On the Relation of Variability Modeling Languages and Non-Functional Properties

Birte Friesel
birte.friesel@uni-osnabrueck.de
Osnabrück University
Osnabrück, Germany

Michael Müller
michael.mueller@uni-osnabrueck.de
Osnabrück University
Osnabrück, Germany

Matheus Ferraz
matheus.ferraz@uni-osnabrueck.de
Osnabrück University
Osnabrück, Germany

Olaf Spinczyk
olaf.spinczyk@uni-osnabrueck.de
Osnabrück University
Osnabrück, Germany

ABSTRACT

Non-functional properties (NFPs) such as code size (RAM, ROM), performance, and energy consumption are at least as important as functional properties in many software development domains. When configuring a software product line – especially in the area of resource-constrained embedded systems – developers must be aware of the NFPs of the configured product instance. Several NFP-aware variability modeling languages have been proposed to address this in the past. However, it is not clear whether a variability modeling language is the best place for handling NFP-related concerns, or whether *separate* NFP prediction models should be preferred. We shine light onto this question by discussing limitations of state-of-the-art NFP-aware variability modeling languages, and find that both in terms of the development process and model accuracy a separate NFP model is favorable. Our quantitative analysis is based on six different software product lines, including the widely used busybox multi-call binary and the x264 video encoder. We use classification and regression trees (CART) and our recently proposed *Regression Model Trees* [8] as separate NFP models. These tree-based models can cover the effects of arbitrary feature interactions and thus easily outperform variability models with static, feature-wise NFP annotations. For example, when estimating the throughput of an embedded AI product line, static annotations come with a mean generalization error of 114.5 % while the error of CART is only 9.4 %.

ACM Reference Format:
Birte Friesel, Michael Müller, Matheus Ferraz, and Olaf Spinczyk. 2022. On the Relation of Variability Modeling Languages and Non-Functional Properties. In *Proceedings of 26th ACM International Systems and Software Product Line Conference - Volume B (SPLC'22).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3503229.3547055

SPLC'22, September 12–16, 2022, Graz, Austria

ACM ISBN 978-1-4503-9206-8/22/09…$15.00
https://doi.org/10.1145/3503229.3547055

1 INTRODUCTION

In addition to *functional* properties (e.g. enabled features), software product line instances also have *non-functional* properties (NFPs) such as ROM/RAM requirements, latency, processing throughput, or energy consumption. In many domains, these are at least as important as functional properties. For instance, mass-produced embedded systems tend to use severely resource-constrained low-cost microcontrollers as execution platforms. Ignoring NFPs in this domain can cause tremendous extra costs, as sub-optimal configurations may quickly exhaust the available resources, resulting in the need to build the product with a more powerful (and, thus, more expensive) microcontroller instead. Subtle microcontroller differences may even raise the need for a hardware re-design, further complicating the matter.

To avoid this issue, product developers must be aware of both functional and non-functional properties of the configured product instance when configuring a software product line. Several NFP-aware variability modeling languages have been proposed to address this by means of feature-wise NFP annotations, such as ClaferMoo, Velvet, TVL, and VM [2, 3, 7, 11].

The contribution of this work to a possible "consensus on a simple feature modeling language"[1] is an answer to the question whether a feature modeling language should incorporate elements for handling NFP-related concerns. The alternative are separate NFP models that are connected to the variability model only on the level of the configuration tool. We illustrate both approaches in Figure 1.

We assume that a *variability modeling language* is used to describe a *variability model* (gray triangle in the figure). On the left, NFP-related data is attached to the elements of the variability model (e.g. optional *features*), and aggregate functions describe how the NFPs of individual features make up the NFPs of the final product. Therefore, specific language elements are required. In order to display NFP predictions, such as code size or performance indicators, the configuration tool has to evaluate the NFP annotations and aggregate functions based on the currently selected features. On the right, separate NFP models are used for NFP prediction. The NFP model is a function of arbitrary nature that takes a *feature vector* (i.e., a formal representation of the current SPL configuration) as input and returns the predicted NFP.

[1] Citation from the MODEVAR workshop's homepage.

