Benchmark Generation with VEVOS: A Coverage Analysis of Evolution Scenarios in Variant-Rich Systems

Alexander Schultheiß
Humboldt-Universität zu Berlin
Germany
alexander.schultheiss@informatik.hu-berlin.de

Paul Maximilian Bittner
University of Ulm
Germany
paul.bittner@uni-ulm.de

Sandra Greiner
University of Bern
Switzerland
sandra.greiner@unibe.ch

Timo Kehrer
University of Bern
Switzerland
timo.kehrer@unibe.ch

ABSTRACT

Clone-and-own development is a simple and flexible approach to realize multi-variant software systems in practice but typically provokes costly challenges in maintaining a continuously evolving set of variants. Therefore, managed clone-and-own development is key to efficiently mitigate these problems. While supporting techniques have been proposed in the literature, hardly any of them have been evaluated in a realistic setting due to a substantial lack of publicly available clone-and-own projects which could be used as experimental subjects. Recently, we presented the benchmark generation framework VEVOS for simulating clone-and-own development. However, it is yet unclear to which extent VEVOS can cover key scenarios for evaluating evolving variant-rich systems. This paper examines to what extent benchmarks created by VEVOS satisfy evaluation requirements for evolution scenarios demanded by the community. In addition, we report on our own experiences when employing VEVOS within six studies and elaborate on necessary extensions we implemented into VEVOS.

CCS CONCEPTS

• **Software and its engineering → Software configuration management and version control systems**; • **General and reference → Evaluation**.

KEYWORDS

Clone-and-own, software product lines, empirical evaluation

ACM Reference Format:
Alexander Schultheiß, Paul Maximilian Bittner, Sandra Greiner, and Timo Kehrer. 2023. Benchmark Generation with VEVOS: A Coverage Analysis of Evolution Scenarios in Variant-Rich Systems. In *17th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2023), January 25–27, 2023, Odense, Denmark.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3571788.3571793

1 INTRODUCTION

To meet specific customer requirements, today's software systems compose a significant amount of inherent variability. Software product line engineering offers a systematic solution to customize and reuse software by relying on variability expressed in form of features which describe common and variable parts of such variant-rich systems [3, 32]. Despite the promised benefits, such as reduced maintenance costs, practitioners frequently employ the ad-hoc principle of copying an existing variant and modifying it to meet new requirements. This development strategy, known as clone-and-own [2, 7, 34, 46], may be chosen to reduce the time to market and high upfront investments [2, 7, 20, 21, 34] or because the need for systematic variability support is unknown at the beginning of development. While not requiring upfront investments, maintaining a family of cloned variants without automated support becomes infeasible once a critical number of variants is reached [2].

Given this observation, recent research investigates the continuum between software product line engineering and ad-hoc clone-and-own development. The efforts for supporting *managed* clone-and-own range from the *controlled* generation of new variants [19, 22, 34] and the synchronization of evolving cloned variants [13, 24] to the migration of clones to a software product line [8, 14, 18, 25], or systematic support for variant development in terms of filtered product lines [41] or variation control systems [20, 23, 42, 45, 48].

While these research efforts promise beneficial maintenance support, they are hardly evaluated exhaustively due to a substantial lack of demanding evaluation data. On the one hand, this fact is caused by missing openly available clone-and-own projects [37]. On the other hand, existing repositories miss a *ground truth* in form of explicit variation-aware data, such as feature mappings or explicit feature models [37]. For that reason, it is indispensable to come up with suitable benchmarks to thoroughly evaluate managed clone-and-own approaches. The community agrees that scenarios for benchmarking variant-rich systems should include the synchronization and integration of variants, the extraction of features and synthesis of a feature model, as well as the recovery of the architecture and testing scenarios, among others [44].

Recently, we proposed the benchmark generator VEVOS [37] as a framework to simulate clone-and-own development. In the proposed state, VEVOS promises to extract ground truth data from a C preprocessor-based software product line repository, such as the Linux kernel, and simulates its evolution by generating variants across the version history. VEVOS provides the required ground truth data in terms of a feature model and presence conditions [3] for each revision. The variant generation samples the feature models to generate *feature-aware* variants, comprising configurations, feature mappings, presence conditions, and traces to the original product line's code. While VEVOS claims to simulate clone-and-own development and to produce beneficial benchmark data, it is yet unclear to what extent VEVOS can satisfy key requirements for evaluating evolving variant-rich systems as demanded by the software product line engineering community [44].

This paper evaluates the capabilities of the benchmark generation framework VEVOS and reports current experiences. Particularly, we qualitatively evaluate to which extent VEVOS covers the requirements of scenarios postulated for evaluating evolving variant-rich systems [44] in Sec. 3. This includes scenarios for variant-rich systems that do not use clone-and-own development (i.e., SPLs). In addition, we report experiences from past and ongoing studies that use VEVOS, and report on required extensions to the framework we implemented in Sec. 4.

2 MOTIVATION AND RELATED WORK

Managed clone-and-own resides between the two extremes of clone-and-own development and systematic software product line engineering. This section briefly describes the difference between both approaches, first. Second, it illuminates the state of the art in evaluating prototypes which support the maintenance of variant-rich systems, in general, and of managed clone-and-own, in particular. The latter concludes that a substantial lack of thorough evaluation of the prototypes exists because of missing demanding benchmarks. Lastly, we give an overview of VEVOS [37].

2.1 Clone-and-Own Development

The workflow of clone-and-own is a practical straightforward approach. Given a new requirement, existing source code is copied and modified to satisfy the new requirement. When developing and maintaining multiple variants with a version control system, an existing branch is cloned and the requirements of the new variant are added to the cloned branch. As a branch is typically associated with one variant, the development of each variant may run in parallel and without merging. Despite the flexibility and speed of integrating a new feature in the short-term, maintaining such an evolving variant-rich repository may prove complex and costly in the long-term [49]. Research on *managed* clone-and-own development [5, 13, 16, 22, 24, 29, 52] addresses problems, such as synchronizing common changes [5, 6, 12, 13, 31, 33, 34, 36] on branches or migrating to an integrated platform as a product line [8, 14, 18].

2.2 Software Product Line Engineering

In contrast to clone-and-own, software product line engineering defines a set of similar software variants systematically and upfront. The development is split into two major phases, referred to as domain engineering and application engineering [3].

