Towards System Analysis with Variability Model Metrics

Thorsten Berger
University of Waterloo
Waterloo, ON, Canada
tberger@gsd.uwaterloo.ca

Jianmei Guo
University of Waterloo
Waterloo, ON, Canada
gjm@gsd.uwaterloo.ca

ABSTRACT

Variability models are central artifacts in highly configurable systems. They aim at planning, developing, and configuring systems by describing configuration knowledge at different levels of formality. The existence of large models using a variety of modeling concepts in heterogeneous languages with intricate semantics calls for a unified measuring approach. In this position paper, we attempt to take a first step towards such a measurement. We discuss perspectives of metrics, define low-level measurement goals, and conceive and implement metrics based on variability modeling concepts found in real-world languages and models. An evaluation of these metrics with real-world models and codebases provides insight into the benefits of such metrics for the defined perspectives.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.8 [**Software Engineering**]: Metrics; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Measurement, Languages, Design

Keywords

software product lines, variability modeling, feature modeling, metrics, empirical software engineering

1. INTRODUCTION

Variability models, such as feature or decision models, are prominent means to cope with variability in highly configurable systems. When carefully managed, variability models support the organization of design knowledge, facilitate interactive system configuration and variant derivation, and foster an overview understanding of large codebases. Therefore, variability models need to comply with certain properties. They need to accurately describe the variability of a system, for instance, they must be consistent with the codebase to prevent misconfigurations of systems. They should also adhere to quality attributes—models should be easy to understand, to configure, and to maintain.

VaMoS '14, January 22 - 24 2014, Sophia Antipolis, France

http://dx.doi.org/10.1145/2556624.2556641.

Analyses ensure such properties, but are commonly specific to variability modeling languages. In practice, we see a large variety of free and commercial languages and tools [9, 7, 15]. Many share concepts with feature and decision models [12, 18], but often come with additional modeling concepts (e.g., derived features, visibility conditions, computed defaults), domain-specific syntaxes, and intricate semantics [12, 20, 36, 10]. Making such models accessible to analyses, or conducting cross-language analyses, often requires detailed understanding of the underlying languages. A solution could be to implement semantics-preserving transformations between languages, or to a standard pivot language for variability modeling. However, current empirical results question any convergence of languages in the near future [9, 8, 23, 22]; and providing semantics-preserving transformations across languages is difficult and often infeasible.

We take the position that a certain class of analyses suffices to rely on metrics that expose core characteristics of variability models. We conjecture that such analyses comprise the prediction of quality attributes of systems, such as the maintainability and cognitive complexity of models (and, thus, of the configuration process), or the prediction of characteristics of the source code with respect to codebase sizes, granularity of extensions, or scattering degrees of features.

Only a few works have discussed metrics over variability models, perhaps due to the fact that realistic models are rarely available to research. These existing metrics are limited to very basic feature modeling concepts [5], or specific to class [19] and architecture modeling languages [38]. Except [5], they have not been evaluated in a realistic scenario. With the availability of large open source systems that have variability models [12], or feature model repositories, such as S.P.L.O.T. [30], a significant corpus of empirical data is available to define and evaluate metrics. Specifically, we believe that the definition of metrics should be driven by modeling concepts from languages that appear in practice, and by the use of these concepts in real models.

In this position paper, we report very early results on exploratory research on variability model metrics, primarily to raise discussion and to provide a framework for future research. We discuss perspectives of variability model metrics, we introduce a set of low-level, reliable metrics to characterize models, implement them, and evaluate them using a correlation analysis. We also provide preliminary results on

a correlation analysis with a set of source code metrics.

The initial set of metrics we propose has the goal to measure size and shape (e.g., number of leaf features), feature representations (e.g., ratio of numerical features), constraints (e.g., ratio of configuration constraints to visibility conditions), and dependencies (e.g., connectivity ratio of the dependency graph spanned by feature constraints). This set likely needs to be complemented with further metrics targeting different goals of measurement (e.g., computational complexity by measuring expression operators), or concepts of further variability modeling languages. Our preliminary evaluation should also be extended with further subjects.

2. BACKGROUND

In this work, we focus on the two languages Kconfig and CDL. We previously analyzed the languages and all their instances available in open source systems software projects [12], and build on these insights in the present work. To the best of our knowledge, the systems using these languages constitute the only freely available corpus of models and code, which allows us to study models and code in their whole.

2.1 Variability Modeling Concepts

The languages Kconfig and CDL share core feature modeling concepts, such as Boolean features, a hierarchy, feature groups (OR, XOR, MUTEX), and cross-tree constraints. They also have advanced concepts rarely considered in academic modeling approaches, such as derived features, default values (also computed), visibility conditions, and expressive constraint languages. Kconfig also has three-valued logic to control binding modes of features. We will briefly explain the concepts when introducing the metrics in Sect. 4.

Other common variability modeling languages have similar concepts, and often also go beyond FODA feature models [25]. Examples comprise extended (attributed) feature models [7], the TVL [16] language, or the languages of the commercial tools pure::variants and GEARS. All share concepts with feature models, but provide additional modeling concepts, such as non-Boolean value domains and expressive constraint languages. Furthermore, decision-model-oriented approaches, such as DOPLER [21], are known to be close to feature models [18], have extended concepts such as visibility conditions and default values, but have different semantics when it comes to the model hierarchy.