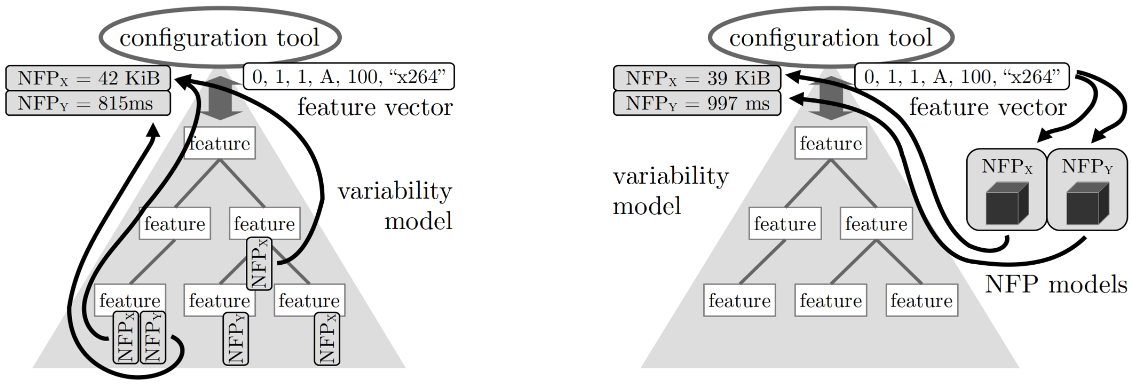

Figure 1: Alternative approaches to handle NFP-related concerns in product line engineering: Integrated NFP-related information (left) vs. separate NFP models (right)

Our qualitative and quantitative analysis in this paper discusses and compares both approaches. Eventually, we come to the conclusion that separate NFP models are favorable. Thus, a variability modeling language should *not* incorporate language elements that can only be used for describing NFPs.

The remainder of this paper is structured as follows. Section 2 introduces the most widely known variability modeling languages that come with language elements for describing NFPs and gives a short overview on corresponding NFP models. The problem analysis in Section 3 explains the main drawbacks of an integrated modeling approach. Section 4 then presents our quantitative analysis and in Section 5 we draw some final conclusions.

2 RELATED WORK

Much effort has gone into the development of concise textual languages for variability models.

In the early stage of the development of variability languages, the need arose to express not only boolean feature toggles, but also feature attributes. Hence, variability languages such as Forfamel [10] and VSL [1] were developed that support parameterizing features with additional attributes (e.g. a clipboard's buffer size).

As a welcome side effect, these allowed system designers to express non-functional properties of individual product features by augmenting them with attributes such as cost or memory usage.

However, parameterized features alone turned out to be insufficient to determine properties of the entire product and find optimal configurations regarding those. TVL [7], Clafer [4, 5], and VELVET [11, 13] overcome this limitation by introducing simple aggregate functions in feature attributes, thus allowing reasoning about optimal configurations with respect to non-functional properties. For example, TVL supports the expression `sum(selectedChildren.textSize)`, stating that a parent feature's text segment size is the sum of the text segment size of all enabled child features.

These languages are suitable for non-functional properties that are a mere aggregation of per-feature values, such as product costs. However, some non-functional properties, such as energy, performance, and memory usage, cannot be determined via the aggregation of feature attributes alone. In many cases, they are susceptible to the interaction of many features. Thus, in addition to feature-wise annotations, modern variability languages such as ClaferMoo [3], SPL Conqueror [12, 14], and UVL [15] also include feature interactions in their variability models. For example, in ClaferMoo, engineers may state that the program size is `(sum Feature.binarySize) * (Debug && 1.1 : 1)`, thus expressing that the Debug feature increases it by 10 % on average.

SPL Conqueror includes a toolchain to automatically calculate NFP values for features, freeing SPL engineers from the need to manually annotate features. Even though SPL Conqueror appears to be the most advanced approach for integrating NFPs into feature models, it is limited to boolean features and only considers relatively simple interactions between features.

Overall, recent approaches for NFP-aware feature models focus on simple feature-wise annotations, often relying on language support for feature attributes. Although a questionnaire among 20 attendants of a past MODEVAR workshop indicates a general preference for modeling languages that support feature attributes [15] – and we agree that those are useful – in our opinion, it is still not decided whether *non-functional properties* (essentially being a specialization of feature attributes) should be integrated into feature models or not. Meanwhile, the open-source community appears to stick to expressing the impact of features on product quality and performance as mere help messages or in-line annotations within feature names rather than a formal non-functional property model.