At the level of domain engineering, a variability model, such as a feature model [3], declares common conceptual features which are shared between different subsets of variants. Furthermore, a variability mechanism establishes a common platform to implement, maintain, and compose the different features. Preprocessor directives (e.g., `#if`, `#ifdef`, etc) are a common means to implement an annotative variability mechanism [3]. The resulting superimposed source code is divided into *blocks* which may contain *block conditions* declared by the directives. We refer to the directive constraining a code block as *feature mapping*. If conditional blocks are nested, they implement a feature interaction which requires a certain combination of features to be fulfilled for the presence in a derived variant. Besides, build files may comprise *file conditions* which constrain the presence of the entire file. File and block conditions combined are denoted as *presence conditions* [3].

In the application engineering, a variant of the product line is derived, at best automatically. Given a total configuration of the variability model (i.e., a selection or deselection of every feature satisfying the variability model's constraints), a generator derives a variant according to the employed variability mechanism. For example, when preprocessor directives are chosen as variability mechanism, as described above, a preprocessor takes the role of the generator to resolve all directives before the compilation of the actual variant. The preprocessor includes only files and code blocks whose presence condition are satisfied by the given configuration.

2.3 Evaluating Research on Evolving Variant-Rich Systems

As every software, also variant-rich systems are subject to frequent change during software development. While research on maintaining managed and unmanaged clone-and-own has advanced, it lacks suitable evaluation data which is in turn necessary for the research's adoption. Strüber et al. [44] recognized this fact and conducted a community survey to identify requirements for benchmarks and classified existing benchmarks according to the identified requirements. They proposed a catalog of evolution scenarios that should be covered by benchmarks (cf., Sec. 3). Out of the 17 existing benchmark candidates proposed by the survey participants, only nine were considered suitable by Strüber et al. [44]. However, we found that they cannot serve to evaluate *evolving* variant-rich systems thoroughly [37], because they encompass either only a low number of revisions (e.g., [1, 27, 35, 51]) or no revisions at all (e.g., [11, 17, 26]).

Similar restrictions apply to other benchmark candidates proposed in the literature. The DoSC [53] data set comprises no variants, but only histories of independent subject systems. Berger et al. [4] collected a set of 128 feature models from open-source projects, but without source code and presence conditions. The ClaferWebTools benchmark comprises only academic projects developed by students [10]. A more recent benchmark by Michelon et al. [30] targets the evolution of systems in time and space [43] (i.e., version history and existing configurations). Two classes of variant sets exist in their benchmark: Sets addressing evolution in space and sets addressing evolution in time. Therefore, the benchmark contains the variants' source code and history. However, the provided ground truth has no feature model and contains only partial presence conditions; they resolve the nesting of block conditions but do not encompass presence conditions of source files.

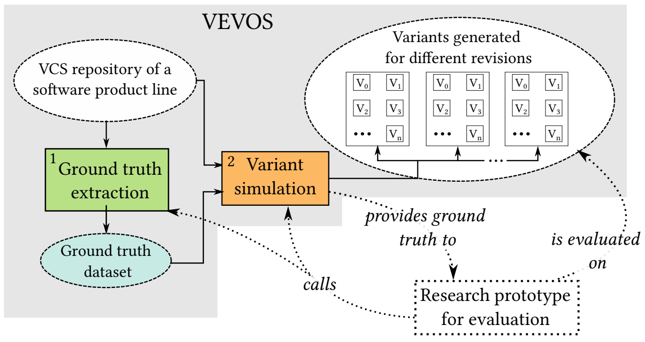
VEVOS
VCS repository of a software product line
1 Ground truth extraction
Ground truth dataset
2 Variant simulation
Variants generated for different revisions
V0 V1 V2 V3 ... Vn
provides ground truth to
is evaluated on
calls
Research prototype for evaluation

Figure 1: Overview of VEVOS.

Furthermore, Wolfart et al. [50] assessed the occurrence of open source software among the subject systems listed in the ESPLA catalog [25]. They found that about one third (44) of the case studies in the catalog stem from open source projects, of which more than half (28) comprise source code. According to their findings, the majority of the open-source case studies is used for research on feature model synthesis, feature identification, and feature location. On the contrary, we examined the ESPLA catalog in January 2022 in search for benchmarks to evaluate research on clone-and-own [37]. Thereby we found that only three out of the 135 case studies, namely, uCLib, BusyBox and the Linux kernel,[1] can serve as proper benchmarks because they comprise variable source code, a version history, and they are publicly available. As explicit feature models and presence conditions are missing in these subjects, additional tool support is necessary to extract this information which is required to generate ground truth data.

For simulating clone-and-own development by generating benchmarks, we proposed to extract ground truth data and employ it to generate new variants of the extracted feature models and annotated source code files [37]. We give an overview of our benchmark generation framework called VEVOS in the following section. Up to this point, despite our introduction of the tool as benchmark generation framework, we conducted only a preliminary study to demonstrate the technical feasibility of our approach. From a conceptual point of view, it is yet unclear to which extent VEVOS is suitable in evaluating research on evolving variant-rich systems. Therefore, we examine VEVOS' capabilities and check whether it meets the requirements for thorough benchmarks postulated by the community [44].

2.4 The VEVOS Framework

The benchmark generation framework VEVOS targets the simulation of clone-and-own development. Fig. 1 provides an overview of VEVOS which consists of two main building blocks, the ground truth extraction and the variant simulation.

Ground-Truth Extraction. To simulate an evolving variant-rich system, such as a clone-and-own project, VEVOS first extracts variability information from the version history of a software product line in which variability is implemented with the C preprocessor. The ground truth extraction retrieves a feature model, feature mappings, and presence conditions at each revision within a specified range of the commit history.[2] VEVOS exports all this data to a single dataset. Internally, the ground truth extraction builds on tooling for the analyses of software product lines, including KernelHaven, CodeBlockExtractor, KBuildMiner, and KConfigReader, explained in detail in the original VEVOS paper [37].

Variant Simulation. Given a dataset provided by the ground truth extraction, the second key component of VEVOS, the variant simulation, simulates the evolution of different variants of the original software product line. Therefore, the variant simulation can generate variants at each revision of the input history by interpreting and resolving conditional preprocessor directives [37]. The generated variants are *feature-aware* in the sense that they consist not only of source code but also the variant's configuration, feature mappings, and presence conditions. These feature mappings and presence conditions differ from those in the input product line because variants are generated by removing certain code blocks, causing offsets in the source code's line numbers. To this end, VEVOS also provides a matching of each variant's source code line numbers and the source code line numbers in the respective revision of the product line.

