FM-PRO: A Feature Modeling Process

Johan Martinson, Wardah Mahmood, Jude Gyimah, and Thorsten Berger

Abstract—Almost any software system needs to exist in multiple variants. While branching or forking—a.k.a. clone & own—are simple and inexpensive strategies, they do not scale well with the number of variants created. Software platforms—a.k.a. software product lines—scale and allow to derive variants by selecting the desired features in an automated, tool-supported process. However, product lines are difficult to adopt and to evolve, requiring mechanisms to manage features and their implementations in complex codebases. Such systems can easily have thousands of features with intricate dependencies. Feature models have arguably become the most popular notation to model and manage features, mainly due to their intuitive, tree-like representation. Introduced more than 30 years ago, thousands of techniques relying on feature models have been presented, including model configuration, synthesis, analysis, and evolution techniques. However, despite many success stories, organizations still struggle with adopting software product lines, limiting the usefulness of such techniques. Surprisingly, no modeling process exists to systematically create feature models, despite them being the main artifact of a product line. This challenges organizations, even hindering the adoption of product lines altogether.
We present FM-PRO, a process to engineer feature models. It can be used with different adoption strategies for product lines, including creating one from scratch (*pro-active adoption*) and re-engineering one from existing cloned variants (*extractive adoption*). The resulting feature models can be used for configuration, planning, evolution, reasoning about variants, or keeping an overview understanding of complex software platforms. We systematically engineered the process based on empirically elicited modeling principles. We evaluated and refined it in a real-world industrial case study, two surveys with industrial and academic feature-modeling experts, as well as an open-source case study. We hope that FM-PRO helps to adopt feature models and that it facilitates higher-level, feature-oriented engineering practices, establishing features as a better and more abstract way to manage increasingly complex codebases.

Index Terms—feature modeling, software engineering processes, features, product lines, configurable systems.

1 Introduction

Software systems often need to exist in multiple variants—accounting for varying customer requirements, hardware, or operating environments. Variants also allow experimenting with new ideas and optimizing non-functional properties, such as cost, performance, or power consumption. Organizations often use branching or forking—a.k.a. clone & own [1]–[4] as a simple and cheap strategy. However, it does not scale with the number of variants, quickly causing high maintenance efforts [5], [6]. Developers lose the overview over the variants [7] and are mainly occupied with integrating changes across variants, instead of developing new features [8]. Engineering software platforms—a.k.a. software product lines [9]–[13]—scales, allowing to automatically derive individual variants by selecting the desired features in a configurator tool. However, platforms are difficult to adopt and evolve, relying on mechanisms to manage features, their constraints (e.g., dependencies), and their implementation in code, often using conditional compilation (e.g., `#ifdef`), feature toggles [14], [15], or configurable build systems [16]. Large systems can easily have thousands of features with intricate constraints [17], [18], requiring proper engineering practices and model-based representations.

- *J. Martinson is with the Faculty of Computer Science at Ruhr University Bochum, Germany.*
  E-mail johan.martinson@rub.de
- *W. Mahmood is with the Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Sweden.*
- *J. Gyimah is with the Faculty of Computer Science at Ruhr University Bochum, Germany*
- *T. Berger is with the Faculty of Computer Science at Ruhr University Bochum, Germany and Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Sweden.*

Feature models can be seen as the most prominent modeling language for features and their constraints. Introduced over three decades ago [19], they quickly became popular in academia and industry due to their intuitive, tree-like notation. Figure 1 shows a small feature model. It is an excerpt (18 of 108 features) from the configurable open-source SSL server axTLS. Features are organized in a hierarchy and represent abstract units of functionality that are meaningful to stakeholders [20]. Features can be mandatory (contained in all variants), optional (contained in some variants), part of a feature group (OR/XOR group), or part of cross-tree constraints (shown below the tree).

Feature models provide various benefits. As input to interactive configurator tools, they allow configuring the desired variant by selecting features. In addition, feature models allow keeping an overview understanding of the codebase, support system design [21], help evolving and maintaining complex platforms, support communication among

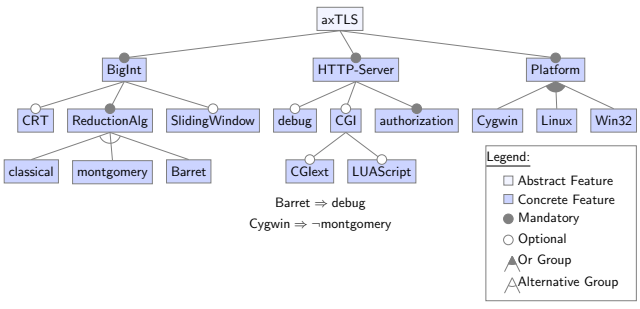


Fig. 1: Feature model excerpt of axTLS (SSL server)

stakeholders [22], and drive quality assurance [23], [24].

Over the years, many companies have adopted product-line platforms and feature models, as witnessed by many case studies, experience reports, or surveys collected in the Product Line Hall of Fame (splc.net/fame.html), in books [11], [25], online catalogs [26], and meta-studies [8], [27]. Feature modeling is supported by commercial and open-source tools [28], such as pure::variants [29], Gears [30], FeatureIDE [31], or UML tools with profiles for feature modeling [32], [33]. It is also considered in standards, such as AUTOSAR, ISO26580:2021, the OASIS Variability Exchange Language, and OMG's upcoming SysML V2.

Since their introduction, feature models have gained increasing popularity and have given rise to many notations [34], [35] and tools [29], [30], [36], [37]. Figure 1 shows the traditional notation, but many variations and extensions exist, including textual languages [38], such as Clafer [34], Kconfig [39], CDL [18], or UVL [40], [41]. Over the decades, the research community has contributed many variants and extension of feature modeling [42], as well as it has built thousands of techniques for model configuration [39], [43]–[45], synthesis [46], [47], analysis [24], [48]–[50], and evolution [51], [52]. The community has also started various initiatives, such as the INCOSE Product Line Working group, the MODEVAR community initiative for a common feature-modeling language [53], and a (now abandoned) OMG initiative on the common variability language CVL [54]–[56].

Unfortunately, despite these success stories, initiatives, and supporting tools, organizations still struggle with adopting platforms and feature models in the first place, limiting the usefulness of all these contributions. Since developers are not accustomed to thinking in terms of features in their routine development activities, they already find it challenging to identify what qualifies as a feature and if it should be modeled as such. Many processes for product-line engineering exist [10], [12], [57]–[60], most of which including "domain requirement engineering" as a core activity in the scoping of the platform. However, they lack concrete guidelines on feature model construction. Consequently, feature models are mainly created manually, since they contain highly domain-specific knowledge (especially the features and their organization in a hierarchy) and gather information (especially constraints) that is typically scattered across the codebase in different artifacts, such as code, requirements, feature databases, or configuration files. As such, it is surprising that no modeling process to systematically create and manage feature models exists. To the best of our knowledge, only modeling principles [61], as a result of our previous work, and modeling guidelines [62] have been presented.

To address this gap, we present FM-PRO, a feature modeling process to systematically create feature models. It covers different adoption strategies for platforms, including re-engineering a platform from existing cloned variants. It covers the three adoption strategies for software platforms and helps the different stakeholders (e.g., modeler, domain expert, developer, and method expert) engineer feature models, including planning, training, and quality-assurance activities. The process is organized into activities, sub-activities, questions, and phases, and as such, provides systematic guidelines to the stakeholders involved. We systematically engineered the process by analyzing existing, empirically elicited modeling principles [61] and triangulating this analysis with our own practical and research experiences. We evaluated and refined FM-PRO with an open-sourced product line case, together with a real-world industrial platform-adoption effort, taking qualitative and quantitative feedback from industrial and academic feature-modeling experts into account. To the best of our knowledge, this is the first process for feature model construction. Our process can be seen as an extension of the existing processes for platform adoption [10], [12], [57]–[60].

We contribute:

- the **feature-modeling process FM-PRO**, described in its own technical documentation [63];
- **evidence of applicability**, stemming from an open-source case study evaluation; **empirical data** from an industrial case study, an open-source case study and an assessment by 27 research and industrial experts on feature modeling; and
- an **online appendix** containing the evaluation data [64].

On a final note, we hope that our work helps adopt feature models and facilitates feature-oriented engineering practices, establishing features as a more abstract way to interface with, and manage, increasingly complex codebases. For instance, cyber-physical or information systems have massive variability, benefiting from systematic engineering practices as FM-PRO provides. In addition, modern software will be increasingly generated based on AI techniques, requiring raising the level of abstraction from code to the domain—that is, the feature level, to implement and manage software artifacts.

2 Background and Related Work

We briefly introduce the necessary background on variant-rich systems and software platforms, their adoption and engineering practices, as well as features, feature models, and the feature modeling principles FM-PRO is built upon.

2.1 Variant-Rich Systems and Platforms

Platform Adoption. FM-PRO supports the different adoption strategies for product-line platforms [57], [65]. *Proactive* adoption involves planning and creating a platform from scratch. *Reactive* adoption involves starting with one variant and incrementally migrating it to a platform by making features optional and adding new ones (a.k.a. featurization [66]). *Extractive* adoption, as the most frequent strategy in practice [6], [26], [67], [68], involves extracting the features from variants that have previously been realized using clone & own. All strategies require creating a feature model, but using different sources for identifying features, their relationships in the hierarchy, and constraints. For instance, in proactive adoption, since no implemented variants exist, organizations will primarily need to rely on the knowledge of domain experts. In extractive adoption, since many features would already have been implemented, organizations need to identify features from existing artifacts [5], [69], [70].

Once adopted, the platform integrates all features of the possible variants, typically using a dedicated architecture, variability mechanisms to realize variation points, the

feature model, an interactive configurator tool, and a configurable build system. Notably, our modeling process does not prescribe a strategy for platform adoption, which has already been done by many existing processes and frameworks against different adoption strategies (as cited above). Instead, it refines the processes for platform adoption by contributing concrete activities facilitating creation of a feature model from scratch (i.e., in proactive adoption) as well as extracting it from one or multiple variants (i.e., in reactive and extractive adoption respectively). Finally, it is also suited for an incremental adoption of product lines [68], [71], [72].

2.2 Features and Feature Models

Feature. Features abstractly represent the commonality and variability of variants in a product-line platform. While many definitions exist [10], [11], [19], [59], [73]–[75], we refer to features as entities relevant and understood by many stakeholders. They represent a domain concept and are units of reuse and communication [20], [76]. Implemented features often cross-cut unless they are modularized [77], which illustrates their flexibility. Modern agile development processes, such as SCRUM, XP or FDD [78]–[80] use features to plan, design, and evolve systems. Creating feature teams [81] allows developers to gather more specific knowledge to become more agile.

Feature Model. Feature models are hierarchical structures of features and their constraints. Introduced as part of the domain analysis technique FODA [19], [82], they have since become popular [8], [67], [83], as also witnessed by the first paper on FODA [19] surpassing 5,800 citations (note: Google Scholar's citation count recently broke for it). Recall from Fig. 1 that features can be either *mandatory* (contained in all variants) or *optional* (contained in some variants), that they can be part of a *feature group*, typically *OR* (at least one feature should be selected), *XOR* (only one feature should be selected), and *MUTEX* (none or one feature can be selected). Additional, so-called cross-tree constraints, which cannot be expressed using these means, are added as expressions (typically using Boolean logic) below the diagram. Beyond these basic concepts, in some languages, features can also have different types beyond Boolean, then requiring more expressive constraints [84], and some feature modeling languages support advanced concepts, such as feature attributes and non-Boolean features. FM-PRO does not support those advanced feature modeling concepts. However, since it does not prevent them, FM-PRO can be used with feature modeling languages of different expressiveness, as long as they support the basic concepts (cf. Sec. 4).

Tools and Textual Feature Models. Recall that a variety of feature modeling tools and different variations of the feature-modeling notation exist [42]. Since the traditional graphical notation shown in Fig. 1 hardly scales, configurator tools typically use different notations, usually showing the feature hierarchy as a tree menu. Since textual feature-modeling languages are even easier to edit and manage (e.g., through version-control systems), we conducted one of our evaluations (the industrial case study) with a textual language.

Textual modeling languages offer several advantages for feature modeling, particularly in industrial settings. They

```
1  AXTLS
2      or Platform
3          Linux
4          Win32
5          CygWin
6      HTTPServer
7          debug ?
8          authorization
9          or CGI ?
10             CGIext ?
11             LUAscript ?
12     BigInt
13         SlidingWindow ?
14         CRT ?
15         xor ReductionAlg
16             Montgomery
17                 [ !Cygwin ]
18             classical
19             Barret
20                 [ debug ]
```

Fig. 2: The axTLS feature model from Fig. 1 in Clafer

are easier to edit and manage, especially when integrated with version-control systems [38], [85]. Among textual languages, Clafer [86] and UVL [40], [41] are prominent examples. UVL is a more recent language, whereas Clafer has a more concise and simple syntax (which is also harder to violate). Both UVL and Clafer support basic and advanced feature modeling concepts and come with tool support. UVL[1] comes with an LSP, a web interface, translators, as well as IDE (Visual Studio and Eclipse) and analysis framework integrations. Clafer[2] offers a compiler and an analysis framework (e.g., instance generator, optimizer), web interfaces, and IDE integration for JetBrains IDEs (e.g., IntelliJ IDEA, PyCharm, Rider) [87]. Our experience shows that the minimal syntax of Clafer is easier to adopt for developers in industry, especially when using it does not require adopting additional tooling. Since UVL's syntax is less succinct (e.g., more keywords), enforces a more stringent structure, and disconnects constraints from feature declarations, which made it harder to understand, we chose Clafer for our industrial case study, while all other textual feature modeling languages are also applicable with FM-PRO.

