Towards Effective and Efficient Feature-Model Analyses for Evolving System Software

Elias Kuiter
kuiter@ovgu.de
University of Magdeburg
Magdeburg, Germany

Abstract

Software-intensive systems are often configurable, with system software frequently being developed as product lines to manage variability. Feature models describe the configurable features and their dependencies in such product lines, enabling automated analysis via reasoning tools. However, some challenges remain: Feature models must be correctly extracted and transformed for effective analysis, their complexity can hamper efficiency, and their evolution over time may exacerbate these issues. In the proposed thesis, we address these challenges by (1) evaluating the impact of extraction and transformation on analysis effectiveness and efficiency, (2) studying long-term evolution trends in the configurability and computational complexity of feature models, and (3) studying individual evolution steps, so practitioners can better assess the impact of updates on end users. Our overall goal is to improve our understanding of large, real-world feature models and their evolution.

CCS Concepts

• **Software and its engineering** → **Software product lines**; *Software evolution*; • **Theory of computation** → *Automated reasoning*.

Keywords

feature modeling, satisfiability solving, system software

ACM Reference Format:
Elias Kuiter. 2025. Towards Effective and Efficient Feature-Model Analyses for Evolving System Software. In *29th ACM International Systems and Software Product Line Conference - Volume B (SPLC-B '25), September 1–5, 2025, A Coruña, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3748269.3748593

PhD Advisors: Prof. Dr. GUNTER SAAKE (University of Magdeburg, Germany)
Prof. Dr. THOMAS THÜM (TU Braunschweig, Germany)

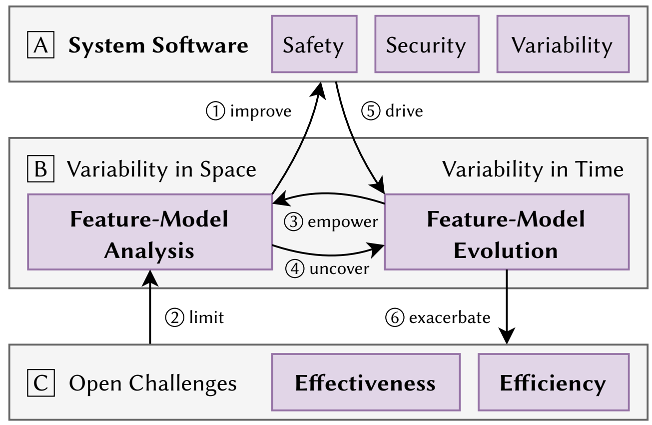

Figure 1: Concept map for the thesis, highlighting titular concepts and key relationships.

1 Motivation

Software Variability In today's industry, software-intensive systems are expected to satisfy diverse customer requirements [25, 75]. For example, modern cars can be heavily customized to fit their owners' individual needs and preferences [31, 101]. Besides the automotive industry, this trend towards *mass customization* [26] can also be observed in other domains of manufacturing, such as avionics, consumer electronics, and the internet of things. Many of the systems in these domains are *embedded* and *cyber-physical systems*, which are primarily customized on a hardware level (e.g., by choosing manual or automatic transmission in a car). However, embedded and cyber-physical systems increasingly rely on large amounts of software as well, which too must be geared towards the end user's customizations [12, 38]. In addition to embedded and cyber-physical systems, the demand for mass customization has also carried over to classical *software systems* (or *application software*) across various domains, such as business software, databases, and finance [84]. So, in practice, many software systems are configurable to some extent. In other words, they contain *software variability* [30], which allows customers to select and refine the functionality they need [20].

System Software Besides embedded, cyber-physical, and software systems, there is *system software* (e.g., operating systems, firmware, and drivers), which acts as a mediator between hard- and software. System software provides customizable hardware abstraction layers (HALs) and application programming interfaces (APIs), which serve to accommodate wide varieties of hard- and software. Thus, system software commonly contains variability, which allows practitioners to derive tailored variants for their use cases. Compared to the other software-intensive systems mentioned above, system software is remarkable in several ways: First, system software is an infrastructure-critical bottleneck between hard- and software and must therefore meet exceedingly high standards for safety and security. Second, system software must also flexibly accommodate almost countless combinations of hard- and software, a diversity that necessitates variability. Consequently, a major goal

ACM ISBN 979-8-4007-2080-2/2025/09
https://doi.org/10.1145/3748269.3748593

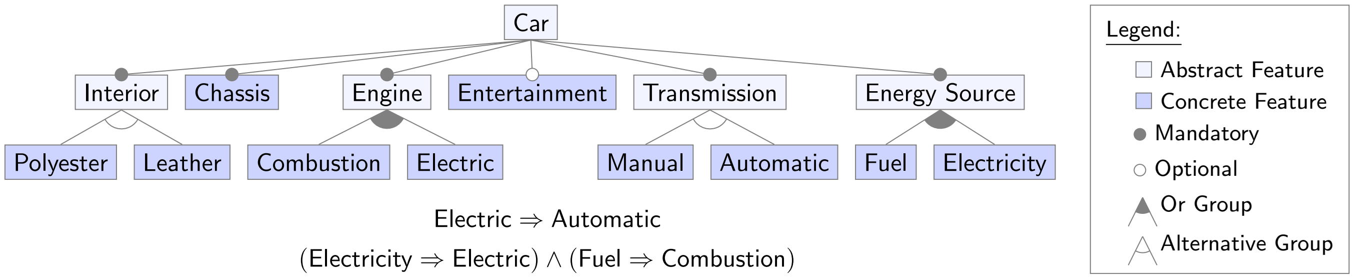

Figure 2: Example feature model for a car product line.

during the development of system software is to ensure safety and security in the presence of large-scale variability. In Figure 1, we schematically depict these properties of system software in $\boxed{A}$.

Software Product Lines To effectively and systematically develop and manage variability, software-intensive systems (including system software) can be developed as software product lines [2, 14, 75, 97]. A *product line* is a set of related products that share a common core but also differ in some functionalities they offer [37]. For example, a car manufacturer may offer different car models that share the same chassis but differ in their interior, engine, or entertainment system. More specifically, a *software product line* (SPL) is "a set of software-intensive systems that share a common, managed set of *features* satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [69]. Besides allowing for mass customization, the adoption of SPLs can reduce development costs and sustain the quality and maintainability of highly-configurable software-intensive systems [14, 44, 75].

In the domain of system software, many projects have successfully adopted techniques from SPL engineering to manage variability [84]. One prominent example is the Linux kernel, which is a general-purpose operating system and well-known for being highly configurable [27, 62, 70, 81, 83, 85].