We believe that the rich modeling concepts found in our subject languages are "theoretically" representative (cf., theoretic sampling [24]) of a larger number of languages. However, showing applicability of our metrics to other languages requires further analyses. Moreover, extending their applicability to more heterogeneous languages, such as the textual languages discussed by Eichelberger et al. [23], would be interesting future work.

2.2 Software Metrics

The typical approach to developing metrics follows the GQM strategy [6], which requires defining i) a concrete *goal* of measurement, ii) *inquiry questions*, and iii) *metrics* that collect supporting data for these questions. Our *goal* in this paper is to measure structural properties comprising core characteristics of models: size and shape, feature representations, constraints, and dependencies. We follow a more lightweight approach by not formally defining *inquiry questions*, but conceiving metrics that measure characteristics of the variability modeling concepts we find in the languages. We believe this approach is justified, since aims of this phase of our work are to provide cross-language model measurements, and to explore the potential of using a set of simple metrics in analyses—by investigating correlations and their potential for predictions. This approach has been used by others, such as Calero et al. [14] and Bagheri et al. [5]

Our proposed metrics target low-level structural characteristics. Thus, they are reliable measurements, but not necessarily able to measure more complex properties, such as maintainability. The latter are usually compound metrics relying on several lower-level metrics. Development of valid [26] measurements for such complex properties is a natural next step, but out of scope of this position paper.

3. PERSPECTIVES

In the following, we describe perspectives for applying variability model metrics. These range from very concrete perspectives (P1–P3), to which we contribute in this paper, to rather abstract long-term visions (P4, P5), requiring further empirical studies and data.

P1. Quantifying model properties. The most apparent use case of model metrics is to share and compare properties of models. In fact, we believe that it is difficult for other researchers to exploit our extracted Kconfig and CDL models, as our analysis infrastructure requires in-depth knowledge of the abstract syntax of both languages. Making these models available to research by reducing their content to easily understandable metrics is an objective of our work. Furthermore, a set of standardized metrics can be used to make valuable characteristics of commercial models available, without exposing the models themselves.

P2. Model Comprehension. Real-world models can be large and can have complex—often textual–syntax [12]. Understanding them consumes much time in system maintenance. Furthermore, understanding system design goals and developers' intents is important to improve the quality and efficiency of maintenance. With a set of metrics reflecting different aspects of variability modeling and design intents, we can identify some metrics with statistical significance and filter out irrelevant ones by various quantitative techniques, such as correlation analysis and hypothesis testing. These identified metrics infer specific characteristics of systems that may be preferred or avoided deliberately by developers.

P3. Understanding the relationship between models and code. Our aim is to understand the relationship between models and code, in order to support variability model reverse engineering techniques. We have worked on exact approaches to statically extract feature constraints from source code [37]. In the present work, we explore a broader range of different characteristics by correlation analysis, which potentially provides additional insights into real-world systems. Insights of such a correlation analysis can be useful to improve model reverse-engineering approaches [37, 4, 3], in particular to support disambiguating the feature hierarchy—a common challenge for such techniques.

P4. Predicting system quality attributes. We hypothesize that certain analyses of highly configurable systems can be reduced to a quantitative analysis of a dedicated set of variability model metrics. Such metrics can provide the basis for experiments, to eventually predict quality attributes, such as maintainability, usability, and evolvability of a system. Appropriate metrics can potentially provide early indicators

of specific characteristics of systems, such as computational complexity of a reasoner that provides choice propagation and conflict resolution, and of the cognitive complexity of an interactive configuration process for a user.

P5. Supporting code analyses. Beyond providing insights into the relationship between models and code, metrics might be able to support code analyses. This conjecture is based on the observation that models and code co-evolve [29, 33]. Thus, several characteristics might correlate. For example, constraint-related metrics might indicate complexity of source code. Finally, we believe that analyses upon metrics can be more efficiently calculated, and might provide early indicators for results that otherwise require expensive source code analyses. Whether model metrics could even be used to predict bugs remains an interesting future research question.

4. METRICS

To address perspective **P1**, we propose metrics for our four measurements goals: i) size and shape of the model, ii) the representation of features, iii) the types and modality of feature constraints, and iv) dependencies between features. Names of metrics are aligned with existing work ([5], see Sect. 7) if applicable. We implemented all metrics in a tool [2].

4.1 Structural Metrics

Structural metrics concern the size and the shape of variability models and should unleash core characteristics of the feature hierarchy. The latter influences and restricts possible visualizations, and the user-perceived complexity. For example, moderate hierarchy depths of models are preferred in practice [8], and the number of leaf features impedes understandability as indicated by a user study [5].

Primary size and shape measures comprise the number of features **NF**, the number of top-level features **NTop**, the number of leaf features **NLeaf** (features without children), and the number of grouping features **NGF** (features with at least one child). The hierarchy can be characterized by the leaf depth **LD**, where we consider the mean, the median, and the maximum of the distribution of hierarchical depths of leaf features. The branching factors **BF** (number of children per feature) influence the width of the tree. We consider the mean, the median, and the maximum of the distribution of branching factors across all non-leaf features.

Feature groups are among the most common concepts in feature models. We observe three kinds of feature groups in our models, imposing OR, XOR, and MUTEX constraints among the grouped features. Three simple metrics capture their respective quantities: **NOr**, **NXor**, and **NMutex**. Although arbitrary cardinality constraints across features are possible in CDL, they do not appear in any model, and we refrain from defining a metric.