Figure 2 shows our axTLS model from Fig. 1 expressed in Clafer's minimal textual syntax. Each line defines one feature, with indentation representing the hierarchy. Optional features are suffixed with a ?. Feature groups are denoted by the keyword *or*, *xor*, or *mutex*. Cross-tree constraints are declared in brackets and are implied by their nesting scope (e.g., Barret requires debug). See the Clafer documentation for further details [86]. Still, FM-PRO is notation-agnostic, which we show by using the tool FeatureIDE in the open-source case study, which offers the graphical notation shown in Fig. 1.

2.3 Feature Modeling Principles

Modeling notations typically come with a modeling process, describing when, where, and how to apply the notation. Processes provide guidelines and an order of activities, as well as they describe the role performing such. For feature models, to the best of our knowledge, only our modeling principles [61] and other modeling guidelines [62] have been presented, but no process.

1. https://universal-variability-language.github.io
2. https://www.clafer.org

The principles aim to address the challenges in building and evolving feature models, offering a structured approach to manage the complexity of software product lines effectively. They are divided into eight groups; **PP:** Planning and Preparation, **T:** Training, **D:** Dependencies, **IS:** Information Sources, **MO:** Model Organization, **M:** Modeling, **MME:** Model Maintenance and Evolution, and **QA:** Quality Assurance. The following will give an overview on what principles they, the individual principle-definitions can be found in the appendix.

PP: Planning and Preparation. The six principles (**PP1–PP6**) emphasize the importance of identifying relevant stakeholders and defining the purpose of the feature model. They also highlight the need for unifying domain terminology and establishing clear communication among stakeholders.

T: Training. It is crucial to ensure that modelers are well-equipped to handle feature modeling tasks. The three principles suggest conducting pilot projects and providing iterative learning opportunities. Training sessions help modelers understand the principles and apply them effectively in practice.

IS: Information Sources. The three principles recommend leveraging domain knowledge from existing artifacts, conducting workshops with domain experts, and applying both bottom-up and top-down modeling approaches. Utilizing diverse information sources ensure that the feature model is well-informed and accurate.

MO: Modeling Organization. Organizing features hierarchically and grouping related features into clusters are essential for managing complexity. The principles also emphasizes modularity and scalability, ensuring that the feature model can accommodate future changes.

M: Modeling. This group of 11 principles (**M1–M11**) focuses on the core activities of feature modeling, such as identifying and defining features, ensuring distinctiveness, and mapping features to software assets. It also stresses the importance of clear dependency declarations, regular validation and verification, and maintaining traceability between features and requirements.

D: Dependencies. Clearly defining dependencies between features is crucial to avoid conflicts and ensure consistency. These two principles (**D1–D2**) highlights the need for precise dependency declarations and regular checks to maintain the integrity of the feature model.

QA: Quality Assurance. Quality assurance involves regular validation and verification of the feature model to ensure it meets the intended requirements. These three principles (**QA1–QA3**) recommends using automated tools for consistency checks and maintaining up-to-date documentation to support ongoing quality assurance efforts.

MME: Model Maintenance and Evolution. The final three principles (**MME1–MME3**) addresses the need for regular updates and maintenance of the feature model. It emphasizes planning for evolution to accommodate changing requirements and keeping documentation current. Ensuring the model evolves with the software product line is essential for long-term success.

3 Methodology

We designed, evaluated, and refined the process using the following methodology. Since evaluating a modeling process is generally challenging, our methodology was inspired by design science [88], with an initial evaluation and refinement cycle, followed by an expert evaluation and finally an open-source case study at the end. First, we synthesized our process from our feature modeling principles [61], that we elicited from 10 interviews with industrial practitioners from nine companies, as well as the analysis of the relevant literature (105 papers). Next, we combined a hands-on and very practical evaluation with practitioners by applying our process to an industrial case study in real-time. Furthermore, based on the feedback from our industrial evaluation, we refined the process, and evaluated it with the aid of academic and industrial feature modeling experts. We also conducted an open-source case study to evaluate FM-PRO in a controlled setting. Further details, raw data, and additional explanations are available in our online appendix [64].

3.1 Data Sources

Our methodology relied on the following data sources.

Feature Modeling Principles. In our previous work [61], we presented 34 principles of feature modeling. These are modeling heuristics that we extracted and synthesized from the literature and from interviews with modeling experts stemming from nine different companies. An example of a principle is *"**M1:** Focus first on features that distinguish variants,"* which suggests that when identifying features, companies should focus on variability instead of commonality between variants to make the activity easier to perform. The principles covered the following categories: planning and preparation (PP), training (T), information sources (IS), model organization (MO), modeling features (M), modeling dependencies (D), quality assurance (QA), and model maintenance and evolution (MME). We synthesized FM-PRO to align with and adhere to these principles.

Authors' Experiences. Our own experiences with product-line engineering and feature modeling range between 5 and 16 years. We have collaborated with many industries building variant-rich software systems, in funded research projects (also large framework projects with up to 30 partners), in action research (company visits with close interaction), and in empirical studies (e.g., via interviews or artifact/longitudinal studies). We are also embedded in the research community, having contributed methods and tools, as well as empirical insights over the years.

Industrial Partner. To evaluate FM-PRO's applicability and effectiveness in real setting, we applied it at our industrial partner, a medium-sized company with around 700 employees. The organization comprises 200 developers, over 300 application specialists (combining the role of requirements engineer and tester), and other roles that focus on software architecture, business analysis, management, or product delivery. The company builds large information systems for logistics companies. Its goal is to establish feature modeling as a starting point to making their variants configurable, deriving variants based on customer requirements by enabling and disabling features, i.e., performing re-active adoption. Our data source was one of their long serving (10+ years) products (comprising around 1250 files, 210k lines of code and 48 features) and a team of 6 employees (4 developers, 1 application specialist,

1 interaction designer) collaborating with 5 researchers to systematically apply the process and collect qualitative and quantitative feedback. One author was employed at the company and had expert knowledge of the subject product.

ESPLA Catalog and Experimental Data. For our open source case study, our data sources primarily came from the Extractive Software Product Line Adoption (ESPLA) Catalog [26] which is a collaborative catalog of case studies on extractive product line adoption and reuse. The Github repository linked to our product line of choice, i.e., *Video.js*, contained an aggregation of case study data gathered during the evaluation of a strategy (RiPLE-HC) [89], which was designed to handle variability on both the feature modeling and code level of JavaScript-based systems. Through RiPLE-HC, feature-based code organization and preprocessing annotations can be used to handle fine-grained variability. Thus, in their case study evaluations, they created a product line from scratch and transformed a selection of them from the qualitas.js corpus JavaScript systems dataset. Based on that, we randomly selected a product line from the dataset provided, for use in our open-source case study.

Research Experts in Feature Modeling. We used qualitative and quantitative feedback of 20 experts known in the academic and industrial research community. We recruited the experts as follows. We contacted experts from our knowledge of the literature, of feature-modeling standardization or community initiatives (e.g., CVL [54], as well as respective books or catalogs of case studies, cf. Sec. 1), as well as tool vendors (e.g., pure::systems). We also sought through the proceedings of the last 14 instances of the Systems and Software Product Line Conference (SPLC), the flagship conference in software product-line engineering, and 12 instances of VaMoS, another well-recognized workshop and (since 2020) working conference on variability modeling. We searched for experience reports and industrial track papers where authors reported their experiences with applying feature modeling in substantial real-world projects. While all experts have published on feature modeling, 11 of them are industrial practitioners, whereas six of them are academic researchers. Three of the 20 experts chose to stay anonymous.

Industrial Experts in Feature Modeling. Similarly to the research experts we used qualitative and quantitative feedback from 7 industrial experts with various roles from a large company with an average of 3–5 years experience in feature modeling. The company builds infotainment systems for the automotive industry and has a decade long history of both using feature models and modeling their systems.

3.2 Process Design

We structured the process into distinct *phases*, each phase focusing on a different component of feature modeling. The phases contain *activities*, *composite activities* with *sub-activities*, and *questions*. The different types of activities can be optional depending on the context or circumstances.

We started with carefully analyzing our feature-modeling principles to identify concrete modeling activities the modelers and other involved stakeholders would need to perform. Subsequently, we defined those activities, also suggesting who should perform them based on our own experience in feature modeling. This resulted in multiple fine-grained activities. Specifically, principles that suggested an action were directly translated into activities. For example, the principle **PP1** (Identify relevant stakeholders) directly translated into the activity *"Identify stakeholder"*, while the principle **PP6** (Keep the number of modelers low) also inspired the design of this activity. For tasks that only fit certain contexts, we designed optional activities. If an activity required a decision to be made, we added a relevant question that would help the modelers in making that decision. For activities requiring multiple sub-tasks, we created composite activities comprising multiple sub-activities. We provide examples to the above-mentioned elements in Sec. 4.

Next, based on the similarity of activities and the context they were suited in, we structured the activities into distinct phases. To make the process simplistic and more applicable, we aimed for a small number of phases. When assigning an activity to a phase, we determined whether the activity intuitively fit the general idea of the phase. We decided on the position of the activity in the phase based on the dependencies between different activities (e.g., activities might require some result or insights that previous activities provide). We made some activities mandatory and others optional based on their relevance in different contexts as well as their required effort. For each activity, we added a short description, details on how to perform it, the involved stakeholders, and the expected outcomes.

3.3 Industrial Case Study

Following the formulation of the first version of FM-PRO, we evaluated it by applying it at our industrial partner. To commence the collaboration, all researchers visited the company. One author presented the general idea of software product-lines, configurable software systems, and the relevance of feature models in that context. We also met with the other participants of our evaluation (identified by the author working in the company), which were the industry stakeholders including four developers, one application specialists, and one interaction designer. Lastly, we received an introduction of the product that would act as the software system we would analyze and construct the feature model for.

After the introductory meeting, we provided the participants with a description of the process to make them familiar with it. Additionally, as a preparatory task, we asked them to fill out a questionnaire. The questionnaire comprised questions pertaining to the clarity, applicability, and effectiveness of the process. The aim of the questionnaire was to determine how the process was perceived by the participants in different aspects (e.g., clarity of the formulation). The questionnaire comprised sections, each relevant to one activity. The rationale for planning the industrial evaluation at the granularity of activities is that industrial practitioners (developers, designers, modelers), when following the process, will go through the process activity by activity. Then, it is important that we get feedback about the clarity and applicability at the activity level so we identify potential area of improvement, making sure each activity, and consequently, the entire process is applicable in industrial context. In our questionnaire, each section comprised (the same) eight questions. The first four questions

were designed to be answered on a five-point Likert scale (fully disagree, disagree, neutral, agree, fully agree), and aligned with the aspects we wanted to evaluate. The last four questions aimed at getting subjective opinions from the participants, and were therefore designed to be open-ended.
I1. The formulation of the step is clear (*Clarity*)
I2. The step is applicable to our project (*Applicability*)
I3. The step comes at the right place in the process (*Order in process*)
I4. The step is useful (*Usefulness*)
I5. How much effort (in person-hours) would the activity require?
I6. What challenges do you expect when performing the activity?
I7. What suggestions for improvement do you have for this activity?
I8. Are there any other comments?

We executed the process using different formats. We used a workshop format when the activities involved collaboration among the stakeholders. We also conducted one-to-one, semi-structured interviews with participants to extract features from their expertise. We deliberately avoided a workshop format and opted for interviews to allow varying points of view on the process and eliminate the risk of participants' views to be influenced by those of each other. We interviewed the industrial participants (cf. Sec. 3.1).

Following the process, we manually inspected the product's codebase to identify features, followed by a validation step with an expert. To this end, we conducted a total of three meetings with one of the core product developers who had three years of experience working with the product under analysis. The first meeting involved product introduction, where the developer walked us through the product's user interface and provided an overview of its functionality. In the second meeting, the focus shifted towards obtaining understanding of the product's codebase. This meeting allowed us to explore the underlying structure and architecture of the product, enabling us to comprehend its functionalities more effectively. The last meeting involved feature model validation, where we presented the created feature model to the developer to gain feedback on the identified features, feature relationships, and cross-tree constraints. The meetings also helped us in identifying which features would be considered optional and which would be mandatory. We conducted the extraction independent of the developer, in-between the three meetings. For the actual modeling, we used Clafer [34] (explained in Sec. 2). As mentioned above (in Sec. 2), we use Clafer for its simplicity. The textual notation allows modelers to easily create, edit, and manage feature models.

Following the completion of the process execution, we asked the participants to edit their responses in the questionnaire if required to provide more updated feedback and insights.

3.4 Process Refinement

The execution of the process resulted in a diverse range of feedback. First, the participants' responses to the questionnaire provided insights into the clarity, intuitiveness, and applicability of the process. Second, the challenges we faced during the process execution helped us identify areas of improvement in the process. Lastly, the experts' responses to the survey (specifically the open-ended questions) also helped us identify potential areas of improvement. Most of the refinements were related to the clarity of an activity, the amount of emphasis we put on a certain activity, or the order of an activity in the process. In some instances, we realized that guidelines were lacking on how to perform a certain activity. In other instances, there were guidelines, but they were incomplete.

We incorporated the feedback in various ways. First, we added more clarity in the activity descriptions by elaborating them further and putting more emphasis on the aspects that were deemed vague. Second, in some instances, we re-arranged the activities in the phase to make the order of activities more intuitive and applicable. Lastly, we proposed new activities in response to the identification of missing guidelines. None of the activities was removed, since each activity was deemed useful and applicable by our participants. Notably, we refined the process twice, once after the industrial evaluation, and once after the feedback from the expert survey.

