Empirical Software Engineering (2022) 27: 72
https://doi.org/10.1007/s10664-021-10112-3

Effects of variability in models: a family of experiments

Wardah Mahmood[1] · Daniel Strüber[2] · Anthony Anjorin[3] · Thorsten Berger[1,4]

Accepted: 24 November 2021

Abstract

The ever-growing need for customization creates a need to maintain software systems in many different variants. To avoid having to maintain different copies of the same model, developers of modeling languages and tools have recently started to provide implementation techniques for such variant-rich systems, notably *variability mechanisms*, which support implementing the differences between model variants. Available mechanisms either follow the *annotative* or the *compositional* paradigm, each of which have dedicated benefits and drawbacks. Currently, language and tool designers select the used variability mechanism often solely based on intuition. A better empirical understanding of the comprehension of variability mechanisms would help them in improving support for effective modeling. In this article, we present an empirical assessment of annotative and compositional variability mechanisms for three popular types of models. We report and discuss findings from a family of three experiments with 164 participants in total, in which we studied the impact of different variability mechanisms during model comprehension tasks. We experimented with three model types commonly found in modeling languages: class diagrams, state machine diagrams, and activity diagrams. We find that, in two out of three experiments, annotative technique lead to better developer performance. Use of the compositional mechanism correlated with impaired performance. For all three considered tasks, the annotative mechanism was preferred over the compositional one in all experiments. We present actionable recommendations concerning support of flexible, tasks-specific solutions, and the transfer of established best practices from the code domain to models.

Keywords Variability mechanisms · Model-driven engineering · Software product line engineering · Empirical study

Communicated by: Philippe Collet, Sarah Nadi, Christoph Seidl, and Leopoldo Motta Teixeira

This article belongs to the Topical Collection: *Software Product Lines and Variability-rich Systems (SPLC)*

✉ Wardah Mahmood
wardah@chalmers.se

Extended author information available on the last page of the article.

1 Introduction

Variant-rich systems can offer companies major strategic advantages, such as the ability to deliver tailor-made software products to their customers. Still, when developing a variant-rich system, severe challenges may arise during maintenance, evolution, and analysis, especially when variants are developed in the naive *clone-and-own* approach, that is, by copying and modifying them (Pohl et al. 2005). The typical solution to these challenges is to manage variability by using dedicated *variability representations*, capturing the differences between the variants (van der Linden et al. 2007). An important type of variability representation are *variability mechanisms*, which are used to avoid duplication and to promote reuse when implementing variability in assets such as code, models, and requirements documents. Over more than three decades, researchers have developed a plethora of variability mechanisms, albeit mostly for source code (Apel et al. 2013; Berger et al. 2014a; Van Gurp et al. 2001).

As companies begin to streamline their development workflows for building variant-rich systems, they recognize a need for variability management in all key development artifacts, including models. The use of models is manifold, ranging from sketches of the system design, to system blueprints used for verification and code generation. The car industry is particularly outspoken on their need for model-level variability mechanisms (Famelis et al. 2015). For example, General Motors named support for variation in UML models as a major requirement (Flores et al. 2012), and Volkswagen reported large numbers of complex, cloned variants of Simulink models in their projects (Schlie et al. 2017). Beyond automotive, the need for model-level variability has been documented for power electronics, aerospace, railway technology, traffic control, imaging, and chip modeling (Berger et al. 2020).

Recognizing this need, researchers have started building variability mechanisms for models. Variability mechanisms are now available both for UML (Apel et al. 2009; Czarnecki and Antkiewicz 2005; Schwägerl et al. 2015) and Domain-Specific Modeling Languages (DSMLs[1] (Alférez et al. 2019; Anjorin et al. 2014; Heidenreich et al. 2008; Mussbacher et al. 2012; Strüber and Schulz 2016; Tolvanen and Kelly 2011; 2019a; White et al. 2009; Zschaler et al. 2009)). Building on these results, researchers have started to address advanced problems such as the migration of a set of "cloned-and-owned" model variants to a given mechanism (Assunção et al. 2017; Martinez et al. 2015; Rubin and Chechik 2013; Schlie et al. 2017; Wille et al. 2017), and efficient analysis of large sets of model variants (Classen et al. 2010; Czarnecki and Pietroszek 2006; Peldszus et al. 2018). Adoption in several industrial DSMLs has demonstrated the general feasibility of model-level variability mechanisms in practice (Tolvanen and Kelly 2019b).

While variability mechanisms for source code are reasonably well understood (Berger et al. 2014a; Melo et al. 2016; Van Gurp et al. 2001), language and tool designers are offered little guidance on selecting the most effective variability mechanism for their purposes. In fact, there is a lack of evidence to support the preference of one mechanism over the other. In line with previous studies on code-level mechanisms (Feigenspan et al. 2013; Fenske et al. 2017; Krüger et al. 2019; Melo et al. 2016), we argue that *comprehensibility* is a decisive factor for the efficiency of a variability mechanism—for any maintenance and evolution activity (e.g. bugfixing, feature implementations), the developers first need to understand the existing system. A better empirical understanding of the comprehension of variability mechanisms could support the development of more effective modeling languages and tools.

[1]DSMLs allow modeling software systems from different domains using domain-specific notations.