4.2 Feature Representation Metrics

Features that admit values beyond Boolean values are commonly used in practice, and supported by major variability modeling tools and languages. In fact, even the original FODA report [25] admitted non-Boolean attributes associated with features. The use of non-Boolean features indicates a technical system domain [12], and their use often requires constraints beyond propositional logic, which challenges model reasoners. We propose the following metrics to characterize the proportions of different types of features: **RSwitch** determines the ratio of Boolean (switch) features, **RData** the ratio of features with an arbitrary value domain, with the sub-metrics **RDataNum** (numbers, including integer, hex, and float types) and **RDataString** (arbitrary strings). Calculating the latter two depends on explicit type support in the variability modeling language, or on type inference mechanisms in dynamically typed languages, such as CDL. Furthermore, as restrictions of value domains (e.g., enumerations or ranges) can support reasoners, we introduce **RDataOpen**, which quantifies the ratio of data features with open value domains among all data features.

Features that have no value assigned or belong to the commonality of the model (often referred to as mandatory or abstract features), are captured by **RNone**, which measures the ratio of such features.

Finally, we observe another kind of features that represent abstract capabilities [12], for example, that the system has a filesystem, or that networking support is provided. These capability features are provided by other features, and are the dependency target of features that require such functionality. Thus, capability features reduce coupling within the model and impact dependency structures. The metric **RCap** measures their ratio in a model.

4.3 Feature Constraint Metrics

Constraints are the primary source of complexity in variability models. They challenge reasoners and users when configuring a system. A significant amount of constraints implicitly exists within the hierarchy (child-parent implications). In the following, we introduce metrics that address the quantity, types, expressiveness, and modality of constraints explicitly declared per feature.

We use our previous classification of feature constraints [12], which covers constraint concepts found in feature models, Kconfig, and CDL. These concepts comprise *configuration constraints*, which restrict values and combinations of features; *visibility constraints*, which blind or gray out feature sub-trees in the configurator; *defaults*, which provide default values for features, either as literals or as expressions; and *derived features*, whose values are determined by an expression or a literal, and are not directly modifiable by a user.

Our first metric **RConstr** determines the ratio of features that explicitly declare any such constraint. With **RPurelyBoolConstr**, we measure the proportion of purely Boolean constraints among all declared constraints, which indicates to what extent propositional-logic-based reasoners are applicable on a model. **RDerived** is the ratio of derived features, with the more specific metrics **RDerivedExpr** (values derived with an expression) and **RDerivedLit** (values set by a constant literal). **RVisibility** is the ratio of features with at least one visibility condition. Note that visibility conditions are always expressions. **RDefault** measures the ratio of features with an explicitly modeled default value, with **RDefaultExpr** as the ratio of expressions, and **RDefaultLit** as the ratio of literals used for such defaults.

Finally, we conjecture that the proportion between visibility conditions and configuration constraints influences the configuration process. Configuration constraints can be temporarily violated and, if supported by the configurator, automatically resolved using a reasoner. In contrast, visibility conditions are applied immediately after each decision a user makes, resulting in complete sub-trees becoming invisible or grayed out. It is still unknown when to use one kind of constraint over the other. Thus, we define **RConfVis**

as the ratio between configuration constraints and visibility conditions. Its evaluation with respect to its impact on the configuration process likely requires a controlled user experiment, however.

4.4 Dependency Metrics

While the previous metrics indicate the extent to what a model is constrained, core characteristics of the resulting dependency graph further indicate the computational and cognitive complexity of a model. Our abstraction here is a *dependency* as the reference of another feature (that is not a direct parent) in a constraint.

First, we adapt the **CTCR** (cross-tree constraint ratio) metric from [31], which is the proportion of features that participate in cross-tree constraints. It captures features that either have at least one dependency, or are the target of the dependency of another feature. Further consider the dependency graph spanned by feature constraints (disregarding hierarchy). We introduce **RCon** as the ratio of connectivity of this graph—more precisely, the percentage of features with a dependency. We also use **RDen** as the density of the graph, defined as the average number of features referenced in constraints per feature.

4.5 Prospective Metrics

We briefly discuss further candidate metrics that might be relevant to predict quality attributes.

Hierarchy specifics. We have observed that both CDL and Kconfig allow to separate the syntactic nesting of features in the models from the hierarchy shown in the configurators [12]. This deviance of both hierarchies might be relavant to maintenance and understandability of models, thus, a metric could measure it. Furthermore, Kconfig allows to violate hierarchy rules of feature models—features do not have to imply their parent and can even exclude it. This might likewise impact quality attributes and should be measured.

Feature descriptions. Textual feature descriptions, contained in all our subject models, are a rich source of information. Applying text metrics, as we did before [37], shows that these provide in fact useful information, for instance, about ontological (hierarchical) relationships between features.

Feature-to-code mapping. The feature-to-code mapping, often implicitly hidden in imperative build logic [11, 32], can provide insights into the granularity and scattering of variability in the codebase. Thus, metrics about this mapping, such as in the form of explicitly extracted file presence conditions [11], could be a valuable addition.

5. CORRELATION ANALYSIS

To address **P2** and **P3**, we conduct a correlation analysis for our model metrics, and for a set of existing code metrics.

5.1 Experimental Setup

We choose subject systems that have both a variability model and source code that we can analyze. As listed in Table 1, we analyze eight highly configurable systems with different sizes of the models (114 to 8355 features) and their primarily C-based codebases (26K–10.2M LOC). All are successful open source projects from the systems software domain, having both a model and a significant codebase. A superset of these was subject to our previous language and model analysis [12]; however, as not all had a proper codebase, we left out those projects that use the variability model to manage packages of third-party software.