Feature Models To systematically model the variability of SPLs (such as the Linux kernel), different kinds of techniques for variability modeling have been proposed [8, 22, 39]. In particular, *feature models* [2, 39] are well-known and widely used for this task [8, 16, 67]. A feature model describes the *valid configurations* of an SPL by modeling its features and their dependencies, where a *feature* is a characteristic or end-user-visible behavior of a software system [2]. For example, manual and automatic transmission in a car are features that are mutually exclusive. As this dependency must always be respected when customizing a car, it should be documented. We can represent such (otherwise tacit) knowledge explicitly in a feature model, which we visualize in Figure 2. Feature models allow stakeholders to communicate about variability in all phases of the SPL development life cycle, for example to avoid redundant development effort [7]. Nonetheless, it can still be challenging for humans to understand and reason about the variability of an SPL. This is especially true when there are many features [91] or complex dependencies [45] between them. For example, suppose a car manufacturer wants to gradually convince their customers to switch to electric vehicles. Most electric vehicles today require automatic transmission (cf. Figure 2). As this conflicts with manual transmission, customers who prefer manual transmission will be hard to reach. This conflict is, however, not immediately obvious from the listed constraints. To reveal such implicit knowledge, the manufacturer can perform a time-consuming manual analysis of the feature model, or employ automated analyses instead.

In the domain of system software, variability is often described in domain-specific languages (DSLs) such as KConfig, the homegrown configuration language of the Linux kernel. Unfortunately, KConfig specifications do not directly correspond to feature models such as the one shown in Figure 2. Hence, several techniques have been proposed for extracting feature models from KConfig specifications, therefore enabling automated analysis.

Feature-Model Analysis A wide variety of automated *product-line analyses* have been proposed in the literature [6, 86, 94] and applied in all phases of the software development process. For example, product-line analyses have successfully been applied in the context of interactive configuration [35, 50], anomaly detection [6, 68, 79] and explanation [23, 46], evolution [47, 95], modularization [49, 78], testing [42, 74], static code analysis [10, 61], type checking [3, 41, 43], model checking [4, 76] and formal verification [54, 96]. Many of these analyses fundamentally rely on automated *feature-model analyses*, which investigate the feature model's *configuration space* (i.e., its semantics instead of its syntax). To infer information about this configuration space, feature models are typically transformed into *propositional formulas* [2, 5, 17, 18, 66, 77]. These formulas can then be analyzed using off-the-shelf reasoning tools, such as *satisfiability solvers* (SAT solvers) [6, 60, 65] or *model counters* (#SAT solvers) [52, 70, 85, 89]. While this approach is straightforward in principle, it is sometimes challenging in practice, as it is not necessarily effective and efficient on all feature models.

In the domain of system software, feature-model analyses (depicted as part of $\boxed{B}$ in Figure 1) can be particularly beneficial, as they may reveal safety and security issues (see 1). At the same time, analyses can be particularly challenging on these SPLs in terms of *effectiveness* and *efficiency* (depicted in $\boxed{C}$, see 2): First, feature models initially have to be extracted from DSL specifications (e.g., KConfig) and transformed into propositional formulas. In principle, these steps could impact the feasibility, correctness, and accuracy of subsequent analyses (e.g., resulting in flawed results). Second, for some SPLs, the resulting feature-model formulas can become large and complex [45, 85, 93]. This sometimes makes their analysis computationally expensive due to SAT and #SAT being NP- and #P-complete, respectively. Thus, the complexity of feature-model formulas can impact the scalability, performance, and resource consumption (e.g., memory or energy) of analysis algorithms.

Feature-Model Evolution Another dimension to analyses is that SPLs (and their feature models) are a moving target, as they evolve over time. Similar to classical software evolution, *SPL evolution* is commonly driven by iterative development and changing requirements [57], new features, or bug fixes. Many open challenges regarding SPLs relate to their evolution, for example in the automotive domain [38]. Thus, when studying software variability, it is advisable to consider both *variability in space* (i.e., variants due to SPLs) and *variability in time* (i.e., revisions due to evolution, both depicted together in $\boxed{\text{B}}$ in Figure 1). If regarded in concert, the analysis of variability in space and time have a symbiotic relationship: On one hand, some automated analyses can leverage information about the evolution of an SPL (see ③). For example, it might be beneficial to reuse old analysis results on newer revisions to avoid a full recomputation [33, 47] or to estimate analysis results. On the other hand, automated analyses can also help improve our knowledge of an SPL's evolution, for example regarding its configuration space (see ④). Thus, analyses can help researchers gain new insights and serve as a decision-making tool for practitioners. Among many evolving artifacts like source code or requirements, feature models are highly relevant when studying SPL evolution. This is because they document high-level changes to an SPL's variability, such as added or removed features or dependencies.

In the domain of system software, evolution is particularly relevant (see ⑤): First, many SPLs in this domain have been time-proven to be trustworthy for powering critical infrastructure, which is why they often have a long and rich evolution history (opposed to more short-term-focused application software). Often, these systems are developed as *free and open-source software* [15], so their evolution histories are publicly available, which improves accountability. Second, the long history, broad usage, and potential upgrade restrictions of these SPLs imply that a variety of revisions are used in the field (opposed to always up-to-date application software). This merits a more in-depth analysis of these SPLs' evolving feature models, which relies on propositional formulas and automated reasoning tools. However, the above-mentioned challenges (see $\boxed{\text{C}}$) regarding the effectiveness and efficiency of feature-model analyses can also hamper such an evolutionary analysis (see ② and ④). Indeed, evolution may even exacerbate these challenges due to a tendency for software systems to grow continuously (see ⑥).

2 Proposal

Objectives In the proposed thesis, we aim to address several open challenges regarding effective and efficient feature-model analyses for evolving system software. We classify our objectives in terms of the following three concerns mentioned above (cf. Figure 1):

- *Effectiveness.* Depending on several influence factors, automated feature-model analyses are not always correct or sufficiently accurate. We aim to identify relevant influence factors and evaluate their impact on the effectiveness of feature-model analyses. In particular, we want to investigate the necessary extraction and transformation of feature models in the system software domain. Moreover, we aim to identify when semantic analyses should be used instead of syntactic analyses, as the former might be more accurate.
- *Efficiency.* Analogous to effectiveness, automated feature-model analyses are not necessarily scalable, performant, or resource-efficient. In the domain of system software, feature models can become particularly large and complex, which can make analyses computationally expensive. We aim to evaluate how the above-mentioned influence factors affect the efficiency of common feature-model analyses (e.g., in terms of runtime, memory or energy consumption). Moreover, we aim to identify when one should invest effort into semantic analyses, as they can be more resource-consuming than syntactic analyses.
- *Evolution.* The evolution of feature models can reveal new insights about the past, present, and potential future development of SPLs. System software, in particular, often has a rich history. However, current methods for its feature models' semantic analysis are limited in terms of effectiveness and efficiency. We aim to propose an improved technique for computing feature-model differences. Based on this technique, we investigate two evolution antipatterns (i.e., inadvertent variability reduction and growth).