To generate variants, the variant simulation provides an extensible mechanism for sampling configurations (i.e., selections of features). This mechanism decides at each revision which configurations should be used to generate variants. The default strategy is to randomly sample the feature model at each revision to obtain random variants. Alternatively, a custom, predefined set of configurations may be used, or custom strategies may be implemented.

3 EVOLUTION SCENARIO COVERAGE

3.1 Coverage Assessment

In this section, we qualitatively assess VEVOS' capabilities to generate benchmarks for common evolution scenarios of variant-rich systems. As a community effort, Strüber et al. [44] distilled eleven key scenarios of evolving variant-rich systems. The authors identify key requirements a benchmark should meet to empirically evaluate a respective scenario. We discuss VEVOS' suitability as a benchmark generator for all proposed scenarios. In Table 1, we summarize all scenarios and benchmark capabilities as classified by Strüber et al. [44] as well as the capabilities of VEVOS [37]. Following the notation of Strüber et al., ● marks a scenario as fully supported by VEVOS, ◓ as partially supported, and ○ as not supported.

The first two evolution scenarios consider clone-and-own development and the migration to an integrated platform:

● **1.Variant Synchronization (VS) [13]** targets the synchronization of variants with respect to common code when a change to one variant occurred. For instance, if the code belonging to a certain feature has been modified, the change should be propagated to all variants implementing the affected feature. A benchmark is required to contain the source code of variants before and after the propagation of changes to be able to evaluate custom change propagation operators. Therefore, the entire benchmark requires [44]:

● **R1.1** Source code of at least two variants

[1] https://www.uclibc.org/, https://busybox.net/, https://github.com/torvalds/linux

[2] While the input repository may consists of multiple branches only the main branch is used for the ground truth extraction.

Table 1: Benchmark capabilities as classified by Strüber et al. [44] (gray background) compared to our assessment of VEVOS [37]

Benchmark	Original Context	VS	VI	FIL	CE	FMS	AR	TR	FT	ANF	VZ	CPS
ArgoUML-SPL FLBench [26]	Feature location	○	○	◓	○	○	○	○	○	○	○	○
Drupal [35]	Bug detection	○	○	○	○	○	○	○	●	○	◓	○
Eclipse FLBench [27]	Feature location	○	○	◓	◓	◓	○	○	○	○	○	○
LinuxKernel FLBench [51]	Feature location	○	○	◓	◓	◓	○	○	○	○	○	○
Marlin & BCWallet [17]	Feature location	○	○	◓	○	○	○	○	○	○	◓	○
ClaferWebTools [10]	Traceability	○	○	◓	○	◓	○	○	○	○	◓	○
DoSC [53]	Change discovery	○	◓	◓	○	◓	○	◓	○	○	◓	○
SystemsSwVarModels [4]	FM synthesis	○	◓	○	●	●	○	○	○	○	◓	○
TraceLab CoEST [11]	Traceability	○	○	◓	○	○	○	○	○	○	○	○
Variability bug database [1]	Bug detection	○	○	◓	○	○	○	○	●	○	◓	○
VEVOS [37]	Clone-and-Own	●	●	●	◓	●	○	●	○	○	○	●

- ● **R1.2** Ground-truth source code of variants after correct propagation
- ● **R1.3** Modifications occurring in at least one variant
- ● **R1.4** Feature locations of at least one variant

As the filled circle ● shows, VEVOS satisfies all requirements: It generates the source code of any desired amount of variants (**R1.1**) and can simulate the evolution of variants: The variant simulation library implements the evolution of the product line in single evolution steps between two commits of the input product line. Thereby, the source code of a variant at the next evolution step serves as the ground truth for the result of a propagation and synchronization technique (**R1.2**). By diffing the state of the variant before and after the evolution step, a set of changes can be obtained to meet requirement **R1.3**. The location of the feature is part of the ground truth (**R1.4**).

● **2.Variant Integration (VI)** represents the key task in *extractive software product-line engineering* [15] or *variant-preserving migration* [9]. It aims to integrate the variants of a clone-and-own project into a common code platform to migrate to a software product line. The overall task may be split into two sub-tasks: First, integrating a given set of products to an integrated platform. Second, due to evolution, it may be necessary to iteratively integrate further variants into the platform. The latter allows to still develop single products and is a typical scenario occurring in variation control systems. Thus, variant integration benchmarks require [44]:

- ● **R2.1** Set of individual variants
- ● **R2.2** Set of revisions of a software product line
- ● **R2.3** Ground-truth software product line after correct integration

As discussed for variant synchronization, VEVOS may provide an arbitrary amount of variants from each revision of the product line history, which satisfies **R2.1** and **R2.2**. Secondly, the state of the product line after a correct integration is available by retrieving the product line's code base at the subsequent evolution step (**R2.3**). However, the product line comprises the correct integration of *all* possible variants, and researchers may require the correct integration of only a subset of variants instead. To this end, the variant simulation of VEVOS supports the derivation of partial variants from partial configurations. A partial variant describes an intermediate state where the in- or exclusion of features into a variant is decided for some but not all features of the product line. Thus, partial variants describe a subset of all valid variants and can be used as the correct integration for a subset of variants into the product line. Thus, VEVOS satisfies the third requirement and, consequently, all requirements for the variant integration benchmark.

● **3.Feature Identification and Location (FIL)** describe the tasks of identifying the names of implemented features, and locating their implementation, respectively. Thus, feature location aims to detect forgotten or unspecified feature mappings and presence conditions. Feature identification and location are a crucial step for variant synchronization and integration. A benchmark for evaluating corresponding techniques requires [44]:

- ● **R3.1** Variant assets, such as version-control history, source code, requirements, documentation, change logs, or issue tracker data
- ● **R3.2** Ground truth feature list
- ● **R3.3** Ground-truth feature mappings or presence conditions

The variant simulation of VEVOS provides the source code of generated variants, the initial commit messages of the input software product line (change logs), as well as any inline documentation (e.g., comments), all across the version-control history. While VEVOS cannot extract further assets, such as requirements, documentation, and issue tracker data, it could be implemented as a future extension. Since the variants' source code is the most relevant input data for feature identification and location, we consider **R3.1** as satisfied. VEVOS fulfills **R3.2** because it provides the configuration of each generated variant as well as the product line's feature model at each evolution step. As VEVOS provides feature mappings and presence conditions for files and code blocks, it also satisfies **R3.3**.