3.5 Expert Assessments

We assessed FM-PRO through a survey answered by experts in the research community and by industrial experts. In the survey, we formulated the questions focusing on the same evaluation metrics as those in the industrial evaluation (i.e., clarity, applicability, and order in process), requiring five-point Likert scale responses. To gain an understanding on the perceptibility of the process from an experts' point of view, we also added a question about intuitiveness of the activities in each phase. Aiming for a broader feedback and considering the time constraints of the experts, in contrast to the industrial evaluation, we formulated the questions per phase (instead of per activity). As a result, the survey comprised four main sections (with two additional sections for introduction and optionally providing contact details). In addition to the questions pertaining to the evaluation metrics, we also asked experts for qualitative feedback, allowing more detailed and subjective responses. Specifically, we asked the following questions:

E1. The formulation of activities in the phase is clear (*Clarity*)

E2. Overall, this phase is applicable in industry (*Applicability*)

E3. The order of activities in the phase is logical (*Order*)

E4. The activities in the phase are intuitive (*Intuitiveness*)

E5. Is there something that needs improvement in this phase?

3.5.1 Research Expert Assessment

We shared the refined process description with 20 research experts, asking them for both quantitative and qualitative feedback. For conducting the survey, we contacted the experts through email, sending them a link to the process (in a PDF document) as well as the questionnaire (realized in the form of an online Google form).

3.5.2 Industrial Expert Assessment

We shared the refined FM-PRO from the research expert evaluation with 7 industrial experts, asking them for both quantitative and qualitative feedback. In contrast to the evaluation with the research experts, we instead conducted a workshop session where the participants got to read the process one phase at a time and then answer questions in a questionnaire about each phase. Similarly to the research expert assessment (Sec. 3.5.1) we also shared the process as a PDF and the questionnaire was realized in the form of an online Google form.

3.6 Open-Source Case Study

To evaluate the the final FM-PRO-revision, we conducted an experiment with a subject that had no prior experience with feature models or FM-PRO. With a reactive adoption approach, we selected a software platform case, that fit our needs (i.e. open-source, reusable assets, LOC, modules etc.), to apply and evaluate FM-PRO on. Our platform selection process was characterized by meticulously searching through the online ESPLA catalog [26] to find a system that sufficed in terms of size and number of features. The ESPLA catalog presents case studies and artefacts related to extractive product line adoption. Then we reviewed and verified existing features already identified within the product line case. This review process involved the inspection of each feature via interactions with a running instance of the system, as well as a step-by-step module inspection and analysis on the codebase level. We then proceeded to model all gathered and verified features. Our modeling efforts required a strong understanding of feature relationships regarding the perceived hierarchy. Furthermore, we continued to model all observable dependencies, relationships and constraints. Then, last but not least, we implemented general model refinements through critical feature reviews to verify feature positions in the model tree.

4 FM-PRO Overview

We present a high-level description of FM-PRO. The process itself is contained in its own technical documentation [63]. Our final process model comprises 30 activities in four phases: *Pre-Modeling* (11 activities, 4 questions), *Domain Analysis and Scoping* (4 activities, 1 question), *Modeling* (12 activities, 1 question), and *Maintenance and Evolution* (3 activities). Figure 3 shows a high-level overview of the feature modeling process. The more detailed Fig. 4, 5, 6 and 10 illustrate all the activities and questions involved in the four phases. In the overall process, the *Pre-Modeling* phase plays a pivotal role as it involves activities pertaining to the planning and setup required for the process, and precedes other phases in the process. An example of a sub-activity in this phase is *"Define model purpose,"* which involves establishing the purpose of the feature model explicitly, thereby setting the focus of modeling. As such, we positioned the activity at the beginning of the entire process, preceding all other activities pertaining to modeling. Instead of making the process sequential, we designed it such that the phases *Domain Analysis and Scoping* and *Modeling* could be performed in parallel. The *Domain Analysis and Scoping* phase involves identifying the features to be modeled, along with their relationships and constraints, whereas the *Modeling* phase involves modeling them into (one or more) feature models. Finally, the *Maintenance and Evolution* phase allows modelers to refine and extend the created feature model based on stakeholder feedback and in response to incoming changes (e.g., new feature requests). This phase is continuous and can be iteratively performed.

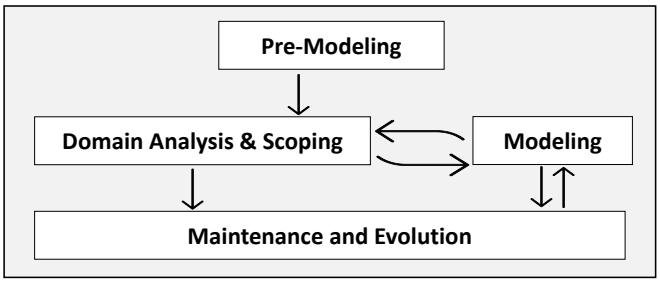


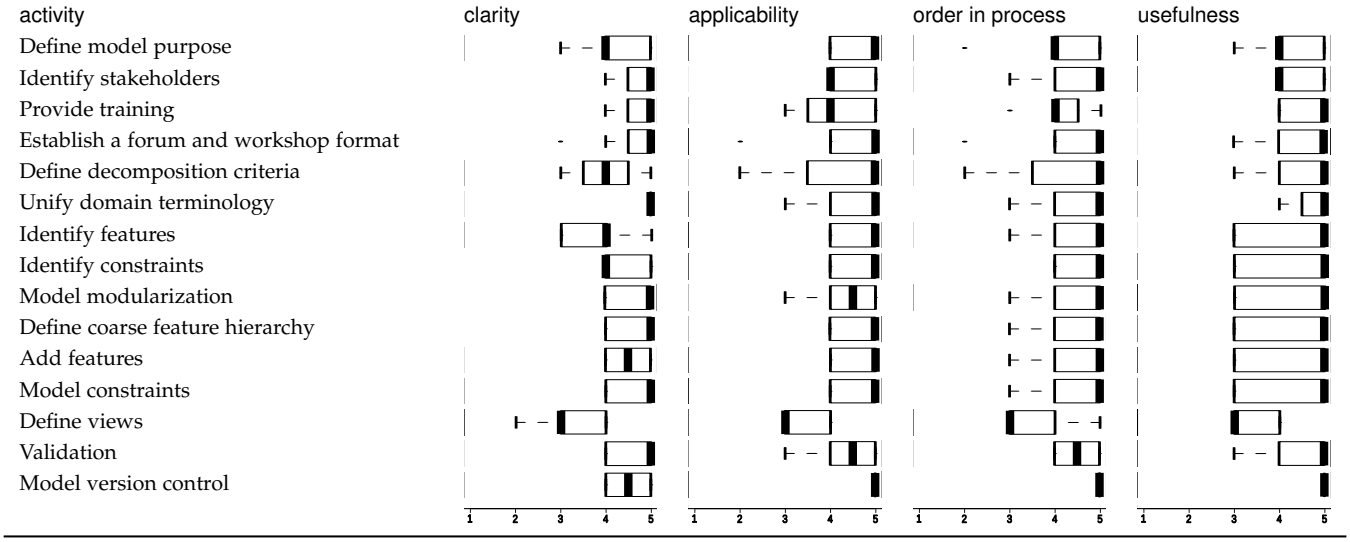
Fig. 3: High-level overview of FM-PRO

As mentioned above (Sec. 3.2), our process comprises optional activities; activities that are only suitable in some contexts. An example of such an activity is *"Identify constraints"* that requires modelers to specify the cross-tree constraints in the feature model. While an important activity, we deem it optional, since modeling constraints can be expensive, and not required if experts (with significant knowledge of the constraints) will use the model for, for example, configuring products. Additionally, as mentioned above, we supplemented the activities requiring decisions with questions that would help developers in making a choice. In the above-mentioned example, modelers can decide if they should identify the constraints or not by answering the question of who will be the end-users of the feature model. Lastly, as mentioned above, for activities involving multiple sub-tasks, we designed them to be composite activities, each comprising multiple sub-activities. An example of a composite activity is *"Identify features,"* a core activity in the *Modeling* phase, which is split into top-down and bottom-up feature identification as sub-activities (explained below).

We designed our process model to cater for all the adoption strategies (Sec. 1)—proposing top-down and bottom-up analyses (in *Domain Analysis and Scoping* phase, Section 6) for feature identification. We recommend that companies use a combination of both if some variants already exist. The **top-down** approach is well-suited for proactive adoption, where modelers are required to perform dedicated domain analysis and scoping. Here, the identified features determine the focus of the domain, and as such, provide an economic benefit and align with the business strategy of the company. The **bottom-up** approach is well-suited for reactive and extractive adoption, where the company already has a variant or set of (similar or even cloned) variants in place. There, modelers should start with pairwise diffing between two variants, taking one of them as the base variant. The identified differences can be translated into variation points, which can directly be embedded in the code assets using any variability mechanism (e.g., preprocessor directives) to make them configurable.

FM-PRO focuses on creating feature models and does not prescribe a specific language. FM-PRO supports Boolean

TABLE 1: Industrial evaluation: Overview of participants' ratings for the studied aspects



features, a feature hierarchy, and constraints in terms of mandatory and optional features, feature groups (OR and XOR), and cross-tree constraints. Furthermore, it advocates for model modularization via simple model inclusion mechanisms (where one can have different files for one model). As such, a feature modeling language should support those basic concepts. Note that, while FM-PRO does not support modeling advanced feature-modeling constructs, such as feature attributes, non-Boolean constraints, cloned features, and arbitrary group cardinalities, it does not prevent using them. Notably, these constructs are rare and did not appear in the initial interviews. For instance, non-Boolean constraints appear to some extent in systems software [84], [90].

On a final note, while maintenance and evolution is beyond the scope of FM-PRO, it contains such a phase to raise awareness for it. However, given the complexity of those activities, especially the co-evolution of the feature-model and the codebase, which can be challenging [91], creating a model evolution and maintenance process (FM-PRO-EVO) is valuable future work.

5 Pre-Modeling

We present the design of the *Pre-Modeling* phase, its industrial evaluation, and the final expert assessment. Thereby, we present the refinements we made in the phase and how the principles and our experiences influenced our design decisions. Notably, we structure our paper along the phases instead of the steps we took to design, evaluate, and refine each phase in order to put more emphasis on each phase, what it provides, and how it relates to the comprised activities and the other phases. Additionally, we strive to be more transparent and intuitive with the evolution of each phase after the multi-fold evaluation. Figure 4 shows all *Pre-Modeling* activities in their final order after all refinements.

5.1 Design of FM-PRO

We derived the first activity *"Define model purpose"* from principle **PP3** (Define the purpose of the feature model), which, as the name implies, requires the modelers to establish the motive for creating a feature model. We made this an opening activity, since having a clear model purpose sets the direction of the process execution, and makes it easier to conduct the subsequent activities. Next, we derived the second activity *"Identify stakeholders"* from principle **PP1** (Identify relevant stakeholders), which requires identifying experts with deep knowledge about the system(s) to be modeled. Taking guidance from principle **PP6** (Keep the number of modelers low), we suggest using a low number of modelers, sometimes as low as one person. Additionally, taking guidance from **D2** (If the main users of a feature model are end-users, perform feature-dependency modeling), we recommend modelers to only perform feature-dependency modeling if the feature model is to be used by the end-users in order to allow only correct configurations. We derived the third activity *"Provide training"* from principles **T1** (Familiarize with the basics of product-line engineering), **T2** (Select a small sub-system to be modeled for training), and **T3** (Conduct a pilot project). The activity comprises three sub-activities: *"Tool and notation training," "Product-line education,"* and *"Pilot project."* Next, we defined the fourth activity *"Create expectation and change management"* to allow reiterating through the motivation and establishing a clear communication plan for detailing the necessary changes in the structure and architecture of the platform as well as the individual variants. The activity was not inspired from any principle; however, we deem it as a natural successor to the previous activity, since a communication plan is crucial to bring all the stakeholder on the same page. Next, we derived the fifth activity *"Establish workshop and forum format"* from principles **IS1** (Rely on domain knowledge and existing artifacts to construct a feature model), **QA1** (Validate the obtained feature model in workshops with domain experts), and **M1** (Use workshops to extract domain knowledge). Next, we derived the sixth activity *"Define decomposition criteria"* from the principle **PP4** (Define criteria for feature to sub-feature decomposition). We make this activity optional, since features can be organized in a hierarchy naturally based on their relationships and constraints, and as such, having a decomposition criteria might not be suitable in all cases. Lastly, we derived the activity *"Unify domain terminology"* from the principle

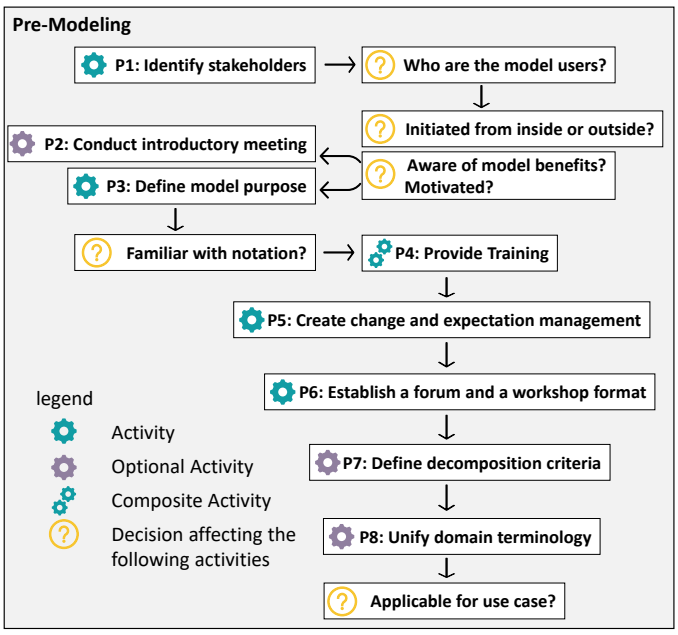


Fig. 4: Overview of phase Pre-Modeling (activities in final order after refinements)

PP2 (In immature or heterogeneous domains, unify the domain terminology). We make this activity optional as well, and only recommend performing it if the terminology is diverse and ambiguous.

