most versatile refactoring. It can be used to implement the features with varying implementation in different products. It can also be used to implement the features occurring in a product feature model, but not in the product line feature model when Move Field from Class to Inter-Type Declaration and Move Method from Class to Inter-Type Declaration are not applicable due to the necessity of preserving the corresponding attributes or methods in their original place.

As it was indicated in the previous section, these refactoring strategies affect the product line feature model as well.

6. EVALUATION

As it was explained in Section 2, the approach proposed here was applied to JForum, creating a product line out of three existing products prepared out of the original, publicly available JForum code [The JForum Team 2018]. For aspect-oriented code, AspectJ was used. Table I indicates the size of the study. As expected, JForum Different Base caused the biggest number of merging conflicts.

Table I. Features and conflicts in the JForum study.

Number of	JForum Simple	JForum Original	JForum Different Base	JForum Product Line
Abstract Features	5	20	17	24
Design Features	11	33	27	49
Merging Conflicts	1	5	10	n/a

In merging, features occurring in a product feature model, but not in the product line feature model occurred most frequently. This indicates the extent of the variability among the products.

Features with varying implementation in different products also occurred frequently. It was caused by how JForum Different Base differed from the other two products.

The other three merging cases mentioned in Section 4 have not been observed probably because of a thorough initial analysis. Moreover, the study was of a modest size and constructed with product lines in mind.

7. RELATED WORK

Liu et al. proposed an approach to feature oriented refactoring [Liu et al. 2006]. However, they do not deal explicitly with feature model merging.

Yoshimura et al. proposed an approach to assess the merging of existing systems into a product line and a method to identify the commonality among them [Yoshimura et al. 2006]. Instead on commonality, the approach proposed in this paper focuses on variability as a driving force in product lines making it fully operational on both feature modeling and code level.

Shatnawi et al. proposed an approach to recovering software product line architecture of a family of object-oriented product variants [Shatnawi et al. 2016]. Similarly as in the approach proposed in this paper, they focus on variability. However, they do not target a complete implementation, but remain on the architecture level.

8. CONCLUSIONS AND FURTHER WORK

This paper introduces an approach to establishing software product lines from a set of existing products based on a reversed Lee–Kang design feature based approach to deriving program code from features [Lee and Kang 2013], Van den Broek's feature model merging [van den Broek 2012] combined with partial feature models [Menkyna and Vranić 2012], and aspect-oriented refactoring strategies cataloged by Monteiro and Fernandes [Monteiro and Fernandes 2005; Monteiro 2004] for implementing the features as modules that can be plugged in or out as needed. The approach interlinks the feature model merging cases with the actual implementation of features and specific aspect-oriented refactoring strategies used for this. The approach approach was successfully applied to JForum, creating a product line out of three existing products prepared out of the original, publicly available JForum code.

Apart from the implementation, the approach is independent of the programming language being used. The implementation part relies on AspectJ style of aspect-oriented programming since this is the basis the aspect-oriented refactoring strategies employed in the approach work on. However, there are indications that some programming languages not explicitly developed as aspect-oriented exhibit aspect-oriented features [Bálik and Vranić 2012]. Aspect-oriented frameworks, which are available for many programming languages, may be used, too. The problem is that these frameworks are often obsolete with respect to the underlying programming language due to the lack maintenance support.

The next steps may lead towards improving the maintenance of the interconnection of features and feature modules, partial automation of feature identification and feature model merging (possibly with an application of design pattern detection techniques [Polášek et al. 2012]) and semi-automatic aspect-oriented refactoring [Pipík and Polášek 2013], and a deeper examination of the approach on larger studies. In the long run, dynamic code structuring [Nosáľ and Porubän 2012; Nosáľ et al. 2013; Porubän and Nosáľ 2014] may be considered as an alternative to the fixed code representation. Also, layered 3D visualization of software models [Ferenc et al. 2017; Gregorovič and Polášek 2015; Gregorovič et al. 2015] possibly with virtual reality [Vincúr et al. 2017b; Vincúr et al. 2017a] could be applied to make simultaneous work with several feature models and code easier and more transparent.

REFERENCES

Sven Apel, Thomas Leich, and Gunter Saake. 2008. Aspectual Feature Modules. *IEEE Transactions On Software Engineering* 34 (2008), 143–168.

Jaroslav Bálik and Valentino Vranić. 2012. Symmetric Aspect-Orientation: Some Practical Consequences. In *Proceedings of NEMARA 2012: International Workshop on Next Generation Modularity Approaches for Requirements and Architecture, at AOSD 2012*. ACM, Potsdam, Germany.

Jan Bosch. 2000. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley.

Pim van den Broek. 2012. Intersection of Feature Models. In *Proceedings of 16th International Software Product Line Conference, SPLC 2012*, Vol. 2. ACM, Salvador, Brazil.