◓ **4.Constraints Extraction (CE)** is a subsequent task to feature identification in which the constraints and dependencies between features are derived from any assets within cloned variants, notably their source code. Sub-tasks include the extraction of constraints either from exemplary configurations, source code or natural-language assets. Constraint extraction is an intermediate step towards feature model synthesis (next paragraph) to shrink the variant space and enable analyses, even if a full feature model is unavailable. A benchmark for constraints extraction requires [44]:

- ● **R4.1** Set of configurations
- ● **R4.2** Source code of at least one variant
- ◓ **R4.3** Natural-language assets, such as documentation
- ● **R4.4** Ground-truth constraints formula

VEVOS uses the FeatureIDE library to provide random sampling of configurations from the extracted feature models and also supports pre-defined sets of configurations, thus, satisfying **R4.1**. As before, **R4.2** is also satisfied. Feature models encode all constraints on features. Therefore, the feature models extracted by VEVOS can serve as ground truth, thus, satisfying **R4.4**.

Regarding **R4.3**, VEVOS provides two kinds of natural language assets. First, VEVOS extracts all commit messages of the input software product line. Second, inline documentation of the input source code (e.g., comments on methods) of the product line is included in the respective generated variants. While the input software product line may contain further natural language assets, they are not related with and integrated into variant-specific assets by VEVOS. In conclusion, constraint extraction from configurations, source code, and some natural-language assets is possible and can be validated against the feature model.

● **5.Feature Model Synthesis (FMS)** describes the task of eventually deriving a feature model from variants or their configurations. As the feature model is mainly computed from extracted constraints, the requirements to benchmarks for feature model synthesis match those for constraint extraction mostly [44]:

- ● **R5.1** Set of configurations
- ● **R5.2** Source code of at least one variant
- ● **R5.3** Product matrix
- ● **R5.4** Ground-truth feature model

For the same reasons explained for the *constraint extraction* benchmark, VEVOS satisfies requirements **R5.1**, **R5.2** and **R5.4**. A product matrix maps features (rows) to their variants (columns), thereby describing the entirety of configurations of all variants. As a consequence, a product matrix can be computed from (e.g., randomly sampled) configurations as done by VEVOS (**R5.3**). Hence, VEVOS supports all requirements for feature model synthesis benchmarks.

○ **6.Architecture Recovery (AR)** identifies architecture models in source code. Such models describe the architecture of a code base at the level of modules and their relations, among others. Benchmark requirements for techniques on architecture recovery are [44]:

- ● **R6.1** Source code of at least one variant
- ● **R6.2** Source code of a software product line
- ○ **R6.3** Ground-truth architectural models

As software architecture is not a design goal of VEVOS, it is not supported. While VEVOS generates the source code from an input software product line (thus, meeting requirements **R6.1** and **R6.2**), it does not provide architectural models as ground truth (**R6.3**).

● **7.Transformations (TR)** refer to the automated transformation of source code, ranging from lightweight refactorings (e.g., consistent renaming of variables across the code base) to model transformations. Requirements to benchmarks comprise [44]:

- ● **R7.1** Feature model and source code of software product line
- ● **R7.2** Transformation specification, such as reference implementations
- ● **R7.3** Ground-truth transformed implementation

VEVOS meets all requirements. It satisfies **R7.1** as the ground truth extraction yields the product line's source code and feature model at each revision of the considered sub-history. While transformation specifications (e.g., patches) are not included explicitly in VEVOS' ground truth datasets, it includes reference implementations in terms of variants before and after a transformation. The ground truth for transformations between variants (i.e., two generated variants at the same revision) as well as versions of variants (i.e., the same variant at different points in the evolution history) may be obtained by diffing. In conclusion, **R7.2** and **R7.3** are satisfied in the context of variability, but VEVOS does not serve other transformation scenarios.

○ **8.Functional Testing (FT)** refers to research on any tests for functional requirements of the software system under study, such as unit, regression, or integration tests. In the context of variability in software, bugs might not only arise from errors in source code but also from wrong feature mappings or an ill-formed feature model [1]. The requirements encompass [44]:

- ● **R8.1** Source code of a software product line
- ○ **R8.2** Ground-truth known faults
- ◓ **R8.3** Co-evolving tests cases

As VEVOS is not explicitly designed for testing, testing is not well supported. While **R8.1** is satisfied by the input software product line, no data is available on known faults: VEVOS does not document known bugs and how they are caused (**R8.2**). Nevertheless, as long as the input software product line exposes tests (e.g., unit tests) researchers can still investigate the evolution of tests and their co-evolution with the remaining code base (**R8.3**). If feature mappings for the product line's tests are known, VEVOS will even generate different tests for respective variants.

○ **9.Analysis of Non-Functional Properties (ANF)** addresses requirements such as memory consumption, response time, or safety aspects. Non-functional properties are an orthogonal concern to the systematic support of clone-and-own development. Thus, they are out of the scope of VEVOS and not supported.

○ **10.Visualization (VZ)** aids developers in understanding software but also its variability [28]. While VEVOS offers various data which may be used for visualization tasks, it is not its primary concern to include specific visualizations for the simulation of clone-and-own development. Therefore, it does not meet the requirements postulated for supporting this scenario.

● **11.Co-Evolution of Problem Space and Solution Space (CPS)** refers to any research on how source code, feature mappings, and feature models evolve in parallel in software product lines or clone-and-own projects. A benchmark requires [44]:

- ● **R11.1** Feature model and source code of software product line
- ● **R11.2** Ground-truth revision sequence for feature model and source code

As all this data is available in VEVOS for the input product line and every generated variant, VEVOS satisfies both requirements.

Conclusion. Based on our assessment of VEVOS, summarized in the bottom row of Table 1, we conclude the following:

The coverage analysis demonstrates that VEVOS can generate benchmarks for the majority of evolution scenarios of variant-rich systems: variant synchronization, variant integration, feature identification and location, constraints extraction, and feature model synthesis.

3.2 Threats to Validity

Internal Validity. Our assessment of VEVOS' scenario coverage depends on our understanding of the scenarios defined by Strüber et al. [44]. We might have misinterpreted scenarios or their requirements, and Strüber et al. [44] might not have identified all requirements for a scenario. Furthermore, we assessed the coverage without the involvement of an impartial referee. Thus, our assessment might be biased by our desire to cover as many scenarios as possible. However, we did a conservative assessment: If VEVOS failed to fulfill at least one requirement completely, we rated this scenario as unsupported (as for (AR) and (FT)).