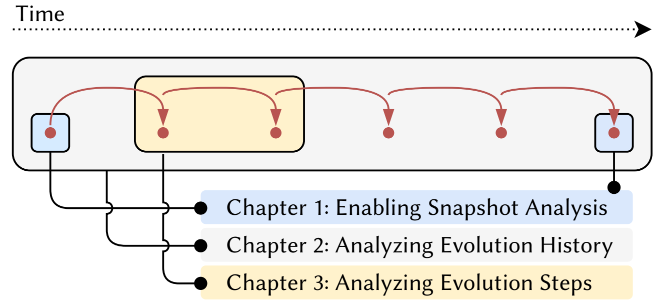

Figure 3: The planned thesis structure, highlighting different granularities of (evolutionary) feature-model analysis.

Consequently, we are primarily concerned with basic research on these three fundamental concerns and how they interact and reinforce each other (cf. Figure 1). First, we perform empirical evaluations to improve our understanding of these concerns and their interplay. Based on this knowledge, we derive according insights and recommendations for researchers and practitioners. Second, we propose new algorithms and metrics that improve on remaining open challenges regarding these concerns. Overall, we intend to strengthen the internal and external validity of our own and other research evaluations (e.g., to avoid flawed results and allow for studying more systems). Moreover, we intend to publish all our contributions as open artifacts to ensure reproducibility.

Structure We aim to divide the proposed thesis into three chapters. The first chapter mostly focuses on the effectiveness and efficiency objectives, while the second and third chapter address the evolution objective in more detail. Each chapter addresses a different use case for (evolutionary) feature-model analysis of system software:

(1) *Enabling Snapshot Analysis.* First, we investigate the extraction and transformation of feature-model formulas for individual

revisions (i.e., *snapshots* in time). Thus, we aim to enable the analysis of historic revisions with improved effectiveness and efficiency, while also laying the necessary groundwork for investigating evolutionary analyses in the following chapters.

(2) *Analyzing Evolution History.* Second, we study the long-term evolution of feature models by considering the entire *history* of an SPL. Thus, we aim to discover and discuss trends and patterns in the evolution of variability in system software.

(3) *Analyzing Evolution Steps.* Third, we study the fine-grained evolution of feature models in terms of individual *steps* (e.g., by comparing two neighboring feature-model revisions). Thus, we aim to gain detailed insights not covered by the previous coarse-grained analysis (e.g., how an update affects end users).

We schematically depict this structure in Figure 3, where each point corresponds to a feature-model revision. The feature model is evolving over time in discrete steps, which we visualize as arrows.

Impact and Novelty First and foremost, the proposed thesis contributes to the field of software engineering, specifically to the area of SPLs. We aim to reach both SPL researchers and practitioners: For researchers, we aim to identify significant influence factors and bottlenecks for feature-model analyses, which have been overlooked or underestimated in previous work. In investigating these influence factors and their tradeoffs, we also aim to make appropriate recommendations for a given analysis use case. We also study the evolution of configuration spaces, where we take a semantic viewpoint (instead of the purely syntactic viewpoint typically found in previous work). For practitioners (e.g., developers or end users), we aim to provide insights and recommendations for making more informed decisions about the development and maintenance of highly-configurable software-intensive systems. Moreover, our open artifacts empower both researchers and practitioners to independently reproduce our results and even gain new insights into the past, present, and future development of SPLs. By incorporating our artifacts into continuous integration and delivery pipelines, this process may even be automated. Second, the proposed thesis also contributes to the field of automated reasoning. That is, in contrast to previous work in this field, we study transformation steps and the evolution of propositional formulas in a real-world setting.

3 Contributions

In the following, we describe the contributions made by each individual chapter (cf. Figure 3) and how they relate to our objectives.

3.1 Enabling Snapshot Analysis

To analyze feature models, we typically extract feature-model formulas from DSL specifications (e.g., written in KConfig) and transform them into a machine-readable representation (e.g., conjunctive normal form). In the first chapter of the thesis, we focus on this extraction and transformation (cf. Figure 4). We primarily address the thesis' objectives regarding efficiency and effectiveness for analyzing individual models (i.e., *snapshots*). Still, this chapter is an important stepping stone towards evolutionary feature-model analyses: First, it lays the foundation for extracting complete and consistent feature-model histories from SPLs in the system software domain. Second, studying the efficiency of the extraction and transformation phases is relevant for long-term evolutionary analyses.

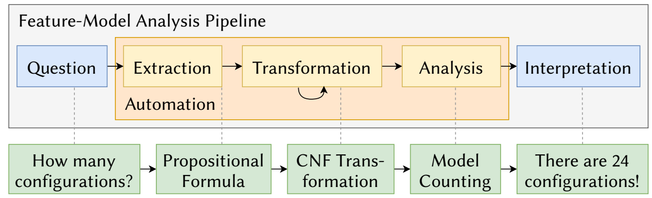

Figure 4: A typical feature-model analysis pipeline, including an example related to Figure 2.

Third, investigating the interaction between the transformation phase and subsequent analyses is crucial for algebraic computations with evolutionary applications (e.g., slicing and diffing) [86].

RQ_1 KConfig Specification To benefit from feature-model analyses in the system software domain, we must be able to extract accurate feature-model formulas from commonly-used KConfig specifications. Unfortunately, existing extractors [58] can usually not be applied without modification to new projects. Moreover, the KConfig language evolves over time. These issues can lead to outdated or incorrect feature-model snapshots or, for evolutionary analyses, to fragmentary or inconsistent histories.

$RQ_{1.1}$ *How to extract feature-model formulas from real-world KConfig specifications, and how do approaches compare?*
In progress – Concerns effectiveness

To assess and mitigate the limitations of existing extractors, we integrate and improve on them with torte,[1] our declarative workbench for feature-model analysis. We use torte to extract complete and consistent feature-model histories and to compare the extractors.

$RQ_{1.2}$ *How to recover feature-model hierarchies from real-world KConfig specifications?*
In progress – Concerns efficiency

Unfortunately, machine-readable formulas extracted by existing tools do not directly correspond to human-readable feature diagrams. We intend to extend torte so it can recover comprehensible feature-model hierarchies from KConfig specifications.

RQ_2 Conjunctive Normal Form To compute automated analyses on feature models, it is almost always necessary to first transform the extracted feature-model formula (RQ_1) into conjunctive normal form (CNF). Besides being unnecessary bottleneck, this step is also nontrivial, as feature models extracted from system software regularly contain constraints of arbitrary complexity [45].