Table 1: Subject systems

model	version	features	LOC
ToyBox	0.1.0	191	26K
axTLS	1.4.9	114	21.4K
Fiasco	2013091917	213	140K
BusyBox	1.21.0	921	195K
eCos i386PC	3.0	1256	301K
uClibc	0.9.31	367	320K
CoreBoot	4.0-nov2013.git	4118	1.5M
Linux x86	3.4	8355	10.2M

The code metrics we explore are taken from Liebig et al. [27], and summarized in Table 2. They capture characteristics of code with respect to preprocessor-driven variability (conditional compilation). We use cppstats [1] to gather them.

We hypothesize that a significant correlation between two model metrics (perspective **P2**) or a model and a code metric (perspective **P3**) can be found (H_1). Correspondingly, the null hypothesis (H_0) is that there is no correlation between such. We choose a correlation test based on insights from Bagheri et al. [5] and Courtney et al. [17]. The former observe that their structural metrics did not have a normal distribution. The latter report that the linear correlation coefficient (e.g., Pearson's correlation coefficient) is unreliable to disclose significant association if the sample size is relatively low compared to the number of the evaluated variables. Thus, we adopt a non-parametric measure, *Spearman's rank correlation coefficient*, to indicate the statistical correlation between two metrics. As usual, the significance level is 0.05.

To help understand the correlation coefficient intuitively, we use the following scale provided by Salkind [35]:

- 0.8 to 1.0 or -0.8 to -1.0: very strong relationship;
- 0.6 to 0.8 or -0.6 to -0.8: strong relationship;
- 0.4 to 0.6 or -0.4 to -0.6: moderate relationship;
- 0.2 to 0.4 or -0.2 to -0.4: weak relationship;
- 0.0 to 0.2 or 0.0 to -0.2: weak or no relationship.

Our analysis is quantitative and qualitative. We first filter out correlation coefficient estimates without statistical significance (p-value$\geq$0.05). Given a small sample size, we then qualitatively inspect strong correlations and try to find explanations based on our knowledge of the languages, models, and configurator tools. In the following, we report such meaningful correlations and provide careful interpretations.

5.2 Results and Discussion

We provide detailed results in Tables 3 and 4, and on the website of our tool [2] (metric values, coefficients, p-values).

5.2.1 Model Metric Correlations

For **P2**, our analysis shows that correlations among model metrics exist. Thus, not all of them are necessary to reveal the characteristics of a model. Some can be used interchangeably. Table 3 shows all coefficients (above diagonal) and p-values (below diagonal) of strong correlations.

Model size and shape. We see a very strong relationship between the number of features (**NF**) and the number of top-level (**NTop**) and leaf features (**NLeaf**), which indicates that functionality is added at both levels when models grow. Together with the observation that the average leaf

depth (**LD**) is strongly negatively correlated with the average branching factor, we see that developers avoid deep trees.

We see a strong relationship between the size metrics (**NF**, **NTop**, **NLeaf**) and the maximum branching (**BF**). Not surprisingly, the presence of abstract (mandatory) features (**RNone**) strongly negatively correlates with the maximum branching factor (**BF**), while the presence of XOR groups is strongly correlated to branching.

Interestingly, the mean and medium measures of the **BF** metric are not correlated with each other. This indicates many outliers; thus, median is likely a more stable measure for branching. Notably, we made the observation about the high variance and many outliers in the branching factors before [12]. Developing well-balanced trees is obviously not a top priority for developers of our subject models.

Feature representation. We observe that higher numbers of numerical features correlate with constraints that use expressions instead of literals. Numerical features (**RDataNum**) strongly correlate with string features (**RDataString**) and we do not see any tendency towards one or the other kind in our subjects. Furthermore, data features (**RData**) are rarely value-domain-restricted (**RDataOpen**) in our models.

We can also see that an increase of abstract (**RNone**) features limits the maximum branching (**BF**) due to a strong negative correlation. Higher proportions of these features indicate more manual effort that went into structuring the model (domain modeling), which naturally limits branching, perhaps intentionally as modelers prefer well-balanced trees.

One limitation of our analysis is that we cannot reliably identify capability features (**RCap**) in Kconfig models. However, we conjecture that their presence improves modularity and reduces coupling—a property that likely is reflected in the codebase. We leave this investigation as future work.

Feature constraints. We observe that the cross-tree constraint ratio (**CTCR**) is strongly correlated with the median branching factor (**BF**), and strongly negatively correlated with the maximum depth of the feature hierarchy (**LD**). In fact, wider and less deep trees, have less opportunities to encode dependencies in the hierarchy, thus, more cross-tree constraints need to be declared.

The metrics **CTCR**, and the connectivity (**RCon**) and density (**RDen**) of the dependency graph (excluding hierarchy) all measure cross-tree constraints and turn out to be very strongly correlated. Although this finding requires further investigation to provide any deeper insight into dependency structures, it might be an indicator of relatively regular, not-skewed structures. However, we did not further analyze the density of the graph, although we believe that checking for specific distributions that are known from network theory, such as the power law, could help to identify "hubs", which have many dependencies or that many features depend on. The latter, in fact, could help to identify capabilities. We leave construction of a corresponding compound metric for future work, and hypothesize that it might be useful to predict cohesion and coupling in a system.