5.2 Industrial Case Study

After concretizing the design of FM-PRO's *Pre-Modeling* phase, we proceeded to evaluate the applicability of this phase in practice, via an industrial based evaluation. This activity aided in quantifying rhe estimated average effort required to realise the FM-PRO *Pre-Modeling* phase.

5.2.1 Results

Table 1 shows a graphical representation of participant ratings for all activities in our process. Notably, Table 1 only comprises the ratings for the activities contained in the first version of the process; the version which was modified later for the expert evaluation. For the activity *"Define model purpose,"* participants provided a high average rating for all aspects (clarity: 4.29, applicability: 4.43, order in process: 4.14, usefulness: 4.29). One participant disagreed with the order of the activity in the process, however, they did not provide any rationale. The estimated average effort by the participants was 15 minutes. For the activity *"Identify stakeholders,"* participants also provided high ratings for all aspects (clarity: 4.71, applicability: 4.42, order in process: 4.42, usefulness: 4.42). One participant expressed the concern that it is possible that some stakeholders remain unaware of features which are very customer-specific. The estimated average effort by the participants was 10 minutes. For the *"Provide training"* activity, the ratings followed the same trend (clarity: 4.71, applicability: 4.14, order in process: 4.14, usefulness: 4.57). The participants' feedback suggested that the activity was time-taking and could be planned more efficiently. e.g., by sharing an agenda before training and improving the quality of the presentation. The estimated average effort by the participants was almost an hour (57 minutes). Participants also struggled with the sub-activity *"Tool and notation training"* attributing to the fact that the process lacked concrete guidelines on conducting it. With respect to the activity *"Establish workshop and forum format,"* we received similar ratings (clarity: 4.57, applicability: 4.29, order in process: 4.29, usefulness: 4.43). One participant gave neutral responses to both clarity and usefulness, even disagreeing with its applicability and order in the process. The estimated average effort by the participants was approximately 20 minutes. In reality, we drastically changed the execution of the phase, also opting for one-to-one interviews in the latter phases due to the challenges we faced when coordinating group sessions. For the activity *"Define decomposition criteria,"* three participants did not provide any ratings for any of the aspects (clarity: 4, applicability: 4.25, order in process: 4.25, usefulness: 4.5). We assume that the activity lacked clarity in its explanation. The estimated average effort by the participants was roughly 45 minutes. Additionally, one participant commented that the decomposition criteria should be decided at a higher level, and that developers are not well-suited to make the decision. For the activity *"Unify domain terminology,"* participants unanimously agreed on all four aspects (clarity: 5, applicability: 4.5, order in process: 4.5, usefulness: 4.75), one comment stating that the activity is especially useful in larger companies. Similar to the previous activity, the estimated average effort by the participants for this activity was also roughly 45 minutes.

5.2.2 Own Reflections

Despite the usefulness of feature models agreed upon, the employees were unclear on how to utilize them in practice. As a result, core decisions, such as *"Define model purpose"* and *"Identify stakeholders"*, were made by voting rather than taken by a responsible authority. This led to varying and at times conflicting responses. Additionally, without any explicit instruction to participate in the process, many employees chose to utilize their time in performing their routine tasks rather than spending it on feature modeling. In line with this, execution of the activity *"Create expectation and change management"* also led to vague responses. Participants mentioned that the expectations might change, however, it was unclear who would be in charge of managing changing expectations, and who to report to in such a scenario.

5.2.3 Process Refinement

In response to the feedback from the industrial evaluation as well as to our own observations, we made a few refinements in the phase. We slightly modified the description of the activity *"Provide training."* In order to enhance the effectiveness of the training, we recommend sharing the agenda of the training meeting beforehand as well as focusing on the quality of the materials employed in training. We also added an optional activity *"Conduct an Introductory meeting"* in the process, that involves conducting an introductory meeting in the host company. We recommend that the meeting should focus on motivating the stakeholders on the benefits of having a feature model. If the conductor of the process is an external person (or group), they can use the meeting as

an opportunity to meet with the participants, get an introduction to the project, and present a road-map of the steps to be followed when executing the process. It is important to lay a solid foundation in this step, because this sets the momentum for the later phases. We make it an optional activity because while important, it is possible that the stakeholders are already aware of the importance of feature models due to the maintenance and evolution challenges they are facing. Lastly, this meeting should also be used to appoint an authority figure. Such a figure can be both internal or external to the company, however, it is important that the authority figure is someone who is trusted and followed by the potential stakeholders. The above-mentioned refinements were incorporated in the second version of the process which was sent to the experts for the secondary evaluation.

5.3 Expert Evaluation

After incorporating participant feedback and applying the above-mentioned refinements, we presented the second version of our process to the experts. We now present an overview of the expert feedback for this phase.

5.3.1 Research Expert Results

Our experts provided high ratings to all four aspects (clarity: 4, applicability: 4, order in process: 5, intuitiveness: 4). Table 2 shows a summary of the median participant ratings per phase on all aspects. Notably, the order of activities in the phase received the lowest rating, with feedback suggesting to conduct the introductory meeting only after the stakeholders have been identified. Additionally, some experts suggested that the activity *"Define model purpose"* should also only be conducted after the identification of the relevant stakeholders. One expert suggested adding the potential benefits of feature models already in the introduction of the process to establish the motivation from the start. Another expert strongly advised on adding another *role* in the process description; a *method expert* who will have significant knowledge of the variability of the software system. Moreover, an expert suggested adding that the authority figure could be a business champion, who would ensure that the practices followed are aigned with the business goals, and will support the feature modeling.

TABLE 2: Expert assessment: Overview of ratings per phase

Research Experts

phase	clarity	applicability	order	intuitiveness
Pre-Modeling	4	4	4	4
Domain Analysis & Scoping	4	4	5	4
Modeling	4	4	5	4
Maintenance & Evolution	4	4	4	4

Industrial Experts

phase	clarity	applicability	order	intuitiveness
Pre-Modeling	4	4	4	4
Domain Analysis & Scoping	4	4	4	4
Modeling	4	4	4	4
Maintenance & Evolution	4	4	4	4

5.3.2 Industrial Expert Results

The industrial experts provided high median ratings to all four aspects (clarity: 4, applicability: 4, order in process: 4, intuitiveness: 4). Table 2 shows a summary of median participant ratings per phase on all aspects. The industrial experts scores are similar to the research experts scores but their suggestions are mostly regarding the clarity of the activities. They suggest to elaborate *"Define decomposition criteria"* and *"Unify domain terminology"* by providing example techniques. One suggestion regarding the order of activities where to move some questions before training is provided. A final suggestion was to make the activities *"Define decomposition criteria"* and *"Unify domain terminology"* mandatory instead of optional.

5.3.3 Process Refinement

Based on the expert feedback, we also made some more refinements in the process, which led to its final version. We re-ordered the first few activities in the phase, specifically, the phase in its final version starts with the activity *"Identify stakeholders"* followed by the optional activity *"Conduct an Introductory meeting."* After the introductory meeting, we recommend modelers to *"Define model purpose"* and *"Provide training."* Furthermore, we extended the introduction of the process by adding the various purposes feature models can serve in order to build a motivation for feature modeling. In the activity *"Identify features,"* we added the role of *method experts* as suggested by one of our experts, also adding a description of the role. We extended the activity *"Conduct an Introductory meeting"* to elaborate that the authority figure could also be a business champion. Finally, we added a question at the end of the *Pre-Modeling* phase, asking whether FM-PRO is applicable, which is a decision that should be made by the stakeholders involved in that phase.

5.4 Open-Source Case Study

In this study, FM-PRO pre-modeling activities were generally not applicable to the scenario at hand. For starters, the only identified stakeholder in this context, is a close collaborator included in the evaluation process of FM-PRO. This stakeholder neither has vast experience in feature modeling nor did he have any familiarity with FM-PRO prior to this open-source case study. We begun by holding an introductory meeting with the subject the we provided him with the official technical documentation of FM-PRO. However, pre-modeling activities such as training provision as well as change and expectation management planning were not explicitly provided nor applicable in this instance.

6 Domain Analysis and Scoping

We now discuss how we engineered the *Domain Analysis and Scoping* phase, following the same structure as above. Figure 5 shows all *Domain Analysis and Scoping* activities in their final order after all refinements.

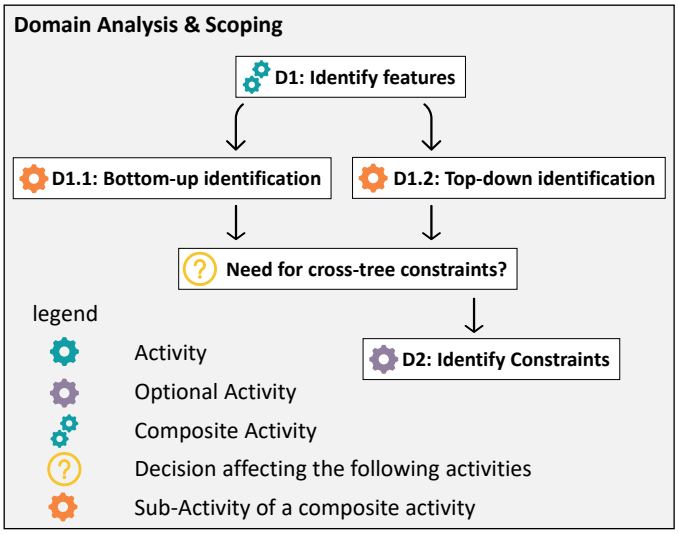


Fig. 5: Overview of phase Domain Analysis and Scoping (activities in final order after refinements)

6.1 Design of FM-PRO

Taking inspiration from principle **PP5** (Plan feature modeling as an iterative process), we designed the phase to be iterative, and concurrent to the *Modeling* (cf Sec. 7.1) phase. We also took inspiration from **M1** (Use workshops to extract domain knowledge) for this phase, however, as explained above, upon encountering challenges in the execution, we propose switching to one-to-one interviews in Sec. 6.2.3. We derived the first activity *"Identify features"* from principles **M6** (A feature typically represents a distinctive, functional abstraction), **M2** (Focus first on identifying features that distinguish variants), and **M10** (Prefer Boolean type features for comprehension). Inspired from principles **M3** (Apply bottom-up modeling to identify differences between artifacts) and **IS1** (Rely on domain knowledge and existing artifacts to construct a feature model), we define the sub-activity *"Top-down feature identification."* Additionally, inspired from principle **M4** (Apply top-down modeling to identify differences in the domain), we create a sub-activity for *"Bottom-up feature identification."* Taking inspiration from principle **M5** (Use a combination of top-down and bottom-up modeling), we recommend companies to perform both analyses if there are already a few variants in place. Lastly, based on principle **M11** (Document the features and the obtained feature model), we recommend modelers to document the features to be modeled in the next phase (cf Sec. 7.1). We derived the second activity *"Identify constraints"* based on principles **D1** (If the models are configured by (company) experts, avoid feature-dependency modeling) and **D2** (If the main users of a feature model are end-users, perform feature-dependency modeling). We recommend modelers to perform the activity only if the model is to be used by end-users of the products.

6.2 Industrial Case Study

As mentioned above (Sec. 3.3), we conducted both interviews (i.e., top-down analysis) and manual code inspection (i.e., bottom-up analysis), which led to two feature models to be merged.

6.2.1 Results

Our final feature model comprised 48 features in total, 17 of which were mandatory whereas 22 were optional. There was significant overlap between the feature models, however, both differed in terms of the used terminology as well as feature hierarchy. The features extracted from interviews were business-oriented and coarse-grained. The features extracted from user interface were also relatively coarse-grained, and were mainly domain-oriented. Lastly, the features extracted from code inspection were technical, fine-grained, and solution-oriented.

For the activity *"Identify features,"* participants generally agreed on all four aspects, with clarity and usefulness receiving two "neutral" votes each (clarity: 3.83, applicability: 4.6, order in process: 4.5, usefulness: 4.3). The estimated average effort of the activity was 46.6 hours. Upon investigation, we found out that the estimates corresponded to the different strategies for feature identification, with lower estimates corresponding to top-down analysis, and higher ones to bottom-up analysis. For the activity *"Identify constraints,"* all aspects received high ratings (clarity: 4.3, applicability: 4.6, order in process: 4.6, usefulness: 4.3), with usefulness receiving two neutral votes. This aligns with our notion that the activity is only useful if the end-users of the feature model are not experts, because otherwise, the knowledge of constraints would already be in the experts' minds, and as such, would not require expensive constraint identification. The estimated average effort of the activity was 26 hours, with one comment stating that the larger the code base, the greater the tangling of features will be, requiring longer times for constraint identification.

6.2.2 Own Reflections

Very early in the bottom-up analysis, we discovered that the process comprised insufficient guidelines on extracting features when there is only one variant in place (i.e., reactive adoption), which was the case in the company. It was also unclear what data sources to consider when extracting features from a stand-alone variant. Consequently, we devised a strategy to extract features from the variant under analysis, which we present as a potential refinement in Sec. 6.2.3.

As mentioned above, we observed coordination and scheduling challenges during the *Pre-Modeling* phase. Specifically, it was difficult to find timeslots where all stakeholders were simultaneously available. Consequently, we adapted our methodology to conduct the *Domain Analysis and Scoping* as well as modeling phases in a one-to-one interview format, iteratively for each participant. We present this adaptation as a refinement in Sec. 6.2.3.