$RQ_{2.1}$ *How to transform feature-model formulas into CNF?*
Published at ASE [55] and SE [56] – Concerns effectiveness

We review existing CNF transformation algorithms and create a taxonomy of recurring properties to classify them. We find three seminal algorithms (i.e., the distributive, Tseitin, and Plaisted-Greenbaum transformation). For each algorithm, we judge its suitability for correctly computing various feature-model analyses. We find that the definitional transformations (which introduce variables, such as Tseitin and Plaisted-Greenbaum) have limitations: That is, both do not compose well with subsequent transformations. Moreover,

[1] https://github.com/ekuiter/torte

the Plaisted-Greenbaum transformation leads to wrong results for #SAT-based analyses.

RQ$_{2.2}$ *How does CNF transformation impact the work of practitioners and researchers?*
Published [55, 56] – Concerns effectiveness and efficiency

We perform a black-box analysis of representative CNF transformation tools (i.e., FeatureIDE [48, 64], KConfigReader [40, 41] and Z3 [19]). We evaluate both the accuracy (i.e., correctness) and efficiency of the CNF transformation tool and of subsequent analyses using SAT and #SAT solvers. We find significant and large differences between the tools in terms of accuracy and efficiency. We empirically confirm that the distributive transformation fails on complex feature-model formulas and the Plaisted-Greenbaum transformation is unsuitable for #SAT-based analyses (RQ$_{2.1}$).

RQ$_{2.3}$ *How to parameterize CNF transformation algorithms?*
In progress [80] – Concerns efficiency

Our black-box analysis (RQ$_{2.2}$) already demonstrates the practical impact of CNF transformation tools on feature-model analysis. However, it cannot be used to directly compare transformation algorithms. Thus, an in-depth, white-box analysis is still missing, which complements our black-box analysis by focusing on internal more than external validity [82, 100]. To this end, we intend to implement the three seminal CNF transformation algorithms in clausy,[2] our own CNF transformation tool. Thus, we avoid confounding factors due to using different tools (e.g., performance of the programming language), so we can focus on the impact of the algorithms themselves. In addition, we will make these algorithms configurable (e.g., in terms of the number of introduced variables), so we can better understand the impact of parametrization.

RQ$_3$ Non-Clausal Slicing A feature-model slice hides parts of a feature model, which has numerous applications for feature-model analyses. Unfortunately, slicing is still a computationally expensive or even infeasible operation, especially on complex feature-model formulas that require variable-introducing CNF transformations.

RQ$_{3.1}$ *How to compute non-clausal slices of feature models?*
In progress – Concerns efficiency

We aim to propose a new slicing algorithm, which can be executed before definitional CNF transformations (RQ$_{2.1}$). To this end, the algorithm must be applied before CNF transformation, making it non-clausal. We intend to base our algorithm on leading slicing algorithms [49, 88]. We will evaluate the efficiency of our algorithm by implementing it in clausy and determining whether our algorithm slices complex feature-model formulas more efficiently than state-of-the-art slicing algorithms.

3.2 Analyzing Evolution History

In the second chapter, we focus on the long-term *evolution history* of feature models, which is rarely studied semantically (i.e., by analyzing feature-model formulas). In particular, we aim to investigate two fundamental aspects of evolving feature models: their configurability and their computational complexity.

RQ$_4$ Configurability Two fundamental and characterizing metrics for SPLs are their numbers of features or configurations. These metrics are used in research and practice to contextualize evaluation results, make development decisions, or build more complex analyses. Thus, it is beneficial to compute configurability metrics correctly, and draw conclusions from their evolution over time.

RQ$_{4.1}$ *Which factors influence configurability metrics?*
Published at TOSEM [58] – Concerns effectiveness and evolution

To ensure correct computation of configurability metrics, we identify relevant factors that may (or may not) influence them. We evaluate the influence of all factors we identify. We find that there are several ways to define the number of features in the system software domain. These definitions differ significantly, which explains seemingly contradictory numbers reported in the literature.

RQ$_{4.2}$ *Case study: How configurable is the Linux kernel?*
Published [58] – Concerns effectiveness, efficiency and evolution

In the domain of system software, the Linux kernel [83] is one of the largest and most influential SPLs to-date. This is well-illustrated by the version control system Git and the configuration language KConfig, which were both originally developed for the kernel to manage variability in time and space. Due to its influence and complexity, the Linux kernel makes a good case study for investigating configurability, for which we currently lack reliable knowledge. We evaluate the kernel's number of features and configurations on more than 3,000 feature-model revisions, which span a timeframe of more than twenty and ten years, respectively. We compare both metrics to assess the merit of semantic analysis (e.g., number of configurations) compared to syntactic analysis (e.g., number of features). Moreover, we hypothesize on the future development of the kernel's configurability and how this may affect its maintainability.

RQ$_5$ Computational Complexity As feature models evolve over time, so does their computational complexity. Thus, evolution may affect the possibilities and limits of current reasoning techniques, which are crucial to the success of feature-model analyses.

RQ$_{5.1}$ *Can reasoning tools keep up with feature-model evolution?*
In progress [11] – Concerns efficiency and evolution

Software systems generally tend to grow more complex [59]. At the same time, automated reasoning tools tend to get more efficient over time [28, 36]. It is unknown whether one of these tendencies dominates the other (this is analogous to Wirth's law [24, 99]). To identify the dominating tendency, we evaluate the efficiency of SAT solvers from the last two decades by using them to analyze feature models from the same timespan. Thus, we aim to determine whether reasoning tools like SAT solvers need more substantial innovation to account for growing feature models in the future.

3.3 Analyzing Evolution Steps

In the third chapter, we perform a more in-depth analysis of individual *evolution steps* (e.g., `v6.11–v6.12` of the Linux kernel). Such evolution steps represent differences between feature models, which can be beneficial to study for improved quality assurance (e.g., to assess the impact of updates on end users). We aim to characterize typical evolution steps (benefiting researchers) and propose new quality assurance metrics (benefiting practitioners).

RQ$_6$ Semantic Differencing Syntactic differences between feature models are well-understood and can be computed efficiently [21, 51, 63, 71, 73]. However, semantic differences are more challenging

[2]https://github.com/ekuiter/clausy

to compute, as they require computationally complex reasoning about feature-model formulas. Consequently, semantic differencing is rarely done by researchers and practitioners, which limits the insights they can gain from feature-model evolution.

$RQ_{6.1}$ *How to compute semantic differences of feature models?*
In progress – Concerns effectiveness, efficiency, and evolution

Existing techniques for semantic differencing [1, 95] have several limitations: They only allow for coarse classification of evolution steps [95] or do not scale to complex feature models due to the use of knowledge compilation [1]. We propose a new technique for semantic differencing, which is based on Tseitin transformation (RQ_2) and slicing (RQ_3). Thus, we avoid knowledge compilation and reduce the number of calls to reasoning tools. Our technique quantifies the differences between two feature models and optionally reifies them as a new feature-difference model. We evaluate our technique on feature-model histories in the system software domain to investigate what characterizes a typical evolution step. We also compare it to existing algorithms in terms of efficiency.