Krzysztof Czarnecki and Ulrich W. Eisenecker. 2000. *Generative Programing: Methods, Tools, and Applications*. Addison-Wesley.

Matej Ferenc, Ivan Polášek, and Juraj Vincúr. 2017. Collaborative Modeling and Visualisation of Software Systems Using Multidimensional UML. In *Proceedings of 5th IEEE Working Conference on Software Visualization, VISSOFT 2017*. IEEE, Shangai, China.

Lukáš Gregorovič and Ivan Polášek. 2015. Analysis and Design of Object-Oriented Software Using Multidimensional UML. In *Proceedings of 15th International Conference on Knowledge Technologies and Data-Driven Business*. ACM, Graz, Austria.

Lukáš Gregorovič, Ivan Polášek, and Branislav Sobota. 2015. Software Model Creation with Multidimensional UML. In *Proceedings of 9th IFIP WG 8.9 Working Conference, CONFENIS 2015, part of WCC 2015 (LNCS 9357)*. Springer, Daejeon, Korea.

Hyesun Lee and Kyo Chul Kang. 2013. A Design Feature-Based Approach to Deriving Program Code from Features: A Step Towards Feature-Oriented Software Development. In *Proceedings of 7th International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS 2013*. ACM, Pisa, Italy.

Jia Liu, Don Batory, and Christian Lengauer. 2006. Feature Oriented Refactoring of Legacy Applications. In *Proceedings of 28th International Conference on Software Engineering, ICSE '06*. ACM, Shanghai, China.

Radoslav Menkyna and Valentino Vranić. 2012. Aspect-Oriented Change Realization Based on Multi-Paradigm Design with Feature Modeling. In *Proceedings of 4th IFIP TC2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2009, Revised Selected Papers (LNCS 7054)*. Springer, Krakow, Poland.

Miguel Pessoa Monteiro. 2004. Object-to-Aspect Refactorings for Feature Extraction. In *Proceedings of 3rd International Conference on Aspect-Oriented Software Development, AOSD 2004*. ACM, Lancaster, UK.

Miguel P. Monteiro and João M. Fernandes. 2005. Towards a Catalog of Aspect-Oriented Refactorings. In *Proceedings of 4th International Conference on Aspect-oriented Software Development, AOSD 2005*. ACM, Chicago, Illinois.

Milan Nosáľ and Jaroslav Porubän. 2012. Supporting Multiple Configuration Sources Using Abstraction. *Central European Journal of Computer Science* 2, 3 (2012), 283–299.

Matej Nosáľ, Jaroslav Porubän, and Milan Nosáľ. 2013. Concern-Oriented Source Code Projections. In *Proceedings of 2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013*. IEEE, Kraków, Poland, 1541–1544.

Roman Pipík and Ivan Polášek. 2013. Semi-Automatic Refactoring to Aspect-Oriented Platform. In *Proceedings of 14th IEEE International Symposium on Computational Intelligence and Informatics, CINTI 2013*. IEEE, Budapest, Hungary.

Ivan Polášek, Peter Líška, Jozef Kelemen, and Ján Lang. 2012. On Extended Similarity Scoring and Bit-vector Algorithms for Design Smell Detection. In *Proceedings of 2012 IEEE 16th International Conference on Intelligent Engineering Systems, INES 2012*. IEEE, Lisbon, Portugal, 115–120.

Jaroslav Porubän and Milan Nosáľ. 2014. Leveraging Program Comprehension with Concern-Oriented Source Code Projections. In *Proceedings of 3rd Symposium on Languages, Applications and Technologies, Slate'14*. Bragança, Portugal, 35–50.

Anas Shatnawi, Abdelhak-Djamel, and Houari Sahraoui. 2016. Recovering Software Product Line Architecture of a Family of Object-Oriented Product Variants. *The Journal of Systems and Software* 131 (2016), 325–346.

The JForum Team. 2018. JForum Discussion Board System. http://jforum.net/. (2018).

Juraj Vincúr, Pavol Návrat, and Ivan Polášek. 2017a. VR City: Software Analysis in Virtual Reality Environment. In *IEEE International Conference on Software Quality, Reliability and Security, QRS 2017*. IEEE, Prague, Czech Republic.

Juraj Vincúr, Ivan Polášek, and Pavol Návrat. 2017b. Searching and Exploring Software Repositories in Virtual Reality. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology, VRST 2017*. ACM, Gothenburg, Sweden.

Valentino Vranić. 2005. Multi-Paradigm Design with Feature Modeling. *Computer Science and Information Systems Journal (ComSIS)* 2, 1 (2005), 79–102.

Kentaro Yoshimura, Dharmalingam Ganesan, and Dirk Muthig. 2006. Defining a Strategy to Introduce a Software Product Line Using Existing Embedded Systems. In *Proceedings of 6th ACM & IEEE International Conference on Embedded Software, EMSOFT 2006*. ACM, Seoul, Korea.