External Validity. VEVOS has certain technical constraints (e.g., the extraction is built for product lines using the C preprocessor). Thus, a scenario being fully-supported does not mean that every possible case of the scenario is supported. Instead, it means that there is at least one case in which VEVOS provides full support. On this note, we based the assessment on our own experiences and subject systems known to us, and there might be cases that fulfill VEVOS' technical constraints, but are still not supported.

4 APPLICATION EXPERIENCES

In this section, we present our experiences with employing VEVOS for clone-and-own research.

4.1 Application in Research Studies

We used VEVOS as a benchmark generation framework in a large-scale empirical study [39] (S1) and in several minor research studies (S2-6). Each of the minor studies is related to a bachelor's or master's thesis and primarily conducted by a single student. Each student had to download and use VEVOS independently and without prior introduction to the tooling. The only sources of information were the original VEVOS paper [37] and the README files of VEVOS' GitHub projects.

An overview of how each study is related to the benchmark scenarios by Strüber et al. [44] is shown in Table 2. Most of the studies investigate variant synchronization (S1, S2, S4, and S6). One scenario investigates variant integration, feature identification and location (S3), and one scenario investigates the transformation of source code through refactoring. In the following, we shortly discuss the studies and the experiences we made.

(S1) Variant Synchronization Study. In this large-scale empirical study, we quantified the potential to automate the synchronization of clone-and-own variants [39]. We investigated to which extent existing context-based patching techniques can be used to propagate changes from one variant to another. Whenever a variant evolved, all changes to the variant are automatically propagated to the remaining variants. In terms of the automation potential, we were interested in the applicability and correctness of patching.

Table 2: Benchmark scenarios by Strüber et al. [44] related to the research studies in which we applied VEVOS.

Study	Variant Synchronization	Variant Integration	Feature Identification and Location	Transformation
S1	✓			
S2	✓			
S3		✓	✓	
S4	✓			
S5	✓			✓
S6	✓			

For conducting the study, we used VEVOS to sample a random set of valid variants at each revision. Due to the considerable runtime overhead of generating all source files for each variant, we extended VEVOS by implementing variant slicing capabilities in terms of a file filter for the variant generation. Based on the filter, it is possible to generate only a desired subset of files for each variant. We consider this to be a useful feature for research in which only some files are of interest (e.g., changed files when considering evolution, or files implementing certain features). In the case of our study, VEVOS only generates files that have changed.

To simulate the evolution of individual variants, we replayed changes from BusyBox' history. We were able to evaluate the correctness of the change propagation with the help of the ground truth provided by VEVOS. In total, we were able to simulate and evaluate half a billion evolution scenarios, which would not have been possible with any of the other benchmarks known to us.

(S2) Conflict Scenarios During Variant Synchronization. In this study, the student investigated conflict scenarios that can arise due to the differences between variants when trying to propagate changes between them. The study focuses on change propagation conflicts that may occur when different features are developed in different variants in parallel.

For this purpose, the student applies VEVOS to simulate the parallel evolution of features on different BusyBox variants. Using the ground truth of VEVOS, the student was able to automatically identify conflicts and their correct resolution. The main drawback, experienced by the student, is that the simulated parallel evolution is artificial. VEVOS considers only the main branch of a product line's repository and omits information about parallel branches. Thus, the investigated conflicts were not directly reflected by BusyBox' history, introducing additional threats to the validity of the results. We conclude that the simulation could be improved by extending VEVOS' functionality so that all development branches are regarded in order to track the parallel evolution of features.

(S3) Utilizing Partial Feature Traces for Product-Line Migration. The goal of this study is to detect the impact of explicit knowledge about feature traces during the migration of clone-and-own variants to a software product line. The student considers a scenario in which developers documented feature mappings for some parts of the variants' code base over time and now want to migrate to a software product line with the help of ECCO [22]. As subject system, the student considers ArgoUML, a modeling tool for UML models.[3] Originally, the student wanted to use the ArgoUML benchmark [26], but they found the benchmark's ground

[3]ArgoUML: https://argouml-tigris-org.github.io/tigris/argouml/

truth unsuitable because it lacks feature mappings and presence conditions for individual source code lines.

Yet, as VEVOS was built for C preprocessor-based product lines, it could not immediately be applied to ArgoUML. Therefore, the student extended VEVOS by a custom ground truth extraction for the JavaPP preprocessor that is used to handle variability in ArgoUML.[4] Using VEVOS, the student was able to generate all 256 ArgoUML variants with an enhanced ground truth containing feature mappings and presence conditions for each line of code.

In retrospective, it appears feasible to extend VEVOS' extraction capabilities to other types of preprocessor-based software product lines. In the case of ArgoUML, the preprocessor annotations are simple and all features are known to be optional. This made the implementation straightforward, leading to an implementation effort of about six hours. The implementation of a ground truth extraction will prove more challenging if the product line employs a more sophisticated build system with conditional compilation and constraints between features. Moreover, the student did not have to change VEVOS' variant simulation, which is empirical evidence that the variant simulation can simulate variants for any kind of preprocessor-based product line, as long as a ground truth dataset is available.

(S4) Matching-based Patch Context Resolution. In this study, the student investigates the effectiveness of code matching techniques during the propagation of changes (e.g., bug fixes, feature changes) between variants. The goal is to apply code matching to determine the correct location for a change. By focusing on the evolution of one variant, changes can be observed that should be propagated to other variants in the system.

We found that VEVOS was perfectly applicable for the scenario investigated by the student and no further changes were required. Similar to our empirical study [39], the student employs VEVOS to simulate the evolution of BusyBox variants, and uses the ground truth provided by VEVOS to determine whether changes in one variant have been propagated correctly to a target variant.

(S5) Effect of Variant Drift on Change Propagation. In this study, the student investigates the effect of unintentional variant drift on automated change propagation. Unintentional variant drift may occur by refactoring code only in some variants although the code is also contained in other variants [38]. To study the effect of variant drift on change propagation, the student employs VEVOS to simulate the evolution of variants of LibSSH, a multiplatform C library for the ssh (secure shell) protocol.

To simulate variant drift, the student tries to integrate automated refactoring operations into VEVOS' simulation process. However, the student faces unresolved technical challenges, because VEVOS' ground truth references locations in a variant based on file paths and line numbers. Some refactorings (e.g., moving functions or files) change the code's structure, thereby, invalidating the feature mappings in VEVOS' ground truth, which references line numbers. This problem has to be addressed before we can fully integrate the simulation of variant drift into VEVOS.