3 PROBLEM ANALYSIS

Although many variability modeling languages provide support for expressing feature attributes, using these for NFP *modeling* is subject to limitations that can affect the usability of the model. These concerns can be grouped into five different topics: the annotation process, annotation consistency, language complexity, implementation dependency, and modularity.

3.1 Annotation Process

NFP-related data is usually not available during variability modeling. Furthermore, for complex product lines, not all possible product instances can be generated and measured. An SPL with just 40 features can generate up to 2^{40} (or about 10^{12}) different configurations, making benchmarking all possible configurations infeasible.

As such, data must be added after the product line is already in use, in a process of generating and measuring different configurations. Changing the NFP annotations in the variability model whenever new data trickles in is a tedious and error-prone task. Also, any change to the features or model may affect the already-present annotations, making their accuracy uncertain unless new measurements are performed. As the variability model gets more complex, the annotation process becomes more and more difficult, especially when done manually. This takes us to the next issue with variability modeling languages with integrated NFP annotations.

3.2 Annotation Consistency

When using a variability modeling language capable of representing NFPs, the annotated data and formulae must be added in some way, be it manually or by an automatic tool. The first case impacts product line development: as explained before, a domain expert will likely not have this kind of data in the early stages of development, and also would be obligated to add and update every NFP after new measurements.

The second case, in contrast, presents a contradiction. If annotations are automatically generated after measurements, there is no reason for keeping them in the variability modeling language. Having a dedicated, separate NFP model is sufficient, and allows performance predictions to be performed without adding an extra layer of complexity to the variability dimension.

3.3 Complexity

In SPLs, stakeholders want to extract the best possible variant of the model for their application. Thus, being able to accurately predict NFP values should be the core concern of NFP modeling. However, as NFP prediction using simple feature annotations with constants is incapable of expressing the influence of cross-cutting concerns on a product's non-functional properties, its accuracy is limited by design. Therefore, approaches in the literature have become more and more complex over time.

TVL [7] is an example of a language that supports NFPs as attributes that can be inserted into the model. The attributes can be from four different types (*int, real, string,* or *enum*), and the language supports operations that can be used to calculate the NFP of each configuration. However, it is not aimed at multi-objective optimization and does not account for feature interactions.

ClaferMoo[3] is another language that models NFPs by means of feature annotations. The corresponding implementation also addresses the need for multi-objective optimization, but only has limited support for feature interaction.

Tools like SPL Conqueror [12, 14] tackle the feature interaction issue. Here, three types of heuristics (pair-wise, higher-order, and hot-spot) are used to identify feature interactions and reduce the number of benchmark measurements needed to obtain a sufficiently accurate model. However, once again, it performs NFP measurements and NFP model generation *after* variability modeling. It can also generate quite complex model expressions for feature interaction, which is often good for model accuracy, but not necessarily helpful for humans working with the variability model.

The trend of getting more complex relations is natural: as models keep expanding, they are more likely to have interaction among their features [16]. Extrapolating this trend into the future means that at some point the rules or formulae in the variability model will be so complex that they have no value for humans, and, therefore, should not be part of the modeling language to begin with.

3.4 Implementation Dependency

One common concern when developing a modeling paradigm is ensuring that the process is independent of implementation factors. Having a dimension only for variability modeling adds more flexibility to the choice of different implementations, and reduces the number of properties the domain expert has to take into consideration. However, NFPs intrinsically depend on the implementation, and as such should not be added to the variability model. There are examples of frameworks that keep these independent, such as pure::variants[6], which is a commercial variant management tool that strictly follows this separation policy.

3.5 Modularity

NFP modeling and NFP model training can be performed in different ways. When using a specific variability language, there are limitations to the expressiveness concerning NFP-related data. TVL, VELVET, and ClaferMoo, for example, only support a small set of possible types and interactions of attributes. However, different methods for predicting NFPs may not fit well into these restricted implementations. In [12] a tool is used to generate NFP data with expressions that cover interactions between features. Another approach presented in [9] uses CART to predict the accuracy of configurations. Other methods, such as our RMT approach or even neural networks, could be used for the same effect. Keeping the NFP model separate from the variability model would allow changes to the NFP model at any time, without being limited by the requirements of the variability modeling language.

With these concerns in mind, we argue that a separation between variability model and NFP model is necessary.