As mentioned above, we conducted one-to-one interviews with our participants. This lead to four feature models from two developers and two customer representatives. The feature models differed greatly, with the ones from the developers comprising very technical features, and those from customer representatives comprising features capturing expected customer interaction. The process lacked guidelines on how to merge such feature models. Consequently, merging the different perspectives was challenging, as combining hard-wired features with descriptive, interactive features is difficult.

Additionally, the feature models from the top-down and bottom-up analyses also needed to be merged to have the unified feature model. The process also lacked guidelines on how to approach this. Consequently, we improvised, and devised a strategy for merging the feature models from different types of analyses. We present our strategy as a potential refinement in the following section (Sec. 6.2.3).

6.2.3 Process Refinement

We extended the sub-activity *"Bottom-up feature identification"* to also cater for single variants (as in reactive adoption), where performing a commonality and variability analysis is not a viable solution (as in extractive adoption). Modelers can start with the analysis of the user interface to gain an overview understanding of the variant as well as interact with the system to identify the functionality meaningful to end-users. Next, they can look into the codebase to identify functionality corresponding to the identified features. It is possible that they find distinct features from the codebase that were not apparent from the user interface analysis. They can consider other sources to identify features and their dependencies as well, including commit messages, pull requests, and user stories. For identifying feature groups, modelers can look into the conditional inclusion at the code level. Once all features are modeled, modelers can consult product documentation to consolidate the terminology of the identified features if needed.

In the first version of FM-PRO, we suggested a workshop format for process execution. We observed that while a workshop format was successful for the introductory meeting and the kick-off meeting (i.e., phase 1), it did not scale well for the later phases. Consequently, for top-down analysis, we changed the activity to prescribe one-to-one interviews for feature and constraint identification to facilitate coordination and allow for varying points of view. We recommend keeping the number of consulted experts low to scale the interviews and prevent the challenges when merging multiple feature models.

We added an activity "Merging multiple feature models" in the *Domain Analysis and Scoping* phase. When merging feature models from different stakeholders, modelers can consult the product's documentation to resolve conflicts regarding terminology. For conflicts in feature relationships and constraints, modelers should prioritize the stakeholders that will likely be the end-users of the feature model. Of course, it is natural to defer some decisions until the validation stage, and factor in the stakeholders' opinion when making them. When merging the feature models from the top-down and bottom-up analyses, it is likely that modelers will find features in the latter that refine features from the former. Again, in the event of conflicts, modelers can refer to the product's documentation as well as relevant stakeholders. We also recommend conducting the merging iteratively, and ideally, versioning the evolution to enable modelers to track changes at a finer level.

6.3 Expert Evaluation

Through a survey involving both academic and industrial feature modeling experts, useful feedback was gathered in the form of ratings that led to relevant process improvements of our *Domain Analysis and Scoping* phase.

6.3.1 Research Expert Results

Our experts provided high median ratings to this phase with respect to all aspects (clarity: 4.5, applicability: 4, order in process: 5, intuitiveness: 4). Firstly, multiple experts recommended against not identifying constraints if the end-users of the model were experts (already having knowledge of the constraints). The reasons were multi-fold. One expert remarked that the feature model loses its value if it cannot be used for automated product derivation. Another expert commented that it is likely that the constraints which the experts are not sure about get forgotten, and consequently, not recorded at all. Additionally, one expert stated that experts leaving the company at some point could also result in loss of information pertaining to the constraints if not recorded. Secondly, one expert remarked that while interviews are a viable solution for top-down feature extraction for small- and medium-sized systems, they might not scale for very large software systems. Having many experts could lead to many feature models to be merged, with potentially a large number conflicts. Thirdly, one expert remarked that during feature identification, one of the most difficult aspects is to find suitable abstractions, and recommended identifying features by the *capabilities*[3] they provide, typically those aligning with business value. Lastly, one expert stated that there is a lack of guidance on feature granularity.

6.3.2 Industrial Expert Results

Our industrial experts provided the following median rating's clarity: 4, applicability: 3, order in process: 4 and intuitiveness: 4. Worth noting regarding the neutral score of applicability is that no suggestions where made to improve or change this phase. Instead the experts pointed out concerns about the sub-activity *"Bottom-up feature identification"* taking substantial effort in time escpecially with respect to alignment on what the term feature means. Moreover, the experts also suggested that modeling constraints should be a mandatory activity.

6.3.3 Process Refinement

We made four improvements in the *Domain Analysis and Scoping* phase. First, we extended the description of the activity *"Top-down feature identification"* to recommend modelers to conduct interviews by grouping the same types of experts into one interview (e.g., one interview with developers and another with product managers). This helps eliminate conflicting viewpoints and lead to a lower number of feature models to be merged. Later, modelers can switch back to the workshop format for, e.g., the *"Validation"* activity. Secondly, we leave the decision of identifying and modeling constraints to the modelers, adding guidelines that can help them decide. Specifically, in the activity *"Identify constraints"*, we recommend modelers to not model constraints if they want to save the cost and effort to identify and model them. However, in other cases, they should be modeled based on the above-mentioned reasons by our experts (cf. Sec. 6.3). Thirdly, for *"Bottom-up feature identification"*, we specified that for finding suitable

3. Capabilities are special features, which are not directly implemented, but are abstractions that indicate the functionality a feature provides.

feature abstractions, modelers can refer to the capabilities the features should provide. Lastly, for *"Identify features"*, we added that it is up to the modelers to decide on their level of granularity—a core strength of features, since they completely abstract over implementation artifacts.

6.4 Open-Source Case Study

Here, we selected our preferred product line, extracted feature relationships and meta-data, while identifying the overall relevance of such system extraction.

Our product line of choice is *Video.js*[4]. The *Video.js* system can be found within the Qualitas.js Dataset[5]) as part of the ESPLA catalog [26]. As one of the six open source systems manually transformed into a software product line using the RiPLE-HC [92] approach, the experimental data from that study shows and offers a total of *13 features*, organised into *38 modules* represented by *7939 LOC*. Our selection goal was to find a small to medium sized project with a heterogeneous set of features, dependencies and constraints. Per the scope of FM-PRO, we sought to derive a feature model from the assets accrued and validate it accordingly.

In the feature identification step, we realized that the embedded Video.js features had already been identified and made readily available to the stakeholder performing this evaluation via the RiPLE-HC GitHub repository, through direct sources, such as an experiment conducted to analyze the impact of RiPLE-HC on code organization and feature location maintenance tasks.

From the documented modules[6] of *Video.js* it was quite difficult to identify and extract existing constraints. However, with the relationships, it was fairly easy to identify or infer mandatory, optional, OR, and alternative relationships from the artifacts available. Some inferences were obvious while others required a bit more effort through code analysis.

7 Modeling

We engineered the *Modeling* phase as follows. Figure 6 shows all *Modeling* activities in their final order after all refinements.

7.1 Design of FM-PRO

Taking inspiration from principle **MO1** (The depth of the feature-model hierarchy should not exceed eight levels), we recommend modelers to aim for a shallower depth than eight levels in the feature model. Additionally, taking inspiration from principle **MO3** (Split large models and facilitate consistency with interface models), we suggest modelers to decompose the feature model into smaller ones if the identified features and dependencies are large and complex. We derived the first activity *"Model modularization"* to provide guidelines on decomposing feature models. We suggest modelers to use a feature decomposition criteria when decomposing feature models, e.g., different feature models for each purpose. Next, we derived the second activity *"Define coarse feature hierarchy"* from principles **MO5** (Maximize cohesion and minimize coupling with feature groups) and **MO2** (Features in higher levels in the hierarchy should be more abstract). We defined the third activity *"Add features"* to provide guidelines on adding the identified features in the feature model. Taking inspiration from principle **M8** (Define default feature values), we recommend defining default values for features when documenting them to facilitate feature model configuration. We derived the fourth activity *"Model constraints"* by taking inspiration from principles **MO3** (Split large models and facilitate consistency with interface models) and **MO4** (Avoid complex cross-tree constraints). The activity is optional, and only required if modelers have performed constraints identification (*"Identify constraints"*) in the *Domain Analysis and Scoping* phase. Next, we derived the fifth activity *"Define views"* directly from principle **M9** (Define feature model views), that suggests creating stakeholder-specific views by selecting only a subset of features that is relevant. Lastly, we grouped all sub-activities related to feature model validation in the sixth activity *"Validation."* The activity comprised three sub-activities. The first sub-activity *"Stakeholder reviewing"* was inspired from principle **QA1** (Validate the obtained feature model in workshops with domain experts). The second sub-activity *"Perform product derivation"* was inspired from principle **QA2** (Use the obtained feature model to derive configurations). Lastly, the third sub-activity *"Regression testing"* was inspired from principle **QA3** (Use regression tests to ensure that changes to the feature model preserve previous configurations).

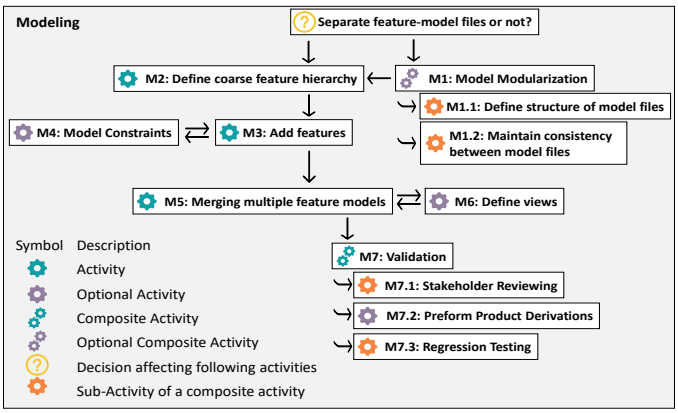


Fig. 6: Overview of phase Modeling (activities in final order after refinements)

7.2 Industrial Case Study

The modeling activities in our industrial case study were carried out as follows and led to the following outcomes.

For the activity *"Model modularization,"* participants provided high average ratings on all aspects (clarity: 4.6, applicability: 4.3, order in process: 4.5, usefulness: 4.3). The estimated average effort of the activity was approximately 6.5 hours. Regarding the activity *"Define coarse feature hierarchy,"* participants provided high average ratings (clarity: 4.6, applicability: 4.6, order in process: 4.5, usefulness: 4.3), with order in process and usefulness receiving one and two neutral votes respectively, each lacking a rationale. The

4. https://github.com/videojs/video.js/tree/main
5. http://aserg.labsoft.dcc.ufmg.br/qualitas.js/
6. https://docs.videojs.com/

estimated average effort of the activity was approximately 13 hours. For the *"Add features"* activity, we observed a similar pattern (clarity: 4.5, applicability: 4.6, order in process: 4.5, usefulness: 4.3). All participants estimated the activity to take roughly eight hours. Regarding the activity *"Model constraints,"* we received almost similar ratings to the previous activity (clarity: 4.6, applicability: 4.6, order in process: 4.5, usefulness: 4.3). The estimated average effort of the activity was approximately 6 hours. *"Define views"* was the lowest rated activity in the process (clarity: 3.16, applicability: 3.3, order in process: 2.83, usefulness: 3.3). One participant suggested using examples to enhance the understandability of the activity as well as demonstrating how the views will look like after performing this activity. Only one participant estimated the effort of the activity, approximating it to take 30 hours. The high estimate could potentially be explained by the lack of details on how and when to conduct it. The activity *"Validation"* received high ratings (clarity: 4.6, applicability: 4.3, order in process: 4.5, usefulness: 4.5), the applicability and usefulness receiving one neutral vote each without explanation. One participant suggested that validation through product derivation should also discuss how to think about variations with only one variant in place, which was the case in the company. The estimated average effort of the activity was 42 hours. We hypothesize that since the activity was long-term, and could be performed repeatedly and in multiple ways (i.e., *"Stakeholder reviewing"*, *"Perform product derivation"*, and *"Regression testing"*), the participant gave a high ballpark figure.

Based on our interactions with one of the industrial participant, creating the feature model facilitated the development of a collective understanding within the company regarding which features should be considered optional and which constitute the core product. Initially, the system lacked any variations, resulting in features now identified as optional being mandatory for each customer of the product. This undertaking served as a catalyst for launching initiatives aimed at transitioning towards a more adaptable and variable system platform.

7.2.1 Process Refinement

We made slight modifications in this phase. In the introduction of the *Modeling* phase, we elaborate that (in addition to modeling the identified features and constraints,) the phase also involves validating that the features are modeled correctly and lead to valid configurations. This makes the activities *"Perform product derivation"* and *"Regression testing"* more suitable for the *Modeling* phase.

7.3 Expert Evaluation

Research modeling experts provided useful feedback through a survey in the form of ratings that led to relevant process improvements of our *Modeling* phase.

7.3.1 Research Expert Results

The experts provided high median ratings on all aspects (clarity: 4, applicability: 4, order in process: 4, intuitiveness: 5). One expert remarked that instead of defining default feature values (in the activity *"Add features"*) as we recommend, it is easier to define default configurations. Another expert remarked that the activities *"Define views," "Perform product derivation,"* and *"Regression testing"* should be seen separately from the purely modeling-related activities. In contrast to this, one expert remarked that *"Perform product derivation"* can already be applied right after the activity *"Model constraints,"* which would help experts understand if the newly added constraints lead to valid configurations. Additionally, one expert suggested that it would be useful to clarify when conducting the activity *"Define views"* would be helpful. Lastly, one expert suggested that the activities *"Merging multiple feature models"* and *"Perform product derivation"* should be optional, as they are only required in certain contexts.

7.3.2 Industrial Expert Results

The industrial experts also provided high median ratings on all aspects (clarity: 4, applicability: 4, order in process: 4, intuitiveness: 4). One expert mentioned that merging of different models was not clear to them and suggested that we could clarify it by providing an example. The expert also suggested that solving basic problems by different perspectives during merging feels to late in the process. Additionally, two experts remarked that regression testing was unclear and might be more suited to the *Maintenance and Evolution* phase instead.