(S6) Applying Product Line Tooling in Clone-and-Own. In this study, the student tries to apply an analysis tool for software product lines to clone-and-own variants to detect potential errors prior to variant synchronization. The key idea is to exploit a technical similarity: Product line tools which analyze source code with preprocessor directives could be applied to variants if the variants also include preprocessor directives. Therefore, we extended VEVOS to optionally embed feature mappings into a generated variant's source code in terms of C preprocessor directives. While preprocessor directives in product lines imply *configurability*, they *document feature mappings* in the variants generated by VEVOS.

The student experienced that in principle, product line tooling might be applicable to clone-and-own variants. However, further dependencies of the product line tool can introduce a technical barrier which is hard to overcome. For example, the dependency of the subject product-line tool on the product-line's build system (here: kconfig) could not be met in generated variants. For such tasks, the variant simulation has to be extended to derive not only the source code but also related artifacts with variability. Whether this is possible for build files, is not yet known to us.

4.2 General Experiences

In this section, we report on general experiences we made during all the previously discussed studies.

Study Integration. Almost all studies using VEVOS use Java as their main implementation language. One study uses C++. The variant simulation of VEVOS is a Java library and could therefore be simply used in other studies implemented in JVM languages. Integrating VEVOS into the C++-based study proved to be challenging and is still work-in-progress. As a workaround, the student implements an adapter for the library in Java, which also controls the execution of their C++ executable. In order to increase the accessibility of the simulation in other programming languages, a rudimentary command line interface for sampling and generating variants should be implemented in the future.

Ground Truth Extraction. The build model analysis is the main challenge of the extraction of a full ground truth that includes a correct feature model, as well as presence conditions for files and code lines. Initially, VEVOS could only extract a ground truth for BusyBox and Linux. To do so, VEVOS uses the KConfigReader and KBuildMiner plugins of KernelHaven. These plugins are not very robust and easily fail because they try to parse files that are only generated when executing Kbuild's build process successfully. Based on our work with VEVOS and by having a closer look at Linux' and BusyBox' build process, we conclude that writing improved extractor plugins is feasible. While the required domain knowledge about the feature model and build system can be read in the generated files, it can also be found in the existing build and configuration files. Here, a more sophisticated parser for Kbuild files could be implemented. By using such a parser, it should be possible to extract the desired domain knowledge for a much larger fraction of the histories of BusyBox and Linux. When inspecting further software product lines, it may be possible to determine further popular build systems and implement more sophisticated parsers for those as well.

As a simple workaround, we extended VEVOS and KernelHaven by removing the need to extract sophisticated feature and build models. This enables the extraction of a partial ground truth for repositories of arbitrary C preprocessor-based software product

[4]JavaPP: https://www.slashdev.ca/javapp/

lines. Instead of crashing when considering a repository besides BusyBox or Linux, the extraction now falls back to a naïve feature model extraction while the build model extraction is omitted. The naïve feature model extraction assumes that all encountered variables in a preprocessor macro are optional features (e.g., when parsing `#if A & B`, the variables `A` and `B` are added as optional features to the feature model as children of an artificial, abstract root feature). While this process does not incorporate any dependencies among features, it serves as a fallback in case extracting required knowledge fails. As a result, VEVOS can now be applied to any C preprocessor-based project by potentially sacrificing some accuracy, and with potential reduction of scenario coverage.

Omitting the build model extraction leads to loss of the knowledge about the condition under which a source file is included in a variant. However, not all product lines make use of their build system to implement variability. Moreover, different projects use different build processes, requiring a custom parser would be needed for each project.

We applied this workaround in scenario (S5). For (S3), we used a custom ground truth extraction that applies the workaround only to the feature model. For the remaining scenarios, we applied VEVOS' original extraction using KBuildMiner and KConfigReader.

Understanding the Data. While conducting the ground truth extraction was no problem, several students struggled with understanding the organization and contents of the extracted dataset. One reason is that the extraction also stored additional (and partially redundant) metadata, and students had a difficult time telling which data was relevant to them and which was not. Another reason is that students had no prior experience with preprocessor-based product lines and were unfamiliar with conditional compilation. This caused misunderstandings regarding the mapping of presence conditions to certain blocks of source code in the software product line. To address these issues, we suggest that a detailed explanation of the extracted data should be included, which provides more insight into how the ground truth's data corresponds to artifacts of the software product line, and how the data could be used. Moreover, we suggest that the community should strive for a unified data format for benchmarks. Such a unified format could lower the difficulty of integrating different benchmarks in research studies.

Variant Simulation. While the simulation of variants is, in principle, possible for any study for which we have a ground truth, not all studies are equally attractive for simulation. The main limiting factor is the time required to extract a ground truth, and once it has been extracted, the time and the disk space required for the generation of the variants. For example, while Linux is a popular instance of a preprocessor-based product line, it is an unattractive subject for variant simulation because of its size. Extracting a ground truth for a single commit requires several minutes, sampling a set of five variants requires about two minutes, and generating the variants also requires several seconds. For most other subjects, on the other hand, it is possible to extract a ground truth for their entire history in less than a day, while sampling and generating variants is a matter of milliseconds.

4.3 Key Takeaways

By employing VEVOS for benchmark generation in six research studies, we collected the following experiences:

First, not all studies need to consider variants in their entirety. For our study on variant synchronization, we implemented variant slicing to concentrate only on changed artifacts of a variant.

Second, we found it challenging to correctly simulate the parallel evolution of variants stemming from a software product line. Considering all of a product line's development branches might improve this simulation.

Third, we were able to extend VEVOS' ground truth extraction to JavaPP-based product lines (i.e., ArgoUML) with little effort. This provides evidence that VEVOS can simulate variants for any kind of preprocessor-based product line.

Fourth, simulating variant drift still poses unresolved technical challenges because structural changes to the artifacts can invalidate the ground truth.

Fifth, some studies might require that the feature mappings are embedded into the variants' source code in form of annotations (e.g., preprocessor directives, comments). We extended VEVOS' generation capabilities to support this functionality.

Sixth, integrating VEVOS' simulation library into non-JVM-based languages is challenging. A simple command line interface could improve the accessibility of the library in other languages.

Seventh, the extraction of a feature model and build model with KConfigReader and KBuildMiner is not robust; it only works for certain projects and commits. While a naïve implementation might serve as temporary workaround (cf. Sec. 4.2), a more robust implementation should be provided for common build systems.