4 COMPARISON

To support our arguments regarding model accuracy, we now compare an integrated feature-wise annotation method (i.e., NFP attributes integrated into the variability model) with two kinds of separate NFP models: Classification and Regression Trees (CART) and Regression Model Trees (RMT).

4.1 Investigated product lines

We examine non-functional properties of six different real-world software product lines.

The *busybox* multi-call binary is a software suite that provides an assortment of common Unix utilities within a single binary. It is specifically designed for use in embedded Linux appliances. Its variability model allows the selection of various utilities to be included in the binary, and configuring the features of each utility.

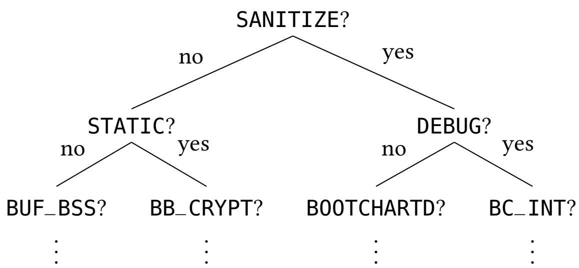

Figure 2: Excerpt of a Classification and Regression Tree (CART) model for busybox RAM usage. Each node holds a boolean decision related to a product line feature. Leaf nodes (not shown) express the NFP value for the partial configuration defined by the path from the root to the leaf.

We examine its binary size and static RAM usage (i.e., data and BSS segment size).

Kratos and *Multipass* are in-house research operating system product lines tailored towards heavily resource-constrained embedded systems. They can be configured to support various microcontroller platforms and common peripherals. We model text and data/BSS segment size.

MxKernel extends Kratos' approach to the domain of heterogeneous many-core systems. Its primary focus is on scalability and speed in high-performance database applications. Again, we are interested in text and data/BSS segment size.

The *resKIL* agricultural AI product line aims to support developers in achieving optimal AI performance on low-power embedded devices, with variable hardware platform, AI platform, AI architecture, and runtime settings. Here, we examine inference accuracy, latency, throughput, and memory usage.

Finally, *x264* is a well-known open source library and tool for encoding video streams in the H.264/MPEG+4-AVC codec. It provides many tunables that affect the trade-off between video quality, encoder speed, and output file size. We measure encoding duration and output file size.

We note that, in contrast to the majority of related work, we configure both boolean feature toggles and scalar configuration options such as buffer sizes or encoder settings.

4.2 Evaluation setup

All evaluated product lines use the Kconfig language for variability modeling. We used our automatic NFP model generation toolchain[2] and Kconfig-frontends version 4.11.0 to create random configurations for each product line, store configurations and corresponding NFP measurements, and generate different kinds of NFP models.

Depending on the number of features, we generated and benchmarked between 1,000 and 30,000 different configurations, and use these for NFP model generation.

First, we extract a feature vector $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{N}$ from each Kconfig configuration. We map disabled boolean features and scalars that cannot be configured due to unsatisfied dependencies

[2]Available online: https://ess.cs.uos.de/git/software/dfatool

Property	FW [%]	CART [%]	RMT [%]
busybox size	1.6	0.3	0.3
busybox RAM	42.1	0.3	0.3
Kratos ROM	7.8	0.5	0.5
Kratos RAM	36.5	0.7	0.8
Multipass ROM	39.9	6.5	2.0
Multipass RAM	29.6	3.7	2.6
MxKernel ROM	0.5	0.5	0.9
MxKernel RAM	0.0	0.0	2.4
resKIL accuracy	16.5	12.4	14.8
resKIL latency	96.8	38.2	97.6
resKIL throughput	114.5	9.4	112.2
resKIL memory	81.1	21.1	93.7
x264 time	39.0	10.3	16.9
x264 size	96.8	2.0	57.9

Table 1: Symmetric mean absolute percentage error of feature-wise annotation (FW) and regression tree (CART, RMT) models for NFPs of the evaluated product lines.

to 0, enabled boolean features to 1, and configured scalar features to their scalar value. We leave out string features.

In order to assess the accuracy of *integrated* feature-wise annotation models that disregard feature interactions for the sake of manageability, we use least squares regression to fit the formula $\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$ to our observations. This formula associates each feature x_i with a static annotation β_i that expresses how enabling or (for scalar features) changing it affects the modeled NFP.