$RQ_{6.2}$ *Do evolution steps preserve backward compatibility?*
In progress – Concerns evolution

As one use case for our semantic differencing technique ($RQ_{6.1}$), we propose a new quality assurance metric on feature models. Our metric measures the backward compatibility of an evolution step by quantifying the amount of existing configurations that require manual intervention from end users to fix. Thus, it can be used to detect inadvertent variability reduction in an SPL. By integrating our metric in a continuous integration pipeline, developers can ensure that updates do not invalidate end users' existing configurations.

$RQ_{6.3}$ *Do evolution steps introduce new feature interactions?*
In progress – Concerns evolution

Analogous to inadvertent variability reduction, developers may also sometimes introduce new, subtle feature interactions [13] with an update (i.e., inadvertent variability growth). For example, this may happen when features (and their dependencies) are naively removed from KConfig specifications. Based on our semantic differencing technique ($RQ_{6.1}$), we propose another new metric that measures the potential of an evolution step to introduce feature interactions. By reifying and sampling [98] an evolution step as a feature-difference model, we can even recommend specific new configurations to test, such that all new interactions are covered.

4 Timeline and Scope

In Table 1, we show all publications that are related to the thesis, including their current status, priority, as well as a rough timeline.

We rely on three pillars, each of which contributes to one of the three thesis chapters by building on and extending an influential publication: First, our black-box analysis of CNF transformations [55] investigates an understudied step in the original introduction of propositional formulas for feature-model analysis, proposed by Batory [5]. Second, our analysis of the Linux kernel's evolution history [58] significantly extends an empirical analysis performed by She et al. [81]. Third, our planned work on semantic differencing improves on a well-known algorithm proposed by Thüm et al. [95]. We consider these to be the pillars of the thesis,

Table 1: Main contributions of published and planned publications related to the thesis.

	Year	Main Contribution	RQ	Publication(s)
∗	2022	Black-Box Analysis of CNF Transforms	RQ_2	ASE [55], SE [56]
–	2024	Instance-Based Meta-Analysis	RQ_5	VaMoS [53]
∗	2024	Analysis of Linux' Evolution History	RQ_4	TOSEM [58]
–	2025	Variability Growth vs. SAT Solving	RQ_5	*Submitted*
∗	2025	Semantic Differencing and QA Metrics	RQ_6	
†	2025	Tool: torte	RQ_1	
–	2026	Scalable Non-Clausal Slicing	RQ_3	
–	2026	White-Box Analysis of CNF Transforms	RQ_2	

∗ High priority – Middle priority † Low priority

Related co-authorships and supervisions With Sundermann et al. [84, 86, 87] on feature-model collection and knowledge compilation; four completed supervisions on computational complexity (RQ_5); eight ongoing supervisions on KConfig specification (RQ_1), non-clausal slicing (RQ_3), and computational complexity (RQ_5).

as each of them builds upon previous influential work (i.e., having earned test-of-time [5] and most influential paper [81, 95] awards).

In addition to these high-priority pillars, we plan to submit several publications of medium or low priority. In case of time constraints, these can be published as preprints. Besides time constraints, we believe that the highest risk of the project lies in the non-clausal slicing, for which we have no algorithm or implementation prototype yet. Still, even without non-clausal slicing, we believe that the thesis contributions will be highly valuable.

To keep the scope of the proposed thesis narrow, we will focus on the issues outlined above. Thus, we will not go into detail about related, but distinct aspects (e.g., domains outside of system software [84], non-Boolean variability [9, 72], solution-space variability [32], solver internals, and alternative solvers [29, 34, 90, 92]).

5 Conclusion

With the proposed thesis, we contribute to the field of software engineering at its intersection with automated reasoning. We focus on the system software domain, where it is crucial to meet safety and security standards in the presence of large-scale hard- and software variability. Specifically, we aim to improve the analysis of evolving feature models in terms of effectiveness and efficiency. Notably, we follow up on several influential papers in the field [5, 81, 95]. We significantly extend these papers by discussing and evaluating previously overlooked aspects in detail, such as conjunctive normal form (RQ_2) [5], configurability (RQ_4) [81], and semantic differencing (RQ_6) [95]. Thus, we aim to advance basic research on feature-model analyses and strengthen the validity of research evaluations. Moreover, we propose new metrics and tools for feature-model analyses, which researchers and practitioners can use to improve their understanding of large, real-world feature models.

Acknowledgments

Many thanks go to my advisors Gunter Saake and Thomas Thüm for their valuable feedback and support. I also want to thank Sebastian Krieter, Jacob Krüger, Paul Maximilian Bittner, David Broneske, and Paul Blockhaus for helpful discussions on this proposal. Finally, thanks to the anonymous reviewers for their constructive feedback.

References

[1] Mathieu Acher, Patrick Heymans, Philippe Collet, Clément Quinton, Philippe Lahire, and Philippe Merle. 2012. Feature Model Differences. In *CAiSE*. Springer, 629–645. doi:10.1007/978-3-642-31095-9_41

[2] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines*. Springer. doi:10.1007/978-3-642-37521-7

[3] Sven Apel, Christian Kästner, Armin Größlinger, and Christian Lengauer. 2010. Type Safety for Feature-Oriented Product Lines. *AUSE* 17, 3 (2010), 251–300. doi:10.1007/s10515-010-0066-8

[4] Sven Apel, Hendrik Speidel, Philipp Wendler, Alexander von Rhein, and Dirk Beyer. 2011. Detection of Feature Interactions Using Feature-Aware Verification. In *ASE*. IEEE, 372–375.

[5] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *SPLC*. Springer, 7–20. doi:10.1007/11554844_3

[6] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems* 35, 6 (2010), 615–708. doi:10.1016/J.IS.2010.01.001

[7] Thorsten Berger, Divya Nair, Ralf Rublack, Joanne M. Atlee, Krzysztof Czarnecki, and Andrzej Wąsowski. 2014. Three Cases of Feature-Based Variability Modeling in Industry. In *MODELS*. Springer, 302–319. doi:10.1007/978-3-319-11653-2_19

[8] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *VaMoS*. ACM, 7:1–7:8. doi:10.1145/2430502.2430513

[9] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. 2013. A Study of Variability Models and Languages in the Systems Software Domain. *TSE* 39, 12 (2013), 1611–1640. doi:10.1109/TSE.2013.34

[10] Eric Bodden, Társis Tolêdo, Márcio Ribeiro, Claus Brabrand, Paulo Borba, and Mira Mezini. 2013. SPLLIFT: Statically Analyzing Software Product Lines in Minutes Instead of Years. In *PLDI*. ACM, 355–364. doi:10.1145/2491956.2491976

[11] Urs-Benedict Braun. 2024. *Automatisierte Feature-Modell-Analyse des Linux-Kernels: Eine Reise durch die Zeit*. Bachelor's Thesis. University of Magdeburg. Submitted.