7.3.3 Process Refinement

We made three additional improvements in this phase. First, we extended the activity *"Add features"* by also suggesting that modelers can define default configurations instead of defining default feature values. However, since configuration is typically achieved via reconfiguration,[7] we believe that defining default feature values is more beneficial. Second, we extended the description of the activity *"Define views"* to clarify that it should be performed when the feature model becomes too large to only view a subset of features in order to facilitate configuration. Finally, we made both the activities *"Merging multiple feature models"* and *"Perform product derivation"* optional, and clarified when to perform them. When merging multiple feature models, in a top-down modeling case, selecting the best model candidates for the merger, is decided by the stakeholders identified. In a bottom-up modeling scenario on the other hand, model validation and merging is specifically delegated to the developers. Overall, in the event of any contention relating to mergers and model validation, the final authority lies in the hands of the feature modeling domain expert in charge of the process.

7.4 Open-Source Case Study

Our open-source case study was realized as follows. We begun by describing the modeling process. The following activities defined our *Video.js* modeling phase. (i) Model modularization. (ii) Coarse feature hierarchy definition. (iii) Creation of an initial model version of the system. (iv) System model refinements.

7. An initial configuration is automatically created using an algorithm based on default feature values, and then modified by users to create the final configuration.

TABLE 3: Feature groups

function	module	associated functionalities
Base	control-bar	control-bar.js, live-display.js, play-toggle.js, progress-control.js, time-display.js, volume-control.js, volume-menu-button.js
Base	media	flash-extern.js, flash.js, html5.js, loader.js, media.js
BigPlayButton	–	big-play-button.js, button.js
FullScreen	control-bar	full-screen-toggle.js
FullScreen	–	full-api.js
LoadingSpinner	–	loading-spinner.js
Mute	control-bar	mute-toggle.js
PlaybackRate	control-bar	playback-rate-menu-button.js

We performed model modularization via code inspections, manual documentation analysis, and checking observable system characteristics. Connections between features were inferred. Likewise, via the same methods, we were able to trace the origins and associations that exist amongst feature source code files, to understand the potential hierarchy of the embedded functionalities. Table 3 lists a selection of internal capabilities captured within *Video.js* modules. Class properties, such as inheritance and extensions, were also used to further decompose functionalities gathered into coherent sub-trees. For instance the *Base* function containing two modules; namely *control-bar* and *media* can be observed.

To model an initial coarse feature hierarchy from previous domain analysis and feature organisation activities, we begun by creating feature decompositions from the system functionalities previously analysed. This step was supported by the official *Video.js* documentation as well as existing experimental data on features and their relationships, available on GitHub.

Decompositions such as mandatory, optional and OR groups were applied to multiple stand-alone features and sub-trees. For example in Fig. 7, the parent *ControlBar* feature has three optional sub features modeled in addition. Nonetheless, forming such model decompositions came with a few challenges. For example, in some group cases, it was quite easy to find a feature within the pool of available features that sufficed as a top-level feature for that decomposition. i.e., the *ControlBar* feature being a parent to the *PlayRate*, *Mute* and *Fullscreen* features. However in other cases where a feature did not strongly correlate to any available feature, an interim top-level feature which sufficiently abstracts that selection of the sub-features in question was assigned.

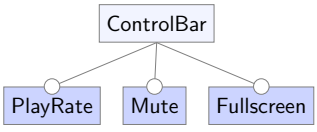


Fig. 7: ControlBar sub-tree

The first version of the *Video.js* model was created by merging the features identified together with their decompositions into one working model. The initial model we created was primarily constructed based on our intuition of how the various decomposed features and groups fit together. Figure 8 shows our very first model of *Video.js*, formed by merging all our decomposed features and feature groups.

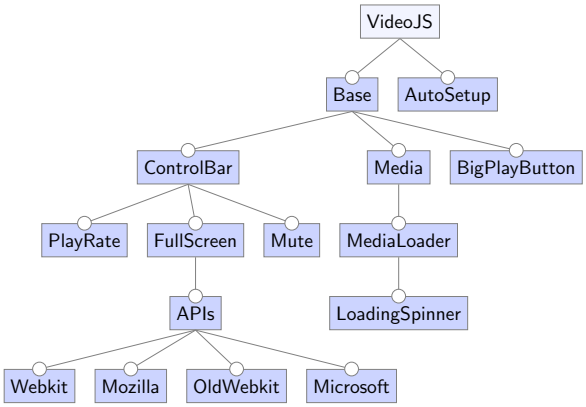


Fig. 8: Initial Video.js model

After successfully creating an initial version of our model, we proceeded to perform further model refinements by reviewing each feature to verify whether they can be absorbed by another feature, or intuitively moved to another branch or sub-tree in the hierarchy.

Considering Fig. 9, redundant intermediate features such as *Media* and *MediaLoader* were completely removed from the hierarchy. Features such as *LoadingSpinner* and *APIs* together with all its sub-features were moved to the *BigPlayButton* feature branch. These modeling decisions were influenced by more in-depth codebase inspections and capability reassessments of *Video.js* via interactions with a running instance of the application.

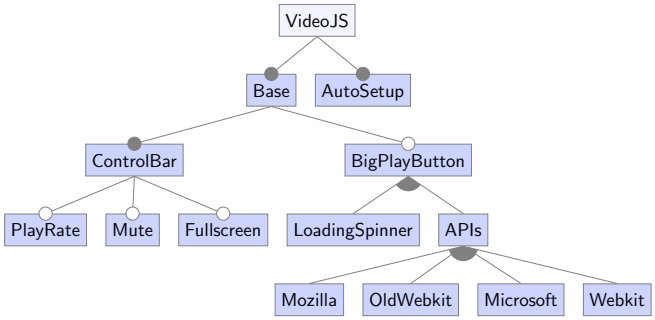


Fig. 9: Final Video.js model

8 Maintenance and Evolution

As structured above, we now discuss how we engineered the *Maintenance and Evolution* phase. Figure 10 shows all *Maintenance and Evolution* activities in their final order after all refinements.

8.1 Design of FM-PRO

Taking guidance from principle **MME1** (Use centralized feature model governance), we suggest that the feature

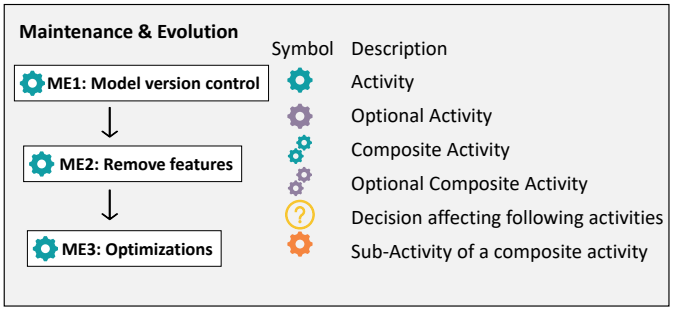


Fig. 10: Overview of phase Maintenance & Evolution (activities in final order after refinements)

model should only be governed and evolved by a limited number of stakeholders. Additionally, taking inspiration from principle **MME3** (New features should be defined and approved by domain experts), we suggest that new features should be added only after their approval from domain experts. We derived the first activity *"Model version control"* directly from principle **MME2** (Version the feature model in its entirety), recommending modelers to version entire feature models instead of individual features. Next, we derived the second activity *"Remove features"* to provide guidelines on removing features. The activity was not inspired from any principle, however, we deem feature removal an important task, and believe that modelers need to be cautious when removing features so as to do it in an effective manner. We derived the last activity *"Optimizations"* to recommend modelers to refactor the feature model to optimize the hierarchy and constraints when needed.

8.2 Industrial Case Study

Since this phase is continuous and longitudinal, it was beyond the scope of this work and we did not execute this phase in the company. Recall our note on the scope of the process and our discussion below. To provide some guidance, we still included the activities of this phase in the questionnaire but kept them optional. Only two participants filled the questionnaire for the activity *"Model version control,"* providing high average ratings to all aspects (clarity: 4.5, applicability: 5, order in process: 5, usefulness: 5). One participant estimated the effort of the activity to be 15 hours.

8.3 Expert Evaluation

Our research and industrial expert evaluation results, with related process refinements introduced are outlined in this section. We highlight key findings and insights gathered from the analysis conducted in preceding phases.

8.3.1 Research Expert Results

From our experts, we received high median ratings to all activities in this phase with respect to all aspects (clarity: 5, applicability: 4, order in process: 4, intuitiveness: 4.5). Three experts remarked that the phase could be elaborated more, one remarking that it is rather abstract than the other phases. One expert remarked that some activities in the phase could be performed in parallel. Most other comments were suggestions for refinement. For example, one expert remarked that when removing a feature, it is important to make it part of the change process, as a defect in one variant can be a feature in another one. Another expert suggested adding drivers behind maintaining the feature model. One expert suggested establishing clear communication channels between developers and modelers so the modelers can be notified of changes that need to be reflected in the feature model. Lastly, one expert remarked that feature model maintenance is especially tricky when features are linked to the code in any way.

8.3.2 Industrial Expert Results

Our industrial experts provided high median ratings to all aspects in this phase (clarity: 5, applicability: 4, order in process: 5, intuitiveness: 4). Two industrial experts suggested to put more information on how to *"Remove feature"* and *"Optimizations"* as they were expecting more guidence for these activities. Another expert stated that the phase was clear and straight forward.

8.4 Process Refinement

This phase only provides very basic support. We acknowledge that it can be extended to add more details in the activities as well as the introduction of *Maintenance and Evolution* in the process (e.g., adding the driving forces). However, in FM-PRO, we consider feature model evolution as an isolated activity, where modelers and other stakeholders have discussed and agreed upon a change already, such as deletion of a feature. We consider it an isolated activity mainly because of two reasons. First, feature model maintenance and evolution beyond the actual model editing activities are beyond the scope of feature model construction. This means that while FM-PRO primarily focus on the process of constructing the feature model, the ongoing tasks of maintaining and evolving the model are treated separately. Still, we provide some guidelines on how the feature models can be versioned and updated to guide modelers in maintaining and evolving feature models. Second, feature model maintenance and evolution, especially in relation to regular software evolution as well as modern development practices (e.g., continuous development, iterative development, and SCRUM) is arguably broad enough to constitute a process of its own. Consequently, we kept the phase brief, only adding the clarification that some activities can be performed concurrently based on feedback. Creating a dedicated and holistic process for feature model maintenance and evolution constitutes valuable future work.

9 Discussion and Future Work

We now discuss our findings and experiences, provide further suggestions for using FM-PRO, and describe valuable future research directions and extensions of FM-PRO.

9.1 Evolution and Maintenance

As mentioned above (Sec. 8.4), the focus of FM-PRO (and the principles that steer it) is feature model creation, specifically when adopting or transitioning towards a platform. Feature model evolution, as pointed out by our experts, is a long-running and frequent activity. One expert remarked that the *Maintenance and Evolution* phase requires activities

about feature evolution. An example of feature evolution is the default value of a feature being changed to true, implying that while initially introduced as a novelty, the feature became mandatory. Another expert suggested that, a use case where there is an existing feature model, which is being refined by adding another feature model, might be useful. Additionally, considering that features can be explicitly linked to code assets, it can happen that after changes in code, there are changes that need to be made in the feature model. Moreover, and as one expert suggested, using an existing feature in a variant can be more complex and tricky than using a newly added feature. In fact, the co-evolution of code and models, as well as refactoring, is an open research problem [93]–[95] and requires dedicated automated support. In our future work, we aim to present FM-PRO-EVO, a process for feature model maintenance and evolution. We intend to specify manual support for the above-mentioned scenarios that can be used as a guideline by tool developers, for implementing support for automated feature model maintenance and evolution, especially in relation to the codebase. FM-PRO-EVO also needs to incorporate change management, as we discuss next.

9.2 Process Integration

Future work comprises process integration. Based on results expected from future longitudinal studies of applying the process on industrial cases, we can extend the process to offer guidelines on integrating the process with existing agile development processes, such as SCRUM, XP, CI/CD, and FDD. Additionally, the process can be extended to elaborate how to align the application with ordinary change management. For example, one of our experts suggested that when removing a feature, it is important to make this a part of the change process. The rationale is that a defect in one product could be considered a feature in another product. Therefore, until all variants comprising the feature have accepted it as a commodity, the feature should not be removed. Similarly, during clean-up, features that always get selected can be removed as a selectable feature and introduced as a core feature, also through a well-defined change process.

Additionally, one expert remarked that it is important to establish clear communication channels between the developers and modelers so the latter can be notified of changes in code that require a change in the feature model. In future work, in FM-PRO-EVO, we aim to offer guidelines to establish such communication challenges that facilitate the co-evolution of code and feature models.

Lastly, one expert remarked that some additional guidelines on change tracking of feature models can be useful. A future direction for FM-PRO-EVO is to include elaborations on how version-control systems and ordinary file information (e.g., last modified time of a file) can be used to track the changes in the feature model.

9.3 Effort Estimation and Cost/Benefit Assessment

For our industrial case study, adding the average effort estimates by the participants per phase, the *Pre-Modeling*, *Domain Analysis and Scoping*, *Modeling*, and *Maintenance and Evolution* phases take 3, 72, 105, and 15 hours respectively. While these estimates are telling, they are subjective and as such, cannot be used when planning the construction of the feature model should a company decide to undertake the endeavor. One of our experts expressed the concern that conducting a five-day workshop with 10 people can easily lead to 50 work days, which is equivalent to two person-months of a project. While we argue that such effort is required one time only, we believe that more realistic estimates are needed for better planning.