To assess *separate* models, we generate CART and RMT models using the corresponding model generation algorithm. As tree-based models have been shown to perform well when modeling non-functional properties of software product lines, we consider these to be suitable examples for this model variant[9]. Fig. 2 shows an excerpt of a classification and regression tree for the busybox product line. In all cases, we determine the generalization error using 10-fold cross-validation.

4.3 Results

Table 1 shows the symmetric mean absolute percentage error (SMAPE) for integrated feature-wise annotation (FW) and separate NFP models (CART, RMT). Given predictions $P = \{p_1, \ldots, p_n\}$ and ground truth $Y = \{y_1, \ldots, y_n\}$, it is defined as follows.

$$\mathrm{SMAPE}(P, Y) = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|p_i - y_i|}{\frac{|p_i|+|y_i|}{2}}$$

First, our results confirm that separate models are better than integrated feature annotations in all cases, with a two to ten times lower model error in most cases – apart from MxKernel, where separate and integrated models are equally good.

We also see that the flexibility offered by separate models pays off. CART are well-suited for high-dimensional variability models using mostly boolean features, whereas RMT tend to perform better when faced with low-dimensional, scalar-heavy configuration spaces. Consequently, CART exhibit the lowest error for the resKIL product line, whereas RMT models are most accurate for Multipass.

With integrated models, system designers would need to choose a suitable NFP model right from the start, whereas separate models allow changing the model type and re-training it at any time.

To assess whether modeling pairwise or more complex feature interactions would improve the accuracy of the integrated model, we examine the classification and regression tree model for busybox RAM usage (see Fig. 2 for an excerpt). The tree has been automatically generated by the CART algorithm, which means that the closer a feature is to the root of the decision tree, the higher its influence on the modeled NFP.

Top nodes include *runtime sanitizers*, *debug build, disable compiler optimizations*, and the *buffer allocation policy*. All of these are cross-cutting concerns that have little effect by themselves, but interact with nearly all other busybox features.

An integrated model would need to express this interaction in hundreds of busybox features, making it complex and repetitive. Although this comes with increased accuracy – for example, when only considering non-debug builds with dynamically allocated buffers, a feature-wise annotation model achieves a generalization error of just 3.6 % – once such complexity is needed, one might as well use separate NFP models.

5 CONCLUSIONS

We have started this paper with the question whether a variability modeling language should incorporate elements for handling NFP-related concerns. The clear answer is *no*. Decades of software engineering experience teach us the principle of *separation of concerns*. We should obey it regarding this particular question as well. NFPs depend completely on the implementation, i.e. the *solution space* of a software product line, while variability modeling is typically done in the *problem space*. Mixing both complicates the development process. With clean separation, variability model and NFP model can be chosen and changed independently.

We believe that the integration of NFP data into variability models was at the beginning merely a matter of practical use cases. Configuration tools should be able to display NFP predictions – ideally based on configurable features. However, over time it turned out that such simple annotations are inaccurate and that more complex models are needed. It makes no sense to follow this path until users are no longer able to interpret the annotated formulae.

Instead, separate NFP models can provide accurate predictions considering arbitrary feature interactions on the implementation level. The configuration tool can link the variability model and the NFP model, using the feature vector as the interface between both sides. In recent months we have implemented such a configuration tool, which combines Kconfig variability models with CART- and RMT-based NFP models. It can answer "what if" questions, e.g. "what is the expected frame rate if I toggle this boolean feature?", or help with more complex tasks such as feature recommendations considering multi-objective optimization.

ACKNOWLEDGMENTS

This work has been supported by funds of the Federal Ministry of Food and Agriculture (BMEL) based on a decision of the Parliament of the Federal Republic of Germany via the Federal Office for Agriculture and Food (BLE) under the innovation support programme, and by Deutsche Forschungsgemeinschaft under grant no. SP 968/9-2.