[12] Manfred Broy. 2006. Challenges in Automotive Software Engineering. In *ICSE*. ACM, 33–42. doi:10.1145/1134285.1134292

[13] Muffy Calder, Mario Kolberg, Evan H. Magill, and Stephan Reiff-Marganiec. 2003. Feature Interaction: A Critical Review and Considered Forecast. *ComNet* 41, 1 (2003), 115–141.

[14] Paul Clements and Linda Northrop. 2001. *Software Product Lines: Practices and Patterns*. Addison-Wesley.

[15] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2008. Free/Libre Open-Source Software Development: What We Know and What We Do Not Know. *CSUR* 44, 2, Article 7 (2008), 35 pages. doi:10.1145/2089125.2089127

[16] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *VaMoS*. ACM, 173–182. doi:10.1145/2110147.2110167

[17] Krzysztof Czarnecki and Andrzej Wąsowski. 2007. Feature Diagrams and Logics: There and Back Again. In *SPLC*. IEEE, 23–34.

[18] Ferruccio Damiani, Michael Lienhardt, and Luca Paolini. 2022. On Logical and Extensional Characterizations of Attributed Feature Models. *TCS* 912 (2022), 56–80. doi:10.1016/j.tcs.2022.01.016

[19] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS*. Springer, 337–340.

[20] Raphael Pereira de Oliveira, Paulo Anselmo da Mota Silveira Neto, Qi Hong Chen, Eduardo Santana de Almeida, and Iftekhar Ahmed. 2022. Different, Really! A Comparison of Highly-Configurable Systems and Single Systems. *IST* 152 (2022), 107035. doi:10.1016/j.infsof.2022.107035

[21] Nicolas Dintzner, Arie van Deursen, and Martin Pinzger. 2018. FEVER: An Approach to Analyze Feature-Oriented Changes and Artefact Co-Evolution in Highly Configurable Systems. *EMSE* 23, 2 (2018), 905–952. doi:10.1007/s10664-017-9557-6

[22] Kevin Feichtinger, Johann Stöbich, Dario Romano, and Rick Rabiser. 2021. TRAVART: An Approach for Transforming Variability Models. In *VaMoS*. ACM, Article 8, 10 pages. doi:10.1145/3442391.3442400

[23] Alexander Felfernig, David Benavides, Jose A. Galindo, and Florian Reinfrank. 2013. Towards Anomaly Explanation in Feature Models. In *ConfWS*. ceur-ws.org, 117–124.

[24] Johannes K. Fichte, Markus Hecher, and Stefan Szeider. 2020. A Time Leap Challenge for SAT-Solving. In *CP*, Helmut Simonis (Ed.). Springer, 267–285. doi:10.1007/978-3-030-58475-7_16

[25] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2014. Enhancing Clone-and-Own With Systematic Reuse for Developing Software Variants. In *ICSME*. IEEE, 391–400. doi:10.1109/ICSME.2014.61

[26] Flavio S. Fogliatto, Giovani J.C. da Silveira, and Denis Borenstein. 2012. The Mass Customization Decade: An Updated Review of the Literature. *Production Economics* 138, 1 (2012), 14–25. doi:10.1016/j.ijpe.2012.03.002

[27] Patrick Franz, Thorsten Berger, Ibrahim Fayaz, Sarah Nadi, and Evgeny Groshev. 2021. ConfigFix: Interactive Configuration Conflict Resolution for the Linux Kernel. In *ICSE-SEIP*. IEEE, 91–100. doi:10.1109/ICSE-SEIP52600.2021.00018

[28] Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. 2021. SAT Competition 2020. *AIJ* 301 (2021), 103572. doi:10.1016/j.artint.2021.103572

[29] José A Galindo, Mathieu Acher, Juan Manuel Tirado, Cristian Vidal, Benoit Baudry, and David Benavides. 2016. Exploiting the Enumeration of All Feature Model Configurations: A New Perspective With Distributed Computing. In *SPLC*. ACM, 74–78.

[30] Matthias Galster, Danny Weyns, Dan Tofan, Bartosz Michalik, and Paris Avgeriou. 2013. Variability in Software Systems-a Systematic Literature Review. *TSE* 40, 3 (2013), 282–306.

[31] Alireza Haghighatkhah, Ahmad Banijamali, Olli-Pekka Pakanen, Markku Oivo, and Pasi Kuvaja. 2017. Automotive Software Engineering: A Systematic Mapping Study. *JSS* 128 (2017), 25–55. doi:10.1016/j.jss.2017.03.005

[32] Marc Hentze, Chico Sundermann, Thomas Thüm, and Ina Schaefer. 2022. Quantifying the Variability Mismatch Between Problem and Solution Space. In *MODELS*. IEEE, 322–333. doi:10.1145/3550355.3552411

[33] Tobias Heß, Simon Karrer, and Lukas Ostheimer. 2024. Multi-Version Decision Propagation for Configuring Feature Models in Space and Time. In *SPLC*. ACM, 88–92. doi:10.1145/3646548.3676550

[34] Tobias Heß, Chico Sundermann, and Thomas Thüm. 2021. On the Scalability of Building Binary Decision Diagrams for Current Feature Models. In *SPLC*. ACM, 131–135. doi:10.1145/3461001.3474452

[35] Mikoláš Janota. 2008. Do SAT Solvers Make Good Configurators?. In *SPLC*, Vol. 2. University of Limerick, Lero, 191–195.

[36] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. 2012. The International SAT Solver Competitions. *AI Magazine* 33, 1 (2012), 89–92. doi:10.1609/aimag.v33i1.2395

[37] Jianxin Jiao, Timothy W Simpson, and Zahed Siddique. 2007. Product Family Design and Platform-Based Product Development: A State-of-the-Art Review. *Intelligent Manufacturing* 18 (2007), 5–29.

[38] Florian Jost and Carsten Sinz. 2024. Challenges in Automotive Hardware-Software Co-Configuration. In *ConfWS*. ceur-ws.org.

[39] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. Software Engineering Institute.

[40] Christian Kästner. 2017. *Differential Testing for Variational Analyses: Experience From Developing KConfigReader*. Technical Report arXiv:1706.09357. Cornell University Library.