A future direction is to conduct longitudinal studies of the process application for getting more realistic effort estimates. Based on that, we can refine activities by, for example, putting thresholds on the number of people (e.g., the maximum team-size in SCRUM is 10 people), as well as the maximum amount of time to spend on an activity or a phase. Additionally, we point future researchers to conduct cost and benefit assessments of creating a feature model using our process (for example, with respect to feature location time and product delivery time). The results of such studies in projects of different scales can themselves act as guidelines on whether such an effort is worthwhile and if so, what are the expected benefits.

9.4 Tools

FM-PRO is designed to be tool-independent, only requiring a feature-modeling language that has at least the concepts listed in Sec. 4. We believe that a textual modeling language is best to model features. Textual modeling languages are generally favored in industry due to their simplicity and ease of adoption. They provide a clear and straightforward way to represent features without the overhead of graphical tools. Among several textual modeling languages Clafer [34] and UVL [40], [41] are among the most notable. Clafer is known for its minimal and concise syntax, making it easier for developers to learn and use, while also offering decent tools support. On the other hand, UVL, a more recent community initiative, offers extensive tool support and a more structured syntax. While UVL's comprehensive tool support can be advantageous, its less succinct syntax (e.g., keyword-heavy) and stringent structure can make textual models harder to understand.

Regardless of which modeling language is used, the following tools can support FM-PRO. In the phase Pre-Modeling, Confluence[8] can be used to collaboratively organize all the planning and pre-modeling related information, such as the form or workshop organization. Next, in the phase Domain Analysis and Scoping, miro[9] can be used collaboratively to perform visually intuitive mind-mapping activities, for instance, to quickly record identified features and annotate relevant findings. In the next phase, the feature models can also be created visually, such as using FeatureIDE,[10] which provides the original visual representation of feature models, while also allowing to visually represent constraints. In this phase, Git is useful for recording and then merging collaboratively created feature models, while it also supports the phase Maintenance and Evolution phase with model versioning, refactoring, and general change

8. https://www.atlassian.com/software/confluence
9. https://miro.com/
10. https://featureide.github.io

management. If used, the tools should be part of the activity *"Tool and notation training"*.

While FM-PRO is designed to be tool-independent, valuable future work would be to establish tool support for the process itself, guiding applicants through its different phases and activities. By integrating functionalities from the tool mentioned above into a coherent tool, we can enhance the usability and adoption of FM-PRO in existing development processes. Ideally, such a tool should incorporate a collaboration mode, visualizations, and version control. An integrated tool would streamline the workflow, reduce context switching, and provide a unified platform for all FM-PRO activities, thereby facilitating better collaboration and efficiency.

10 Threats to Validity

Construct Validity. Evaluating a process is challenging. A general problem is that applying a new process is substantial effort for a company, which companies usually only invest when a process is proven to be effective. FM-PRO falls into this category, since it requires substantial investment not only into the modeling, but also in adopting a product-line strategy, which usually involves software re-engineering effort. To enhance validity, without being able to fully perform the process (which would require longitudinal studies, which is a paper on its own and subject to our future work), we used diverse constructs, comprising two kinds of case studies and two kinds of expert surveys, triangulating from the results. We also used an iterative methodology. Furthermore, another threat is that the industrial participants in the industrial expert assessment were not knowledgeable enough to judge the process. However, we only selected those with years of feature modeling experience with a company that has been applying feature modeling for decades already. Their input is complemented with the researcher survey, which provides a cross-company perspective and included a feature-modeling tool vendor.

External Validity. A threat to external validity is that the process is too specific to our evaluation companies and to the various companies that we interviewed prior to establishing the principles (which were the basis for the process). Regardless, the latter were very diverse, including different kinds of companies [61], thus by design the process already accounts for diverse application environments. To account for the problem that the evaluation could be too specific, we not only performed an industrial case study with one company, but also an open-source case study, an expert evaluation with industrial experts from another company that has been performing feature modeling for decades, as well as with researchers from the community, to obtain a cross-industry perspective.

For the evaluation in the industrial case study, we applied the process to one of its software systems only. However, the case was substantial and complex, and we used multiple methods to collect data. In addition, the company had a core interest in migrating to a product-line platform, currently facing various challenges (e.g., delivering unnecessary code because there is no means to configure the product at runtime) that could potentially be solved by having a feature model. In fact, right after our study, the company took the initiative to start migrating the code as well. Additionally, we used multiple methods to collect data (e.g., workshops, interviews, code inspection). As such, our findings should hold for companies comparable in size and products.

For the evaluation with the open-source case study, there is the threat of the case being too artificial. However, we semi-systematically sifted through the ESPLA catalog to identify a substantial case. For the assessment with the industrial experts, a threat is that the participant roles are too specific or that the participants lack experience. However, we only invited experts on feature modeling who have years of experience, but also account for different roles in the company. Finally, our researcher survey could have targeted only a specific set of researchers. However, we not only used community contacts, but also systematically searched through publications of the relevant venues since 2020, assuring a diverse set of reviewers.

Internal Validity. Researchers and industrial participants might be biased in favor of the process. However, since one author was also employed at the company, and did not design the process, the successful creation of a feature model was more important. Consequently, the author provided deeper insights. A threat might be that the industrial participants in the industrial case study made mistakes during feature modeling. This could have happen when they were either not familiar enough with he subject system or with feature modeling. However, we made sure that the participants were the most relevant for this subject system, each having significant experience with it. Moreover, we provided training to equip participants with the required knowledge for effective participation. In the open-source case study, a threat is that the modeler was biased. However, he was neither involved in creating the process itself, nor conducting the other evaluations. While he knew feature models, he first read the process when conducting the case study, avoiding bias. In the expert evaluations, a potential threat is that the respondents were biased towards the authors. However, the respondents were not required to provide their names, and among the 20 researcher experts, only two were previous co-authors, but any conflict of interest had already expired when the survey was conducted. Finally, an additional threat is that we incorrectly analyzed the data. However, we organized several meetings to cross-check and discuss not only the methodology, but also the results, to enhance internal validity.

11 Conclusion

We presented FM-PRO [63], the first modeling process for feature models, which are among the most intuitive modeling techniques to organize features and their constraints. Feature models are the lingua franca for modeling portfolios of system variants called software product lines. Despite thousands of (mainly academic) techniques building upon feature models, no modeling process had been contributed so far, despite feature modeling being a largely manual process, since models are highly domain-specific, containing domain information about features that is not contained in the codebase [17]. In multiple iterations, we systematically

designed FM-PRO and evaluated it on a substantial industrial case, an open-source case and with feature-modeling experts from both academia and industry. Our process extends the existing processes for product-line adoption, and as such, can be incorporated in companies creating a platform from scratch or transitioning to a platform from a set of closely related systems. Our concrete activities present a clear road-map for feature model creation, which when followed correctly, lead to shared in-depth understanding of the software system as well as the variability it comprises.

Future work (as discussed) comprises the integration of FM-PRO with existing development processes, defining effective process integration patterns. Furthermore, as also noted by our experts, FM-PRO focuses on the adoption of feature models and platforms, while the maintenance and evolution typically causes long-term effort and other challenges (e.g., the consistent co-evolution of code and feature model). We plan to conduct longitudinal studies and develop a feature-model maintenance and evolution process (FM-PRO-EVO). Alongside the FM-PRO-EVO we plan to create a coherent tool that aid users of the process in completing the activities in FM-PRO and FM-PRO-EVO.

Acknowledgments

We thank our experts Slawomir Duszynski, Rick Rabiser, Jabier Martinez, Thomas Fogdal, Rafael Capilla, Heiko Koziolek, Niels Jørgen Strøm, Mehrdad Saadatmand, Stefan Stanciulescu, Mathieu Acher, Danilo Beuche, Michael Kircher, Philippe Collet, Tobias Beichter and Jesper Lysemose Korsgaard for their valuable insights and constructive feedback which helped us improve our process. We also thank our thesis students Leonhard Wermert, Malik Hanan Ahmed, and Malik Wadeed for helping with our evaluations.

References

[1] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," in *CSMR*, 2013.

[2] R. Koschke, P. Frenzel, A. P. Breu, and K. Angstmann, "Extending the reflexion method for consolidating software variants into product lines," *Software Quality Journal*, vol. 17, pp. 331–366, 2009.

[3] J. Businge, O. Moses, S. Nadi, and T. Berger, "Reuse and maintenance practices among divergent forks in three software ecosystems," *Empirical Software Engineering*, vol. 27, no. 2, p. 54, 2022.

[4] Ş. Stănciulescu, S. Schulze, and A. Wąsowski, "Forked and integrated variants in an open-source firmware project," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 151–160.

[5] J. Rubin, K. Czarnecki, and M. Chechik, "Managing cloned variants: a framework and experience," in *Proceedings of the 17th International Software Product Line Conference*, 2013, pp. 101–110.

[6] T. Fogdal, H. Scherrebeck, J. Kuusela, M. Becker, and B. Zhang, "Ten years of product line engineering at danfoss: Lessons learned and way ahead," in *SPLC*. ACM, 2016.

[7] J. Krüger, M. Mukelabai, W. Gu, H. Shen, R. Hebig, and T. Berger, "Where is my feature and what is it about? a case study on recovering feature facets," *Journal of Systems and Software*, vol. 152, pp. 239–253, 2019.

[8] T. Berger, J.-P. Steghöfer, T. Ziadi, J. Robin, and J. Martinez, "The state of adoption and the challenges of systematic variability management in industry," *Empirical Software Engineering*, vol. 25, pp. 1755–1797, 2020.

[9] P. Clements and L. Northrop, *Software product lines*. Addison-Wesley Boston, 2002.

[10] K. Pohl, G. Böckle, and F. Van Der Linden, *Software product line engineering: Foundations, Principles, and Techniques*. Springer, 2005, vol. 10.

[11] F. J. Van der Linden, K. Schmid, and E. Rommes, *Software product lines in action: the best industrial practice in product line engineering*. Springer, 2007.

[12] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-Oriented Software Product Lines*, 2013.

[13] D. M. Weiss and C. T. R. Lai, *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[14] M. T. Rahman, L.-P. Querel, P. C. Rigby, and B. Adams, "Feature toggles: practitioner practices and a case study," in *Proceedings of the 13th international conference on mining software repositories*, 2016, pp. 201–211.

[15] J. Meinicke, C.-P. Wong, B. Vasilescu, and C. Kästner, "Exploring differences and commonalities between feature flags and configuration options," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, 2020.

[16] T. Berger, S. She, R. Lotufo, K. Czarnecki, and A. Wasowski, "Feature-to-code mapping in two large product lines," in *14th International Software Product Line Conference (SPLC)*, 2010, extended Abstract.

[17] S. Nadi, T. Berger, C. Kästner, and K. Czarnecki, "Where do configuration constraints stem from? an extraction approach and an empirical study," *IEEE Transactions on Software Engineering*, vol. 41, no. 8, pp. 820–841, 2015.

[18] T. Berger, S. She, R. Lotufo, A. Wąsowski, and K.-t. Czarnecki, "A study of variability models and languages in the systems software domain," vol. 39, no. 12, pp. 1611–1640.

[19] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Software Engineering Institute, Carnegie Mellon University, Tech. Rep., 1990.

[20] T. Berger, D. Lettner, J. Rubin, P. Grünbacher, A. Silva, M. Becker, M. Chechik, and K. Czarnecki, "What is a feature? a qualitative study of features in industrial software product lines," in *Proceedings of the 19th international conference on software product line*, 2015, pp. 16–25.

[21] P. Kajsa and P. Návrat, "Design pattern support based on the source code annotations and feature models," in *Proceedings of the 38th international conference on Current Trends in Theory and Practice of Computer Science*, 2012, pp. 467–478.

[22] D. Beuche, H. Papajewski, and W. Schröder-Preikschat, "Variability management with feature models," *Science of Computer Programming*, vol. 53, no. 3, pp. 333–352, 2004.

[23] M. F. Johansen, Ø. Haugen, and F. Fleurey, "Properties of realistic feature models make combinatorial testing of product lines feasible," in *Model Driven Engineering Languages and Systems: 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings 14*. Springer, 2011, pp. 638–652.

[24] M. Mukelabai, D. Nesic, S. Maro, T. Berger, and J.-P. Steghöfer, "Tackling combinatorial explosion: A study of industrial needs and practices for analyzing highly configurable systems," in *33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018.

[25] A. Wasowski and T. Berger, *Domain-specific Languages: Effective Modeling, Automation, and Reuse*. Springer, 2023.

[26] J. Martinez, W. K. Assunção, and T. Ziadi, "Espla: A catalog of extractive spl adoption case studies," in *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*, 2017.

[27] C. Marimuthu and K. Chandrasekaran, "Systematic studies in software product lines: A tertiary study," in *21st International Systems and Software Product Line Conference - Volume A*, ser. SPLC '17, 2017.

[28] R. Bashroush, M. Garba, R. Rabiser, I. Groher, and G. Botterweck, "Case tool support for variability management in software product lines," *ACM Computing Surveys*, vol. 50, no. 1, pp. 14:1–14:45, 2017.

[29] D. Beuche, "Modeling and building software product lines with pure:: variants," in *Software Product Line Conference, International*. IEEE Computer Society, 2008, pp. 358–358.

[30] C. W. Krueger, "Biglever software gears and the 3-tiered spl methodology," in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, 2007, pp. 844–845.

[31] J. Meinicke, T. Thüm, R. Schröter, F. Benduhn, T. Leich, and G. Saake, *Mastering Software Variability with FeatureIDE*. Springer.

[32] T. Ziadi, L. Hélouët, and J.-M. Jézéquel, "Towards a UML profile for software product lines," in *Software Product-Family Engineering*.

[33] T. Ziadi and J.-M. Jézéquel, "Software product line engineering with the UML: Deriving products," in *Software Product Lines*. Springer, pp. 557–588.