REFERENCES

[1] Andreas Abele, Rolf Johansson, Henrik Lönn, Yiannis Papadopoulos, Mark-Oliver Reiser, David Servat, Martin Törngren, and Matthias Weber. 2010. The CVM framework : A prototype tool for compositional variability management. In *Proceeding of : Fourth International Workshop on Variability Modelling of Software-Intensive Systems*. Technische Universität Berlin, DE, 101–105. http://www.sse.uni-due.de/vamos/proceedings/VaMoS_2010_Proceedings.pdf

[2] Mauricio Alférez, Mathieu Acher, José A. Galindo, Benoit Baudry, and David Benavides. 2019. Modeling Variability in the Video Domain: Language and Experience Report. *Software Quality Journal* 27, 1 (mar 2019), 307–347. https://doi.org/10.1007/s11219-017-9400-8

[3] Michał Antkiewicz, Kacper Bąk, Alexandr Murashkin, Rafael Olaechea, Jia Hui (Jimmy) Liang, and Krzysztof Czarnecki. 2013. Clafer Tools for Product Line Engineering. In *Proceedings of the 17th International Software Product Line Conference Co-Located Workshops* (Tokyo, Japan) *(SPLC '13 Workshops)*. Association for Computing Machinery, New York, NY, USA, 130–135. https://doi.org/10.1145/2499777.2499779

[4] Kacper Bąk, Krzysztof Czarnecki, and Andrzej Wąsowski. 2010. Feature and meta-models in clafer: Mixed, specialized, and coupled. In *International Conference on Software Language Engineering*. Springer, 102–122.

[5] Kacper Bąk, Zinovy Diskin, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. 2016. Clafer: unifying class and feature modeling. *Software & Systems Modeling* 15, 3 (2016), 811–845.

[6] Danilo Beuche. 2008. Modeling and Building Software Product Lines with Pure::Variants. In *SPLC '08: Proceedings of the 2008 12th International Software Product Line Conference*. IEEE Computer Society, Washington, DC, USA, 358. https://doi.org/10.1109/SPLC.2008.53

[7] Quentin Boucher, Andreas Classen, Paul Faber, and Patrick Heymans. 2010. Introducing TVL, a text-based feature modelling language. In *Proceedings of the Fourth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10)*. 159–162.

[8] Birte Friesel and Olaf Spinczyk. 2022. Regression Model Trees: Compact Energy Models for Complex IoT Devices. In *Proceedings of the Workshop on Benchmarking Cyber-Physical Systems and Internet of Things (CPS-IoTBench '22)*. IEEE Press, 6 pages. https://doi.org/10.1109/CPS-IoTBench56135.2022.00007

[9] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. 2018. Data-Efficient Performance Learning for Configurable Systems. *Empirical Softw. Engg.* 23, 3 (June 2018), 1826–1867. https://doi.org/10.1007/s10664-017-9573-6

[10] Rafael Olaechea, Steven Stewart, Krzysztof Czarnecki, and Derek Rayside. 2012. Modelling and Multi-Objective Optimization of Quality Attributes in Variability-Rich Software. In *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages (NF-PinDSML '12)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2420942.2420944

[11] Marko Rosenmüller, Norbert Siegmund, Thomas Thüm, and Gunter Saake. 2011. Multi-Dimensional Variability Modeling. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS '11)*. Association for Computing Machinery, New York, NY, USA, 11–20. https://doi.org/10.1145/1944892.1944894

[12] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting performance via automated feature-interaction detection. In *2012 34th International Conference on Software Engineering (ICSE)*. 167–177. https://doi.org/10.1109/ICSE.2012.6227196

[13] Norbert Siegmund, Marko Rosenmüller, Christian Kästner, Paolo G. Giarrusso, Sven Apel, and Sergiy S. Kolesnikov. 2011. Scalable Prediction of Non-functional Properties in Software Product Lines. In *2011 15th International Software Product Line Conference*. 160–169. https://doi.org/10.1109/SPLC.2011.20

[14] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. 2012. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal* 20, 3-4 (2012), 487–517. https://doi.org/10.1007/s11219-011-9152-9

[15] Chico Sundermann, Kevin Feichtinger, Dominik Engelhardt, Rick Rabiser, and Thomas Thüm. 2021. Yet Another Textual Variability Language? A Community Effort towards a Unified Language. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume A*. Association for Computing Machinery, New York, NY, USA, 136–147. https://doi.org/10.1145/3461001.3471145

[16] Xhevahire Tërnava, Luc Lesoil, Georges Aaron Randrianaina, Djamel Eddine Khelladi, and Mathieu Acher. 2022. On the Interaction of Feature Toggles. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems* (Florence, Italy) *(VaMoS '22)*. Association for Computing Machinery, New York, NY, USA, Article 9, 5 pages. https://doi.org/10.1145/3510466.3510485