[41] Christian Kästner, Paolo G. Giarrusso, Tillmann Rendel, Sebastian Erdweg, Klaus Ostermann, and Thorsten Berger. 2011. Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation. In *OOPSLA*. ACM, 805–824. doi:10.1145/2048066.2048128

[42] Chang Hwan Peter Kim, Don Batory, and Sarfraz Khurshid. 2011. Reducing Combinatorics in Testing Product Lines. In *AOSD*. ACM, 57–68. doi:10.1145/1960275.1960284

[43] Chang Hwan Peter Kim, Christian Kästner, and Don Batory. 2008. On the Modularity of Feature Interactions. In *GPCE*. ACM, 23–34. doi:10.1145/1449913.1449919

[44] Peter Knauber, Jesús Bermejo Muñoz, Günter Böckle, Julio Cesar Sampaio do Prado Leite, Frank van der Linden, Linda Northrop, Michael Stark, and David M. Weiss. 2001. Quantifying Product Line Benefits. In *PFE*. Springer, 155–163. doi:10.1007/3-540-47833-7_15

[45] Alexander Knüppel, Thomas Thüm, Stephan Mennicke, Jens Meinicke, and Ina Schaefer. 2017. Is There a Mismatch Between Real-World Feature Models and Product-Line Research?. In *ESEC/FSE*. ACM, 291–302. doi:10.1145/3106237.3106252

[46] Matthias Kowal, Sofia Ananieva, and Thomas Thüm. 2016. Explaining Anomalies in Feature Models. In *GPCE*. ACM, 132–143. doi:10.1145/2993236.2993248

[47] Sebastian Krieter, Rahel Arens, Michael Nieke, Chico Sundermann, Tobias Heß, Thomas Thüm, and Christoph Seidl. 2021. Incremental Construction of Modal Implication Graphs for Evolving Feature Models. In *SPLC*. ACM, 64–74. doi:10.1145/3461001.3471148

[48] Sebastian Krieter, Marcus Pinnecke, Jacob Krüger, Joshua Sprey, Christopher Sontag, Thomas Thüm, Thomas Leich, and Gunter Saake. 2017. FeatureIDE: Empowering Third-Party Developers. In *SPLC*. ACM, 42–45. doi:10.1145/3109729.3109751

[49] Sebastian Krieter, Reimar Schröter, Thomas Thüm, Wolfram Fenske, and Gunter Saake. 2016. Comparing Algorithms for Efficient Feature-Model Slicing. In *SPLC*. ACM, 60–64. doi:10.1145/2934466.2934477

[50] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Reimar Schröter, and Gunter Saake. 2018. Propagating Configuration Decisions With Modal Implication Graphs. In *ICSE*. ACM, 898–909. doi:10.1145/3180155.3180159

[51] Christian Kröher, Lea Gerling, and Klaus Schmid. 2023. Comparing the Intensity of Variability Changes in Software Product Line Evolution. *JSS* 203 (2023), 111737. doi:10.1016/j.jss.2023.111737

[52] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin. 2010. Model Counting in Product Configuration. In *LoCoCo*. Open Publishing Association, 44–53. doi:10.4204/EPTCS.29.5

[53] Elias Kuiter, Tobias Heß, Chico Sundermann, Sebastian Krieter, Thomas Thüm, and Gunter Saake. 2024. How Easy Is SAT-Based Analysis of a Feature Model?. In *VaMoS*. ACM, 149–151. doi:10.1145/3634713.3634733

[54] Elias Kuiter, Alexander Knüppel, Tabea Bordis, Tobias Runge, and Ina Schaefer. 2022. Verification Strategies for Feature-Oriented Software Product Lines. In *VaMoS*. ACM, 12:1–12:9. doi:10.1145/3510466.3511272

[55] Elias Kuiter, Sebastian Krieter, Chico Sundermann, Thomas Thüm, and Gunter Saake. 2022. Tseitin or not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses. In *ASE*. ACM, 110:1–110:13. doi:10.1145/3551349.3556938

[56] Elias Kuiter, Sebastian Krieter, Chico Sundermann, Thomas Thüm, and Gunter Saake. 2023. Tseitin or not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses. In *SE*, Vol. P-322. Gesellschaft für Informatik, 83–84.

[57] Elias Kuiter, Jacob Krüger, and Gunter Saake. 2021. Iterative Development and Changing Requirements: Drivers of Variability in an Industrial System for Veterinary Anesthesia. In *VariVolution*. ACM, 113–122. doi:10.1145/3461002.3473950

[58] Elias Kuiter, Chico Sundermann, Thomas Thüm, Tobias Heß, Sebastian Krieter, and Gunter Saake. 2025. How Configurable is the Linux Kernel? Analyzing Two Decades of Feature-Model History. *TOSEM* (2025). doi:10.1145/3729423 To appear.

[59] Meir M Lehman, Juan F Ramil, Paul D Wernick, Dewayne E Perry, and Wladyslaw M Turski. 1997. Metrics and Laws of Software Evolution–The Nineties View. In *METRICS*. IEEE, 20–32.

[60] Jia Hui Liang, Vijay Ganesh, Krzysztof Czarnecki, and Venkatesh Raman. 2015. SAT-Based Analysis of Large Real-World Feature Models Is Easy. In *SPLC*. Springer, 91–100.

[61] Jörg Liebig, Alexander von Rhein, Christian Kästner, Sven Apel, Jens Dörre, and Christian Lengauer. 2013. Scalable Analysis of Variable Software. In *ESEC/FSE*. ACM, 81–91. doi:10.1145/2491411.2491437

[62] Michael Lienhardt, Ferruccio Damiani, Einer Broch Johnsen, and Jacopo Mauro. 2020. Lazy Product Discovery in Huge Configuration Spaces. In *ICSE*. ACM, 1509–1521. doi:10.1145/3377811.3380372

[63] Rafael Lotufo, Steven She, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wąsowski. 2010. Evolution of the Linux Kernel Variability Model. In *SPLC*. Springer, 136–150.

[64] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability With FeatureIDE*. Springer. doi:10.1007/978-3-319-61443-4

[65] Marcílio Mendonça, Andrzej Wąsowski, and Krzysztof Czarnecki. 2009. SAT-Based Analysis of Feature Models Is Easy. In *SPLC*. Software Engineering Institute, 231–240.

[66] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory. 2019. Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features. In *SPLC*. ACM, 289–301. doi:10.1145/3336294.3336297

[67] Damir Nešić, Jacob Krüger, Stefan Stănciulescu, and Thorsten Berger. 2019. Principles of Feature Modeling. In *ESEC/FSE*. ACM, 62–73. doi:10.1145/3338906.3338974

[68] Michael Nieke, Jacopo Mauro, Christoph Seidl, Thomas Thüm, Ingrid Chieh Yu, and Felix Franzke. 2018. Anomaly Analyses for Feature-Model Evolution. In *GPCE*. ACM, 188–201. doi:10.1145/3278122.3278123

[69] Linda Northrop and P. Clements. 2012. *A Framework for Software Product Line Practice, Version 5.0.* Technical Report. Software Engineering Institute.