Lastly, the ground truth data should be accompanied by a more detailed explanation, and the community should strive towards a unified data format for benchmarks, to aid researchers in integrating benchmarks. For such a format, existing standardization efforts (e.g., the variability exchange language (VEL) [40, 47]) could be extended.

5 CONCLUSION

We examined to which extent the framework VEVOS can aid researchers in generating benchmarks for the empirical evaluation of clone-and-own techniques. For this purpose, we classified VEVOS' capabilities according to the eleven evolution scenarios for benchmarks presented by Strüber et al. [44]. We reported our experiences from six different studies using VEVOS in these evolution scenarios.

We found that VEVOS fully addresses the requirements of six out of eleven scenarios, and partially addresses the requirements of a further scenario. This is a considerable improvement over existing benchmarks, which address most scenarios only partially if at all. For five scenarios, VEVOS is the only benchmark to fully support them. With respect to its original intent, we conclude that VEVOS fully supports all scenarios which are immediately relevant for the evaluation of clone-and-own research.

ACKNOWLEDGMENTS

This work has been supported by the German Research Foundation within the project *VariantSync* (TH 2387/1-1 and KE 2267/1-1).

REFERENCES

[1] Iago Abal, Jean Melo, Stefan Stănciulescu, Claus Brabrand, Márcio Ribeiro, and Andrzej Wąsowski. 2018. Variability Bugs in Highly Configurable Systems: A Qualitative Analysis. *Trans. on Software Engineering and Methodology (TOSEM)* 26, 3, Article 10 (2018), 10:1–10:34 pages.

[2] Michał Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Stefan Stănciulescu, Andrzej Wąsowski, and Ina Schaefer. 2014. Flexible Product Line Engineering with a Virtual Platform. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. ACM, 532–535.

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines*. Springer.

[4] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. 2013. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Trans. on Software Engineering (TSE)* 39, 12 (2013), 1611–1640.

[5] Paul Maximilian Bittner, Alexander Schultheiß, Thomas Thüm, Timo Kehrer, Jeffrey M. Young, and Lukas Linsbauer. 2021. Feature Trace Recording. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 1007–1020.

[6] Paul Maximilian Bittner, Christof Tinnes, Alexander Schultheiß, Sören Viegener, Timo Kehrer, and Thomas Thüm. 2022. Classifying Edits to Variability in Source Code. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM.

[7] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An Exploratory Study of Cloning in Industrial Software Product Lines. In *Proc. Europ. Conf. on Software Maintenance and Reengineering (CSMR)*. IEEE, 25–34.

[8] Wolfram Fenske, Jens Meinicke, Sandro Schulze, Steffen Schulze, and Gunter Saake. 2017. Variant-Preserving Refactorings for Migrating Cloned Products to a Product Line. In *Proc. Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 316–326.

[9] Wolfram Fenske, Thomas Thüm, and Gunter Saake. 2014. A Taxonomy of Software Product Line Reengineering. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 4:1–4:8.

[10] Wenbin Ji, Thorsten Berger, Michal Antkiewicz, and Krzysztof Czarnecki. 2015. Maintaining Feature Traceability with Embedded Annotations. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 61–70.

[11] Ed Keenan, Adam Czauderna, Greg Leach, Jane Cleland-Huang, Yonghee Shin, Evan Moritz, Malcom Gethers, Denys Poshyvanyk, Jonathan Maletic, Jane Huffman Hayes, et al. 2012. Tracelab: An Experimental Workbench for Equipping Researchers to Innovate, Synthesize, and Comparatively Evaluate Traceability Solutions. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, 1375–1378.

[12] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. 2014. Propagation of Software Model Changes in the Context of Industrial Plant Automation. *Automatisierungstechnik* 62, 11 (2014), 803–814.

[13] Timo Kehrer, Thomas Thüm, Alexander Schultheiß, and Paul Maximilian Bittner. 2021. Bridging the Gap Between Clone-and-Own and Software Product Lines. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, 21–25.

[14] Rainer Koschke, Pierre Frenzel, Andreas P. Breu, and Karsten Angstmann. 2009. Extending the Reflexion Method for Consolidating Software Variants into Product Lines. *Software Quality Journal (SQJ)* 17, 4 (2009), 331–366.

[15] Charles W. Krueger. 2002. Easing the Transition to Software Mass Customization. In *Proc. Int'l Workshop on Software Product-Family Engineering (PFE)*. Springer, 282–293.

[16] Jacob Krüger and Thorsten Berger. 2020. An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 432–444.

[17] Jacob Krüger, Mukelabai Mukelabai, Wanzi Gu, Hui Shen, Regina Hebig, and Thorsten Berger. 2019. Where is my Feature and What is it About? A Case Study on Recovering Feature Facets. *J. Systems and Software (JSS)* 152 (2019), 239–253.

[18] Miguel A. Laguna and Yania Crespo. 2013. A Systematic Mapping Study on Software Product Line Evolution: From Legacy System Reengineering to Product Line Refactoring. *Science of Computer Programming (SCP)* 78, 8 (2013), 1010–1034.

[19] Raúl Lapeña, Manuel Ballarin, and Carlos Cetina. 2016. Towards Clone-and-Own Support: Locating Relevant Methods in Legacy Products. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 194–203.

[20] Lukas Linsbauer, Thorsten Berger, and Paul Grünbacher. 2017. A Classification of Variation Control Systems. In *Proc. Int'l Conf. on Generative Programming: Concepts & Experiences (GPCE)*. ACM, 49–62.

[21] Lukas Linsbauer, Stefan Fischer, Roberto E. Lopez-Herrejon, and Alexander Egyed. 2015. Using Traceability for Incremental Construction and Evolution of Software Product Portfolios. In *Proc. Int'l Symposium on Software and Systems Traceability (SST)*. IEEE, 57–60.

[22] Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2017. Variability Extraction and Modeling for Product Variants. *Software and System Modeling (SoSyM)* 16, 4 (2017), 1179–1199.

[23] Lukas Linsbauer, Felix Schwägerl, Thorsten Berger, and Paul Grünbacher. 2021. Concepts of Variation Control Systems. *J. Systems and Software (JSS)* 171 (2021).

[24] Wardah Mahmood, Daniel Strueber, Thorsten Berger, Ralf Laemmel, and Mukelabai Mukelabai. 2021. Seamless Variability Management With the Virtual Platform. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, 1658–1670.

[25] Jabier Martinez, Wesley K. G. Assunção, and Tewfik Ziadi. 2017. ESPLA: A Catalog of Extractive SPL Adoption Case Studies. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 38–41.