We also find some natural correlations that show that the higher proportions of abstract (**RNone**) features exist, the less constraints are defined in the model. But interestingly, **RNone** is strongly correlated with default values, indicating that when more effort goes into structuring a model, the more explicit default values (**RDefault**) are defined.

Finally, we propose further analysis of the effects of the ratio between visibility conditions and configuration constraints (**RConfVis**) on the configuration process. Dependencies can be realized using configuration constraints, which can be temporarily disabled (and resolved using an inference engine), or with visibility conditions, which are verified immediately and cannot be disabled. We find these differences not only in our subjects, but also in the commercial tool pure::variants. It is still not clear what the right balance between the two modalities of feature constraints should be.

Table 2: Code metrics (from [27])

metric	description
LOC	lines of code
NOFC	number of feature constants (features referenced in source code)
LOFC	lines of feature code
ND	average (**AND**) and maximal (**NDMAX**) nesting of conditional compilation directives (#IF*)
SD	scattering degree: average and standard deviation of the number of occurrences of features in different expressions of conditional compilation directives
TD	tangling degree: average and standard deviation of the number of features in expressions of conditional compilation directives
GRAN	number of #IFDEFS occurring at a certain kind of language granularity: global level (**GRANGL**), function or type level (**GRANFL**), block level (**GRANBL**), statement level (**GRANSL**), expression level (**GRANEL**), method signature level (**GRANML**)
TYPE	number of extensions under equivalent #IF* expressions: homogeneous extensions with duplicated code (**HOM**), heterogeneous extensions with varying code (**HET**), and mixed (**HOHE**).

5.2.2 Model and Code Metric Correlations

As a contribution to **P3**, we discuss model and code metric correlations. Table 4 shows coefficients (p-values in [2]).

We observe very strong correlations between model size metrics (**NF**, **NTop**, and **NLeaf**) and code size metrics (**LOC**, **NOFC**, and **LOF**), which also implies a potential application of model metrics for code analysis and, thus, evidences the conjectured perspective **P5**. Moreover, model size metrics have very strong correlations with code granularity metrics (**GRANGL**, **GRANFL**, **GRANBL**, **GRANSL**, **GRANEL**, **GRANML**, and **GRANERR**), which indicates that our subject systems in the domain of system software are similar with respect to the granularity of their variability. Since code size metrics are the most commonly used metrics in code analysis, and code granularity metrics have been used to analyze the architecture of variability-aware systems [27], we can carefully expect a bright prospect of model metrics used in these application areas.

Our correlation analysis helps us to filter out irrelevant code metrics and to analyze the specific characteristics of systems. For example, we observe that the model size metrics have a very strong correlation with two of the code extension metrics (**HOM** and **HOHE**), but have no correlation with the third one (**HET**). That inspired us that only the two types of extensions are the most common and grow proportionally to the system size.

Likewise, our correlation analysis identifies some important model metrics that reflect the specific characteristics of systems. For example, we observe that the code size metrics are strongly correlated with the number of XOR feature groups, but not with the number of OR or MUTEX feature groups. That means most of feature groups in our subject systems

are XOR groups and their size grows stably with the increase of system size, which also confirms a previous finding [12].

However, we may still need more model metrics or code metrics to extend the insight into systems. For example, in our experiment, we do not see any correlation between the maximum depth of a model and the maximum nesting of #IF* directives. Moreover, we did not measure the code complexity, for instance using McCabe's cyclomatic complexity; correspondingly, we may need to propose some compound kind of model complexity metrics, which will be done in future work.

6. THREATS TO VALIDITY

Internal validity. To enhance internal validity, we collected all metrics automatically in terms of their prescribed definitions. Moreover, we conducted the correlation analysis following a standard calculation process of Spearman's rank correlation coefficient and chose only the statistically significant (p-value$<$0.05) results for analysis. However, we cannot guarantee that the analysis results of our experiment depend on specific definitions of the evaluated metrics.

We are aware that our sample size (eight systems) is not sufficient and may give rise to unreliable statistics [17]. We mitigate this threat by choosing a non-parametric measure (as suggested by [17]), and by a qualitative inspection of strong (and significant) correlations based on our knowledge of the languages.

External validity. To increase external validity, we chose all real-world systems with models and analyzable codebases we are currently aware of. These systems have different sizes and use two different variability modeling languages. Moreover, we propose a set of variability model metrics from different perspectives, and collected all variability-related code metrics from existing literature [27]. However, we are aware that the results of our experiment are not automatically transferable to all other systems, since our metrics might not comprehensively take specific modeling concepts of other languages into account. On the other hand, many other languages share concepts with feature modeling and, thus, with our subject languages. Our experience with other tools, such as pure::variants, also shows that these have similar advanced concepts (e.g., configuration and visibility constraints) as our subjects—although a thorough analysis would be required for sufficient confirmation.

7. RELATED WORK

Metrics for variability models have been defined before. Bagheri et al. [5] propose ten structural metrics on FODA feature models and evaluate their ability to predict the maintainability (classified into analyzability, changeability, and understandability) of a model. They gather these quality attributes in a user experiment with S.P.L.O.T. models [30], and analyze correlations with the model metrics. The authors also build prediction models for the quality attributes. A difference to our work is that we investigate the relationship between models and code, while Bagheri et al.'s focus is on quality attributes as perceived by users. We also consider real-world variability modeling concepts that are beyond feature models. In fact, frequent occurrence of those modeling concepts indicates a need for metrics that cover them.