[70] Jeho Oh, Paul Gazzillo, Don Batory, Marijn Heule, and Margaret Myers. 2020. *Scalable Uniform Sampling for Real-World Software Product Lines.* Technical Report TR-20-01. University of Texas at Austin, Department of Computer Science.

[71] Leonardo Passos and Krzysztof Czarnecki. 2014. A Dataset of Feature Additions and Feature Removals From the Linux Kernel. In *MSR*. ACM, 376–379.

[72] Leonardo Passos, Marko Novakovic, Yingfei Xiong, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wąsowski. 2011. A Study of Non-Boolean Constraints in Variability Models of an Embedded Operating System. In *FOSD (SPLC)*. ACM, Article 2. doi:10.1145/2019136.2019139

[73] Leonardo Passos, Leopoldo Teixeira, Nicolas Dintzner, Sven Apel, Andrzej Wąsowski, Krzysztof Czarnecki, Paulo Borba, and Jianmei Guo. 2016. Coevolution of Variability Models and Related Software Artifacts. *EMSE* 21, 4 (2016). doi:10.1007/s10664-015-9364-x

[74] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. 2010. Automated and Scalable T-Wise Test Case Generation Strategies for Software Product Lines. In *ICST*. IEEE, 459–468. doi:10.1109/ICST.2010.43

[75] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer. doi:10.1007/3-540-28901-1

[76] Hendrik Post and Carsten Sinz. 2008. Configuration Lifting: Verification Meets Software Configuration. In *ASE*. IEEE, 347–350.

[77] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *RE*. IEEE, 136–145. doi:10.1109/RE.2006.23

[78] Reimar Schröter, Sebastian Krieter, Thomas Thüm, Fabian Benduhn, and Gunter Saake. 2016. Feature-Model Interfaces: The Highway to Compositional Analyses of Highly-Configurable Systems. In *ICSE*. ACM, 667–678. doi:10.1145/2884781.2884823

[79] Sergio Segura. 2008. Automated Analysis of Feature Models Using Atomic Sets. In *SPLC*, Vol. 2. IEEE, 201–207.

[80] Till Sehlen. 2023. *Evaluating the Efficiency of Hybrid CNF Transformations for Feature-Model Formulas.* Master's Thesis. University of Magdeburg.

[81] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. 2010. The Variability Model of the Linux Kernel. In *VaMoS*. 45–51.

[82] Janet Siegmund, Norbert Siegmund, and Sven Apel. 2015. Views on Internal and External Validity in Empirical Software Engineering. In *ICSE*. IEEE, 9–19.

[83] Julio Sincero, Horst Schirmeier, Wolfgang Schröder-Preikschat, and Olaf Spinczyk. 2007. Is the Linux Kernel a Software Product Line?. In *OSSPL*. IEEE, 9–12.

[84] Chico Sundermann, Vincenzo Francesco Brancaccio, Elias Kuiter, Sebastian Krieter, Tobias Heß, and Thomas Thüm. 2024. Collecting Feature Models from the Literature: A Comprehensive Dataset for Benchmarking. In *SPLC*. ACM, 54–65. doi:10.1145/3646548.3672590

[85] Chico Sundermann, Tobias Heß, Michael Nieke, Paul Maximilian Bittner, Jeffrey M. Young, Thomas Thüm, and Ina Schaefer. 2023. Evaluating State-of-the-Art #SAT Solvers on Industrial Configuration Spaces. *EMSE* 28, 2 (2023), 38. doi:10.1007/s10664-022-10265-9

[86] Chico Sundermann, Elias Kuiter, Tobias Heß, Heiko Raab, Sebastian Krieter, and Thomas Thüm. 2024. On the Benefits of Knowledge Compilation for Feature-Model Analyses. *AMAI* 92, 5 (2024), 1013–1050. doi:10.1007/s10472-023-09906-6

[87] Chico Sundermann, Elias Kuiter, Tobias Heß, Heiko Raab, Sebastian Krieter, and Thomas Thüm. 2024. On the Benefits of Knowledge Compilation for Feature-Model Analyses. In *SPLC*. ACM, 217. doi:10.1145/3646548.3676540

[88] Chico Sundermann, Jacob Loth, and Thomas Thüm. 2024. Efficient Slicing of Feature Models via Projected d-DNNF Compilation. In *ASE*. ACM, 1332–1344. doi:10.1145/3691620.3695594

[89] Chico Sundermann, Michael Nieke, Paul Maximilian Bittner, Tobias Heß, Thomas Thüm, and Ina Schaefer. 2021. Applications of #SAT Solvers on Feature Models. In *VaMoS*. ACM, Article 12, 10 pages. doi:10.1145/3442391.3442404

[90] Chico Sundermann, Heiko Raab, Tobias Heß, Thomas Thüm, and Ina Schaefer. 2024. Reusing d-DNNFs for Efficient Feature-Model Counting. *TOSEM* 33, 8, Article 208 (2024), 32 pages. doi:10.1145/3680465

[91] Reinhard Tartler, Daniel Lohmann, Julio Sincero, and Wolfgang Schröder-Preikschat. 2011. Feature Consistency in Compile-Time-Configurable System Software: Facing the Linux 10,000 Feature Problem. In *EuroSys*. ACM, 47–60. doi:10.1145/1966445.1966451

[92] Christian Thiffault, Fahiem Bacchus, and Toby Walsh. 2004. Solving Non-Clausal Formulas With DPLL Search. In *CP*. Springer, 663–678.

[93] Thomas Thüm. 2020. A BDD for Linux? The Knowledge Compilation Challenge for Variability. In *SPLC*. ACM, Article 16, 6 pages. doi:10.1145/3382025.3414943

[94] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *CSUR* 47, 1 (2014), 6:1–6:45. doi:10.1145/2580950

[95] Thomas Thüm, Don Batory, and Christian Kästner. 2009. Reasoning About Edits to Feature Models. In *ICSE*. IEEE, 254–264. doi:10.1109/ICSE.2009.5070526

[96] Thomas Thüm, Ina Schaefer, Sven Apel, and Martin Hentschel. 2012. Family-Based Deductive Verification of Software Product Lines. In *GPCE*. ACM, 11–20. doi:10.1145/2371401.2371404

[97] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer.

[98] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. 2018. A Classification of Product Sampling for Software Product Lines. In *SPLC*. ACM, 1–13. doi:10.1145/3233027.3233035

[99] Niklaus Wirth. 1995. A Plea for Lean Software. *IEEE Computer* 28, 2 (1995), 64–68. doi:10.1109/2.348001

[100] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. 2012. *Experimentation in Software Engineering*. Springer. doi:10.1007/978-3-642-29044-2

[101] Len Wozniak and Paul Clements. 2015. How Automotive Engineering Is Taking Product Line Engineering to the Extreme. In *SPLC (SPLC '15)*. ACM, 327–336. doi:10.1145/2791060.2791071