[34] K. Bąk, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wąsowski, "Clafer: unifying class and feature modeling," *Software & Systems Modeling*, vol. 15, pp. 811–845, 2016.

[35] C. Seidl, T. Winkelmann, and I. Schaefer, "A software product line of feature modeling notations and cross-tree constraint languages," *Modellierung 2016*, 2016.

[36] M. Antkiewicz and K. Czarnecki, "Featureplugin: Feature modeling plug-in for eclipse," in *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, 2004, pp. 67–72.

[37] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, "Featureide: An extensible framework for feature-oriented software development," *Science of Computer Programming*, vol. 79, pp. 70–85, 2014.

[38] M. H. t. Beek, K. Schmid, and H. Eichelberger, "Textual variability modeling languages: an overview and considerations," in *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume B*, 2019, pp. 151–157.

[39] P. Franz, T. Berger, I. Fayaz, S. Nadi, and E. Groshev, "Configfix: Interactive configuration conflict resolution for the linux kernel," in *43rd International Conference on Software Engineering, Software Engineering in Practice track (ICSE/SEIP)*, 2021.

[40] C. Sundermann, K. Feichtinger, J. A. Galindo, D. Benavides, R. Rabiser, S. Krieter, and T. Thüm, "Tutorial on the universal variability language," in *Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume A*, 2022, pp. 260–260.

[41] C. Sundermann, S. Vill, T. Thüm, K. Feichtinger, P. Agarwal, R. Rabiser, J. A. Galindo, and D. Benavides, "Uvlparser: Extending uvl with language levels and conversion strategies," in *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B*, 2023.

[42] T. Berger and P. Collet, "Usage scenarios for a common feature modeling language," in *First International Workshop on Languages for Modelling Variability (MODEVAR)*, 2019.

[43] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration using feature models," in *Software Product Lines: Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004. Proceedings 3*. Springer, 2004, pp. 266–283.

[44] A. Hubaux, D. Jannach, C. Drescher, L. Murta, T. Männistö, K. Czarnecki, P. Heymans, T. Nguyen, and M. Zanker, "Unifying software and product configuration: A research roadmap," in *CONFWS*, 2012.

[45] H. Gomaa and M. E. Shin, "Automated software product line engineering and product derivation," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. IEEE, 2007, pp. 285a–285a.

[46] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki, "Reverse Engineering Feature Models," in *International Conference on Software Engineering*, ser. ICSE. ACM, 2011, pp. 461–470.

[47] N. Andersen, K. Czarnecki, S. She, and A. Wąsowski, "Efficient synthesis of feature models," in *International Software Product Line Conference*, ser. SPLC, 2012.

[48] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Information systems*, vol. 35, no. 6, pp. 615–636, 2010.

[49] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake, "A classification and survey of analysis strategies for software product lines," *ACM Computing Surveys*, vol. 47, no. 1, pp. 6:1–6:45, 2014.

[50] F. Ensan, E. Bagheri, and D. Gašević, "Evolutionary search-based test generation for software product line feature models," in *Advanced Information Systems Engineering: 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings 24*. Springer, 2012, pp. 613–628.

[51] M. Acher, P. Collet, P. Lahire, and R. B. France, "Familiar: A domain-specific language for large scale management of feature models," *Science of Computer Programming*, vol. 78, no. 6, pp. 657–681, 2013.

[52] ——, "Composing feature models," in *International Conference on Software Language Engineering*, ser. SLE. Springer, 2010, pp. 62–81.

[53] D. Benavides, R. Rabiser, D. Batory, and M. Acher, "First international workshop on languages for modelling variability (modevar 2019)," in *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*, 2019, pp. 323–323.

[54] Ø. Haugen, A. Wąsowski, and K. Czarnecki, "Cvl: common variability language," in *Proceedings of the 16th International Software Product Line Conference-Volume 2*, 2012, pp. 266–267.

[55] CVL Submission Team, "Common variability language (CVL), OMG revised submission," 2012, available at https://doi.org/10.1109/SPLC.2008.25.

[56] O. Haugen and O. Ogard, "BVR – better variability results," in *International Conference on System Analysis and Modeling*. Springer, pp. 1–15.

[57] J. Krüger, W. Mahmood, and T. Berger, "Promote-pl: a round-trip engineering process model for adopting and evolving product lines," in *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*, 2020, pp. 1–12.

[58] K. Czarnecki, "Overview of generative software development," in *International workshop on unconventional programming paradigms*. Springer, 2004, pp. 326–341.

[59] K. C. Kang, J. Lee, and P. Donohoe, "Feature-oriented product line engineering," *IEEE software*, vol. 19, no. 4, pp. 58–65, 2002.

[60] L. M. Northrop, "Sei's software product line tenets," *IEEE software*, vol. 19, no. 4, pp. 32–40, 2002.

[61] D. Nesic, J. Krueger, S. Stanciulescu, and T. Berger, "Principles of feature modeling," in *FSE*, 2019.

[62] K. Lee, K. C. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering," in *International Conference on Software Reuse*, ser. ICSR, 2002.

[63] "FM-PRO Technical Documentation," Chair of Software Engineering, Ruhr University Bochum, Tech. Rep., 2024. [Online]. Available: http://se.rub.de/fmpro

[64] "Online Appendix," https://github.com/isselab/2024-fmpro-appendix.

[65] C. Krueger, "Eliminating the adoption barrier," *IEEE Software*, vol. 19, no. 4, pp. 29–31, 2002.

[66] L. Passos, L. Teixeira, N. Dintzner, S. Apel, A. Wąsowski, K. Czarnecki, P. Borba, and J. Guo, "Coevolution of variability models and related software artifacts," *Empirical Software Engineering*, vol. 21, no. 4, pp. 1744–1793, 2016.

[67] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski, "A survey of variability modeling in industrial practice," in *Proceedings of the 7th International Workshop on Variability Modelling of Software-intensive Systems*, 2013, pp. 1–8.

[68] H. P. Jepsen, J. G. Dall, and D. Beuche, "Minimally invasive migration to software product lines," in *11th International Software Product Line Conference (SPLC 2007)*. IEEE, 2007, pp. 203–211.

[69] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Bottom-up technologies for reuse: automated extractive adoption of software product lines," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 67–70.

[70] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "The ecco tool: Extraction and composition for clone-and-own," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 665–668.

[71] S. Grüner, A. Burger, T. Kantonen, and J. Rückert, "Incremental migration to software product line engineering," in *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*, 2020.

[72] W. Mahmood, D. Strueber, T. Berger, R. Laemmel, and M. Mukelabai, "Seamless variability management with the virtual platform," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1658–1670.

[73] S. Apel and C. Kästner, "An overview of feature-oriented software development." *J. Object Technol.*, vol. 8, no. 5, pp. 49–84, 2009.

[74] I. Jacobson, M. Griss, and P. Jonsson, *Software reuse: architecture, process and organization for business success*. ACM Press/Addison-Wesley Publishing Co., 1997.

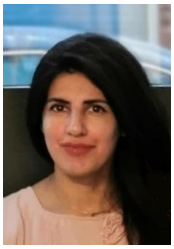
[75] J. Savolainen and J. Kuusela, "Violatility analysis framework for product lines," in *Proceedings of the 2001 symposium on Software reusability: putting software reuse in context*, 2001, pp. 133–141.

[76] J. Krüger, T. Berger, and T. Leich, *Features and How to Find Them: A Survey of Manual Feature Location*. Taylor & Francis Group, LLC/CRC Press, 2018.

[77] L. Passos, J. Padilla, T. Berger, S. Apel, K. Czarnecki, and M. T. Valente, "Feature scattering in the large: A longitudinal study of linux kernel device drivers," in *14th International Conference on Modularity (MODULARITY)*, 2015.

[78] F. Anwer, S. Aftab, U. Waheed, and S. S. Muhammad, "Agile software development models TDD, FDD, DSDM, and Crystal methods: A survey," *International journal of multidisciplinary sciences and engineering*, vol. 8, no. 2, pp. 1–10, 2017.
[79] A. Firdaus, I. Ghani, and S. R. Jeong, "Secure feature driven development (sfdd) model for secure software development," *Procedia-Social and Behavioral Sciences*, vol. 129, pp. 546–553, 2014.
[80] C. Budoya, M. Kissaka, and J. Mtebe, "Instructional design enabled agile method using addie model and feature driven development method," *International Journal of Education and Development using ICT*, vol. 15, no. 1, 2019.
[81] C. Larman, *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum*. Pearson Education India, 2008.
[82] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000.
[83] T. Berger, D. Nair, R. Rublack, J. M. Atlee, K. Czarnecki, and A. Wąsowski, "Three cases of feature-based variability modeling in industry," in *Model-Driven Engineering Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, September 28–October 3, 2014. Proceedings 17*. Springer, 2014, pp. 302–319.
[84] L. Passos, M. Novakovic, Y. Xiong, T. Berger, K. Czarnecki, and A. Wasowski, "A study of non-boolean constraints in a variability model of an embedded operating system," in *Third Workshop on Feature-Oriented Software Development (FOSD)*, 2011.
[85] H. Eichelberger and K. Schmid, "Mapping the design-space of textual variability modeling languages: a refined analysis," *International Journal on Software Tools for Technology Transfer*, vol. 17, pp. 559–584, 2015.
[86] P. Juodisius, A. Sarkar, R. R. Mukkamala, M. Antkiewicz, K. Czarnecki, and A. Wąsowski, "Clafer: Lightweight modeling of structure, behaviour, and variability," vol. 3, no. 1, p. 2.
[87] J. Martinson, H. Jansson, M. Mukelabai, T. Berger, A. Bergel, and T. Ho-Quang, "Hans: Ide-based editing support for embedded feature annotations," in *Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume B*, 2021, pp. 28–31.
[88] A. Hevner and S. Chatterjee, *Design Science Research in Information Systems*. Springer US, 2010.
[89] I. d. C. Machado, A. R. Santos, Y. a. C. Cavalcanti, E. G. Trzan, M. M. a. de Souza, and E. S. de Almeida, "Low-level variability support for web-based software product lines," in *VaMoS*, 2014.
[90] T. Berger, R.-H. Pfeiffer, R. Tartler, S. Dienst, K. Czarnecki, A. Wąsowski, and S. She, "Variability mechanisms in software ecosystems," *Information and Software Technology*, vol. 56, no. 11, pp. 1520–1535, 2014.
[91] L. Passos, J. Guo, L. Teixeira, K. Czarnecki, A. Wąsowski, and P. Borba, "Coevolution of variability models and related artifacts: a case study from the linux kernel," in *SPLC*, 2013.
[92] A. R. Santos, I. do Carmo Machado, and E. S. de Almeida, "Riple-hc: Javascript systems meets spl composition," in *Proceedings of the 20th International Systems and Software Product Line Conference*, ser. SPLC, 2016.
[93] C. Seidl, F. Heidenreich, and U. Aßmann, "Co-evolution of models and feature mapping in software product lines," in *Proceedings of the 16th International Software Product Line Conference-Volume 1*, 2012, pp. 76–85.
[94] K. Feichtinger, D. Hinterreiter, L. Linsbauer, H. Prähofer, and P. Grünbacher, "Supporting feature model evolution by suggesting constraints from code-level dependency analyses," in *Proceedings of the 18th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, 2019, pp. 129–142.
[95] D. E. Khelladi, B. Combemale, M. Acher, and O. Barais, "On the power of abstraction: a model-driven co-evolution approach of software code," in *Proceedings of the ACM/IEEE 42nd International conference on software engineering: new ideas and emerging results*, 2020, pp. 85–88.

Johan Martinson is a software engineer at a logistics company in Sweden. He is also a doctoral student in software engineering at Ruhr University Bochum. In 2019, he received his bachelors in computer engineering and in 2022 his masters in software engineering at Chalmers University of Technology. His research focuses on automating the complex process of featurization in software systems using large language models to introduce and manage variability, making code adaptable to diverse user requirements. In his previous studies he researched and developed projects in tool support for feature traceability in source code, which was published as tool papers in SPLC'24 and SPLC'21.



Wardah Mahmood is a doctoral student at Chalmers | University of Gothenburg, Sweden. Her research interests are software product-line engineering, model-based engineering, formal methods, and empirical software engineering. She received her Master's degree from Fast National University of Computer and Emerging Sciences, Pakistan in 2016.

Jude Gyimah is a junior doctoral student in software engineering at Ruhr University Bochum. He received his bachelors in information technology at the Ghana Technology University College (2016) and his masters in software engineering and management at the University of Gothenburg (2021). In his previous studies he worked on software systems and tools for software engineering automation. He currently applies feature-oriented software development on projects relating to Domain specific languages, Configurable Systems Engineering and Adaptive Systems Design, in the context of service robots, autonomous systems and system softwares.

Thorsten Berger is a Professor in Computer Science at Ruhr University Bochum in Germany. After receiving the PhD degree from the University of Leipzig in Germany in 2013, he was a Postdoctoral Fellow at the University of Waterloo in Canada and the IT University of Copenhagen in Denmark, and then an Associate Professor jointly at Chalmers University of Technology and the University of Gothenburg in Sweden. He received competitive grants from the Swedish Research Council, the Wallenberg Autonomous Systems Program, Vinnova Sweden (EU ITEA), and the European Union. He is a fellow of the Wallenberg Academy—one of the highest recognitions for researchers in Sweden. He received two *best-paper* and two *most-influential-paper* awards. His service was recognized with distinguished reviewer awards at the tier-one conferences ASE 2018 and ICSE 2020, and at SPLC 2022. His research focuses on software product line engineering, AI engineering, model-driven engineering, and software security. He is co-author of the textbook on Domain-Specific Languages: Effective Modeling, Automation, and Reuse.