De Oliveira et al. [19] introduce metrics on UML class diagrams that use specific stereotypes to describe product line variability. They illustrate correlations of their metrics to quality attributes (complexity, maintainability, and testability) on an example, and discuss use cases of these metrics for product line architectural assessments. They emphasize the benefit of metrics to support design decisions.

Zhang et al. [38] define metrics for the product-line-specific architecture definition language vADL. The metrics measure similarity, variability, reusability, and complexity of a product line. The authors emphasize the capability of their metrics to predict quality attributes of a whole system. In contrast to our work, they define complex compound metrics, which is out of scope of our position paper and future work. Unfortunately, their metrics are not evaluated on real systems.

Code metrics related to variability have been defined in various works. We used the code metrics defined by Liebig et al. [27] from their empirical work about the low-level characteristics of variability in 40 highly configurable systems that use preprocessor #IF* directives. Lopez-Herrejon et al. [28] adapt McCabe's cyclomatic complexity metric to measure the complexity of highly configurable systems. They evaluate the metric on an academic system (KWIC) and conjecture that defining metrics specific to highly configurable systems can leverage existing work and results on software metrics. Our present work is also motivated by this conjecture.

We previously studied the languages CDL and Kconfig, and all models available in these languages in depth [12, 36, 10]. We also related characteristics of these languages and models to the feature modeling language and the feature models in the S.P.L.O.T. repository, showing that our models significantly deviate from the academic models in S.P.L.O.T.. In the present work, we properly define perspectives, low-level measurement goals and metrics, and implement them in a tool. The correlation analysis, both among model metrics, and between model and code metrics, is also new.

8. CONCLUSION

In this position paper, we have discussed the potentials of variability model metrics to system analysis; introduced a set of low-level variability model metrics; implemented them in a tool; conducted a correlation analysis among these metrics; and explored their relationship to code metrics in real-world systems. The tool and detailed results are available online [2].

An immediate benefit of these metrics is that they allow to compare models across languages (P1). They also make core properties of our models available to the research community for further analysis, without requiring in-depth understanding of the underlying complex languages.

Our preliminary correlation analysis so far shows that we can identify significant and meaningful correlations between model metrics (P2), gather interesting insights about model design (P2), and identify correlations among model and code metrics (P3). However, while size metrics are highly correlated between models and code, shapes of the models (e.g., hierarchy depth and branching factors) appear independent from the code, at least with respect to our limited set of code metrics. We see the following next steps:

- Evaluation of applicability of metrics to further languages and further realistic models.
- Definition and evaluation of valid compound metrics specific to a complex system property, such as cognitive complexity of the model, or complexity of the codebase.
- Theoretical evaluation of the metrics regarding ac-

cepted properties (e.g., additivity, triangle inequality), for instance, using the DISTANCE [34] framework or Briand et al.'s [13] rules of software measurement.

9. REFERENCES

[1] cppstats tool. http://fosd.net/cppstats.

[2] VMM tool. https://bitbucket.org/tberger/vmm.

[3] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire. Reverse engineering architectural feature models. In *ECSA*, 2011.

[4] N. Andersen, K. Czarnecki, S. She, and A. Wąsowski. Efficient synthesis of feature models. In *SPLC*, 2012.

[5] E. Bagheri and D. Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Control*, 19(3):579–612, Sept. 2011.

[6] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. *Encyclopedia of software engineering*, 2(1994):528–532, 1994.

[7] D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.

[8] T. Berger, D. Nair, R. Rublack, J. M. Atlee, K. Czarnecki, and A. Wąsowski. Variability modeling in industry: Practices, benefits, and challenges. Under review.

[9] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski. A survey of variability modeling in industrial practice. In *VaMoS*, 2013.

[10] T. Berger and S. She. Formal semantics of the CDL language. Technical Note. Available at http://informatik.uni-leipzig.de/~berger/cdl_semantics.pdf, 2010.

[11] T. Berger, S. She, K. Czarnecki, and A. Wąsowski. Feature-to-Code mapping in two large product lines. Technical report, University of Leipzig, 2010. Available at http://informatik.uni-leipzig.de/~berger/tr/2010-berger.pdf.

[12] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. A study of variability models and languages in the systems software domain. *IEEE Transactions on Software Engineering*, 39(12):1611–1640, 2013.

[13] L. Briand, S. Morasca, and V. Basili. Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–86, 1996.

[14] C. Calero, M. Piattini, and M. Genero. Empirical validation of referential integrity metrics. *Information and Software Technology*, 43(15):949 – 957, 2001.

[15] L. Chen and M. A. Babar. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, 53(4):344 – 362, 2011.

[16] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Science of Computer Programming*, 76(12):1130 – 1143, 2011.

[17] R. Courtney and D. Gustafson. Shotgun correlations in software measures. *Software Engineering Journal*, 8(1):5–13, 1993.

[18] K. Czarnecki, P. Gruenbacher, R. Rabiser, K. Schmid, and A. Wasowski. Cool features and tough decisions: A comparison of variability modeling approaches. In *VAMOS*, 2012.

[19] E. A. de Oliveira Junior, I. Gimenes, and J. Maldonado. A metric suite to support software product line architecture evaluation. In *CLEI*, 2008.

[20] D. Dhungana, P. Heymans, and R. Rabiser. A formal semantics for decision-oriented variability modeling with DOPLER. In *VaMoS*, 2010.

[21] D. Dhungana, R. Rabiser, P. Grünbacher, and T. Neumayer. Integrated tool support for software product line engineering. In *ASE*, 2007.