[26] Jabier Martinez, Nicolas Ordoñez, Xhevahire Tërnava, Tewfik Ziadi, Jairo Aponte, Eduardo Figueiredo, and Marco Tulio Valente. 2018. Feature Location Benchmark with ArgoUML SPL. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 257–263.

[27] Jabier Martinez, Tewfik Ziadi, Mike Papadakis, Tegawendé F. Bissyandé, Jacques Klein, and Yves le Traon. 2018. Feature Location Benchmark for Extractive Software Product Line Adoption Research Using Realistic and Synthetic Eclipse Variants. *J. Information and Software Technology (IST)* 104 (2018), 46–59.

[28] Raul Medeiros, Jabier Martinez, Oscar Díaz, and Jean-Rémy Falleri. 2023. Visualizations for the evolution of Variant-Rich Systems: A systematic mapping study. *J. Information and Software Technology (IST)* 154 (2023), 107084.

[29] Gabriela Karoline Michelon. 2020. Evolving System Families in Space and Time. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 104—-111.

[30] Gabriela Karoline Michelon, David Obermann, Wesley K. G. Assunção, Lukas Linsbauer, Paul Grünbacher, and Alexander Egyed. 2021. Managing Systems Evolving in Space and Time: Four Challenges for Maintenance, Evolution and Composition of Variants. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 75–80.

[31] Tristan Pfofe, Thomas Thüm, Sandro Schulze, Wolfram Fenske, and Ina Schaefer. 2016. Synchronizing Software Variants with VariantSync. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 329–332.

[32] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer.

[33] Poedjadevie Kadjel Ramkisoen, John Businge, Brent Van Bradel, Alexandre Decan, Serge Demeyer, Coen De Roover, and Foutse Khomh. 2022. PaReco: Patched Clones and Missed Patches among the Divergent Variants of a Software Family. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, ACM.

[34] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. 2013. Managing Cloned Variants: A Framework and Experience. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 101–110.

[35] AnaB. Sánchez, Sergio Segura, JoséA. Parejo, and Antonio Ruiz-Cortés. 2015. Variability Testing in the Wild: The Drupal Case Study. *Software and System Modeling (SoSyM)* (2015), 1–22.

[36] Thomas Schmorleiz and Ralf Lämmel. 2014. Similarity Management via History Annotation. In *Proc. Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE)*. Dipartimento di Informatica Università degli Studi dell'Aquila, L'Aquila, Italy, 45–48.

[37] Alexander Schultheiß, Paul Maximilian Bittner, Sascha El-Sharkawy, Thomas Thüm, and Timo Kehrer. 2022. Simulating the Evolution of Clone-and-Own Projects with VEVOS. In *Proc. Int'l Conf. on Evaluation Assessment in Software Engineering (EASE)*. ACM, 231–236.

[38] Alexander Schultheiß, Paul Maximilian Bittner, Timo Kehrer, and Thomas Thüm. 2020. On the Use of Product-Line Variants as Experimental Subjects for Clone-and-Own Research: A Case Study. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, Article 27, 6 pages.

[39] Alexander Schultheiß, Paul Maximilian Bittner, Thomas Thüm, and Timo Kehrer. 2022. Quantifying the Potential to Automate the Synchronization of Variants in Clone-and-Own. In *Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME)*. IEEE. To appear.

[40] Michael Schulze and Robert Hellebrand. 2015. Variability Exchange Language-A Generic Exchange Format for Variability Data. In *Software Engineering (Workshops)*. 71–80.

[41] Felix Schwägerl and Bernhard Westfechtel. 2016. SuperMod: Tool Support for Collaborative Filtered Model-Driven Software Product Line Engineering. In *Proc. Int'l Conf. on Automated Software Engineering (ASE)*. ACM, 822–827.

[42] Felix Schwägerl and Bernhard Westfechtel. 2019. Integrated Revision and Variation Control for Evolving Model-Driven Software Product Lines. *Software and System Modeling (SoSyM)* 18, 6 (2019), 3373–3420.

[43] Christoph Seidl, Ina Schaefer, and Uwe Aßmann. 2014. Capturing Variability in Space and Time with Hyper Feature Models. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, Article 6, 6:1–6:8 pages.

[44] Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, and Thorsten Berger. 2019. Facing the Truth: Benchmarking the Techniques for the Evolution of Variant-Rich Systems. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 177–188.

[45] Stefan Stănciulescu, Thorsten Berger, Eric Walkingshaw, and Andrzej Wąsowski. 2016. Concepts, Operations, and Feasibility of a Projection-Based Variation Control System. In *Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 323–333.

[46] Stefan Stănciulescu, Sandro Schulze, and Andrzej Wąsowski. 2015. Forked and Integrated Variants in an Open-Source Firmware Project. In *Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME)*. IEEE, 151–160.

[47] Chico Sundermann, Kevin Feichtinger, Dominik Engelhardt, Rick Rabiser, and Thomas Thüm. 2021. Yet Another Textual Variability Language? A Community Effort Towards a Unified Language. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 136–147.

[48] Eric Walkingshaw and Klaus Ostermann. 2014. Projectional Editing of Variational Software. In *Proc. Int'l Conf. on Generative Programming: Concepts & Experiences (GPCE)*. ACM, 29–38.

[49] Daniele Wolfart, Wesley Klewerton Guez Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In *XX Brazilian Symposium on Software Quality (SBQS '21)*. ACM, Article 17, 10 pages.

[50] Daniele Wolfart, Wesley Assunção, and Jabier Martinez. 2019. Open Source Software on the Research of Extractive Adoption of Software Product Lines. In *Anais do XVI Congresso Latino-Americano de Software Livre e Tecnologias Abertas*. SBC, 142–145.

[51] Zhenchang Xing, Yinxing Xue, and Stan Jarzabek. 2013. A Large Scale Linux-Kernel Based Benchmark for Feature Location Research. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, 1311–1314.

[52] Shurui Zhou, Ştefan Stănciulescu, Olaf Leßenich, Yingfei Xiong, Andrzej Wąsowski, and Christian Kästner. 2018. Identifying Features in Forks. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. ACM, 105–116.

[53] Chenguang Zhu, Yi Li, Julia Rubin, and Marsha Chechik. 2017. A Dataset for Dynamic Discovery of Semantic Changes in Version Controlled Software Histories. In *Proc. Working Conf. on Mining Software Repositories (MSR)*. IEEE, 523–526.