[22] H. Eichelberger, C. Kröher, and K. Schmid. An analysis of variability modeling concepts: Expressiveness vs. analyzability. In *ICSR*, 2013.

[23] H. Eichelberger and K. Schmid. A systematic analysis of textual variability modeling languages. In *SPLC*, 2013.

[24] K. M. Eisenhardt and M. E. Graebner. Theory building from cases: Opportunities and challenges. *Academy of management journal*, 50(1):25–32, 2007.

[25] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21, SEI, Carnegie Mellon University, 1990.

[26] L. M. Laird and M. C. Brennan. *Software Measurement and Estimation: A Practical Approach*. IEEE, 2007.

[27] J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze. An analysis of the variability in 40 preprocessor-based software product lines. In *ICSE*, 2010.

[28] R. E. Lopez-Herrejon and S. Trujillo. How complex is my product line? the case for variation point metrics. In *VaMoS*, 2008.

[29] R. Lotufo, S. She, T. Berger, K. Czarnecki, and A. Wasowski. Evolution of the Linux kernel variability model. In *SPLC*, 2010.

[30] M. Mendonca, M. Branco, and D. Cowan. S.P.L.O.T.: software product lines online tools. In *OOPSLA*, 2009.

[31] M. Mendonça, A. Wasowski, and K. Czarnecki. SAT-based analysis of feature models is easy. In *SPLC*, 2009.

[32] S. Nadi and R. Holt. The linux kernel: a case study of build system variability. *Journal of Software: Evolution and Process*, 2013.

[33] L. Passos, J. Guo, L. Teixeira, K. Czarnecki, A. Wąsowski, and P. Borba. Coevolution of variability models and related artifacts: A case study from the linux kernel. In *SPLC*, 2013.

[34] G. Poels and G. Dedene. Distance-based software measurement: necessary and sufficient properties for software measures. *Information and Software Technology*, 42(1):35 – 46, 2000.

[35] N. J. Salkind. *Exploring research*. Prentice Hall, 2003.

[36] S. She and T. Berger. Formal semantics of the Kconfig language. Technical Note. Available at http://eng.uwaterloo.ca/~shshe/kconfig_semantics.pdf, 2010.

[37] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki. Reverse engineering feature models. In *ICSE*, 2011.

[38] T. Zhang, L. Deng, J. Wu, Q. Zhou, and C. Ma. Some metrics for accessing quality of product line architecture. In *CSSE*, volume 2, 2008.

Table 4: Analysis results of model and code metric correlations. The correlation coefficient estimates with p-values ≥0.05 is marked as ——. P-values are omitted for space reasons, but available under [2].

	NF	NGF	NTop	LD_median	NLeaf	LD_mean	BF_mean	BF_max	LD_max	BF_median	RSwitch	RData	RData_num	RNone	RData_string	RCap	RDataOpen	RDerived	RDataRes	RConstr	RDefault	RDerived_lit	RVisibility	RDefault_lit	NOr	RConfVis	NXor	NMutex	ROr	RMutex	RXor	CTCR	RDen	RCon
LOC	1	0.731	0.952	——	1	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.762	——	——	——	——	——	——	——
NOFC	0.905	——	0.857	——	0.905	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.905	——	——	——	——	——	——	——
LOF	0.905	0.778	0.881	——	0.905	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.786	——	——	——	——	——	——	——
ANDAVG	——	——	——	——	——	——	——	——	——	——	-0.857	0.857	——	——	0.833	——	0.857	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
ANDSTDEV	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.81	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
SDEGMEAN	——	——	——	——	——	——	——	——	——	——	-0.738	0.738	0.738	——	——	——	0.738	——	——	——	——	——	——	——	——	0.786	0.738	——	——	——	——	——	——	——
SDEGSTD	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	-0.786	——	——	——	——	——	0.762	0.857	——	——	——	——	——	——	——
TDEGMEAN	——	——	——	——	——	——	——	——	-0.773	0.843	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.833	——	——	——	——	0.762	0.762	0.762	——
TDEGSTD	——	——	——	——	——	——	——	——	——	0.723	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.857	——	——	——	——	——	——	——	——
HOM	0.952	0.755	1	——	0.952	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.762	——	——	——	——	——	——	——
HET	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
HOHE	0.976	0.79	0.976	——	0.976	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.786	——	——	——	——	——	——	——
GRANGL	0.905	——	0.857	——	0.905	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.905	——	——	——	——	——	——	——
GRANFL	0.929	0.85	0.929	——	0.929	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
GRANBL	0.905	0.922	0.905	——	0.905	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
GRANSL	0.738	——	——	——	0.738	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
GRANEL	0.805	0.785	——	——	0.805	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
GRANML	0.85	——	0.802	——	0.85	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
GRANERR	0.755	——	——	——	0.755	——	——	——	——	——	——	——	——	-0.79	——	——	——	——	——	——	——	——	——	——	——	——	0.85	——	——	——	——	——	——	——
NDMAX	——	——	——	——	——	——	——	——	——	0.758	-0.731	0.731	——	——	——	——	0.731	0.778	——	——	——	0.778	——	——	——	0.934	0.778	——	——	——	——	——	——	——
RDerived_expr	0.764	0.741	0.764	——	0.764	——	——	——	——	——	——	——	0.764	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
RDefault_expr	——	0.768	——	——	——	0.733	——	——	——	——	——	——	——	——	——	0.756	——	——	——	——	——	——	——	——	——	——	——	0.756	——	0.756	——	——	——	——
RPurelyBoolConstr	-0.723	——	——	——	-0.723	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——

Table 3: Analysis results of model metric correlations. The upper diagonal part contains correlation coefficient estimates, and the lower diagonal part contains corresponding p-values. The diagonal is marked as ***, and correlation coefficient estimates with p-values ≥0.05 as ——.**

	NF	NGF	NTop	LD_median	NLeaf	LD_mean	BF_mean	BF_max	LD_max	BF_median	RSwitch	RData	RData_num	RNone	RData_string	RCap	RDataOpen	RDerived	RDataRes	RConstr	RDefault	RDerived_lit	RVisibility	RDefault_lit	NOr	RConfVis	NXor	NMutex	ROr	RMutex	RXor	CTCR	RDen	RCon
NF	*****	0.731	0.952	——	1	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.762	——	——	——	——	——	——	——
NGF	0.04	*****	0.755	——	0.731	0.731	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
NTop	0.001	0.031	*****	——	0.952	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.762	——	——	——	——	——	——	——
LD_median	——	——	——	*****	——	0.913	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
NLeaf	<0.001	0.04	0.001	——	*****	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.762	——	——	——	——	——	——	——
LD_mean	——	0.04	——	0.002	——	*****	-0.762	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
BF_mean	——	——	——	——	——	0.037	*****	——	-0.724	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
BF_max	——	——	——	——	——	——	——	*****	——	——	——	——	-0.786	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
LD_max	——	——	——	——	——	——	0.042	——	*****	-0.745	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	-0.737	-0.786	-0.835	——
BF_median	——	——	——	——	——	——	——	——	0.034	*****	-0.771	0.771	——	——	——	——	0.771	——	——	——	——	——	——	——	——	——	——	——	——	——	0.843	0.783	0.771	——
RSwitch	——	——	——	——	——	——	——	——	——	0.025	*****	-1	-0.833	——	-0.929	——	-1	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
RData	——	——	——	——	——	——	——	——	——	0.025	<0.001	*****	0.833	——	0.929	——	1	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
RData_num	——	——	——	——	——	——	——	——	——	——	0.015	0.015	*****	——	0.762	——	0.833	——	——	——	——	0.762	——	——	——	——	——	——	——	——	——	——	——	——
RNone	——	——	——	——	——	——	——	0.028	——	——	——	——	——	*****	——	——	——	——	——	——	0.762	——	-0.833	——	——	——	-0.857	——	——	——	——	——	——	——
RData_string	——	——	——	——	——	——	——	——	——	0.002	0.002	0.037	——	*****	——	——	0.929	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
RCap	——	——	——	——	——	——	——	——	——	——	——	——	——	——	*****	——	——	——	——	——	——	——	——	——	——	——	——	1	——	1	——	——	——	——
RDataOpen	——	——	——	——	——	——	——	——	——	0.025	<0.001	<0.001	0.015	——	0.002	——	*****	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
RDerived	——	——	——	——	——	——	——	——	——	——	——	——	0.015	——	——	——	——	*****	——	——	-0.81	1	——	-0.81	——	0.857	0.833	——	——	——	——	——	——	——
RDataRes	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	*****	*****	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——
RConstr	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	*****	*****	——	——	——	——	——	——	——	——	——	——	——	——	——	——
RDefault	——	——	——	——	——	——	——	——	——	——	——	——	0.037	——	——	——	0.022	——	——	*****	-0.81	——	——	1	——	——	-0.857	——	——	——	——	——	-0.738	——
RDerived_lit	——	——	——	——	——	——	——	——	——	——	——	——	0.015	——	——	——	<0.001	——	——	——	*****	——	——	-0.81	——	0.857	0.833	——	——	——	——	——	——	——
RVisibility	——	——	——	——	——	——	——	——	——	——	——	0.037	——	——	——	——	——	——	——	——	*****	*****	——	——	——	——	——	——	——	——	——	——	——	——
RDefault_lit	——	——	——	——	——	——	——	——	——	——	——	——	0.037	——	——	——	0.022	——	——	<0.001	0.022	——	*****	——	——	——	-0.857	——	——	——	——	——	-0.738	——
NOr	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	*****	——	——	——	——	1	——	——	——	——	——
RConfVis	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.011	——	——	——	0.011	——	——	——	*****	0.81	——	——	——	——	——	——	——	——
NXor	0.037	——	0.037	——	0.037	——	——	——	——	——	——	——	0.011	——	——	——	0.015	——	——	——	0.011	0.015	——	0.011	——	0.022	*****	——	——	——	——	——	——	——
NMutex	——	——	——	——	——	——	——	——	——	——	——	——	——	——	<0.001	——	——	——	——	——	——	——	——	——	——	——	——	*****	——	1	——	——	——	——
ROr	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	<0.001	——	——	——	*****	——	——	——	——	——
RMutex	——	——	——	——	——	——	——	——	——	——	——	——	——	——	<0.001	——	——	——	——	——	——	——	——	——	——	——	<0.001	——	*****	——	——	——	——	——
RXor	——	——	——	——	——	——	——	——	0.037	0.009	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	*****	——	0.786	——
CTCR	——	——	——	——	——	——	——	——	0.021	0.022	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	*****	0.929	0.929
RDen	——	——	——	——	——	——	——	——	0.01	0.025	——	——	——	——	——	——	——	——	——	——	0.046	——	——	0.046	——	——	——	——	——	——	0.028	0.002	*****	0.929
RCon	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	——	0.002	0.002	*****