To this end, we present an empirical study of variability representations in models. We report on a family of three experiments in which we studied how the choice of variability mechanism affects performance during model comprehension tasks. We consider comprehension tasks for three popular model types[2]: class diagrams, state machine diagrams, and activity diagrams. The experiments are fully randomized, and employ student developers from three countries. We consider two selected variability mechanisms that are representative for the two main types distinguished in the literature (Kästner et al. 2011): *Annotative* mechanisms maintain an integrated, annotated representation of all variants. Examples include preprocessor macros (Spencer and Collyer 1992) (for code) and model templates (Czarnecki and Antkiewicz 2005) (for models). Annotative mechanisms are conceptually simple, but can impair understandability, since they clutter model or code elements with variability information (Melo et al. 2016; Spencer and Collyer 1992). *Compositional* mechanisms allow to compose a set of smaller sub-models to form a larger model. Examples include *feature-oriented programming* (Apel and Kästner 2009) (for code) and *model refinement* (Faitelson and Tyszberowicz 2017) (for models). Compositional mechanisms are appealing, as they establish a clear separation of concerns, but they involve a composition step which might be cognitively challenging. We aimed to shed light on the impact of these inherent trade-offs.

We focus on three model types used in various modeling languages: class diagrams, state machine diagrams, and activity diagrams. The diagrams are three commonly used types of UML models, popular both in academia and industry. *Class diagrams* play a significant role in domain and system analysis and design. They are representative for a wide array of visual languages modeling domain concepts, such as Entity-Relationship diagrams (ER diagrams, Chen (1976)), and they can be used for generating the architecture of a system (Assunção et al. 2020). *State machine diagrams* model system behavior in terms of the different states a system exists in. They play an important role in software verification (Ali et al. 2007). *Activity diagrams* also model behavior, but in contrast to state machine diagrams, they model the interaction between the user and the system. Both state machine diagrams and activity diagrams are representative of other behavioral representations such as sequence diagrams (which model a system in terms of sequential interactions between actors), and can also be used for code generation and system verification (Gessenharter and Rauscher 2011).

We make the following contributions:

- We present our findings on a family of experiments, each investigating how the choice of variability mechanism affects the comprehensibility of different model-related tasks for three popular model types.
- We present a quantitative analysis of correctness, the completion time, and subjective assessments of our participants for six model comprehension tasks.
- We present a qualitative analysis of participant responses, adding rationale to explain the observed results.
- Based on our synthesized findings, we propose recommendations for language and tools developers.
- We provide a replication package (Mahmood et al. 2021) that includes our experimental material, anonymized responses, and analysis scripts.

[2]In this paper, we use the terms *diagram* and *model* interchangeably. In modeling languages such as UML, models can consist of a single diagram. The difference between such a model and the contained diagram is then not essential.

This paper considerably extends our earlier conference paper (Strüber et al. 2020) that presented the first of our three experiments, focusing on class diagrams (73 participants). Based on these earlier results, we designed, performed, and analyzed two follow-up experiments, keeping the methodological setup of the first experiment, while varying the considered modeling languages. The results add nuance and additional insights, while largely confirming the findings from the first experiment. Based on the results, we are able to derive conclusion about a broader class of modeling languages and based on more observations (from 164 participants).

We present the first empirical study of variability mechanisms for models that investigates the effect of different mechanisms in a controlled experiment. While earlier empirical studies considered the comprehensibility of code-level variability mechanisms (see Section 8), their generalizability to models is unclear. Code usually has a tree-like structure and is expressed in textual notations. Modeling languages support the structuring of models in a graph-like manner and usually have graphical notations. Since different representations are known to affect performance during decision-making tasks (Vessey 1991), specifically, software engineering tasks (Acreţoaie et al. 2018; Labunets et al. 2017), we argue that the comprehensibility of model variability mechanisms requires a dedicated investigation. In the scope of models, related work is on experience reports in variability modeling (e.g., Alférez et al. 2019; Berger et al. 2014b; García et al. 2019) and controlled experiments outside the scope of variability (e.g., Acreţoaie et al. 2018; Nugroho 2009; Ramadan et al. 2020).

2 Background

There has been a recent surge of interest in dedicated variability mechanisms for models. Lifting the related distinction from code-level mechanisms, two main types are distinguished: Annotative mechanisms represent variability with an annotated integrated representation of all variants. Mechanisms in this category are *model templates* (Czarnecki and Antkiewicz 2005; Heidenreich et al. 2008; Taentzer et al. 2017), *union models* (Alwidian and Amyot 2019), and the *top-down approach* (Kühn and Cazzola 2016). Compositional mechanisms represent variability by composing variants from smaller sub-models (from here referred to as *model fragments*). Available approaches mostly differ in their model fragment syntax and composition semantics. Examples are *delta modeling* (Clarke et al. 2010), *model superimposition* (Apel et al. 2009), *refinement* (Faitelson and Tyszberowicz 2017; White et al. 2009), *components* (Zschaler et al. 2009), and the *bottom-up approach* (Kühn and Cazzola 2016).

To illustrate the role of both types of mechanisms in industry, we refer to a recent survey of variability support in 23 DSMLs (Tolvanen and Kelly 2019b). It describes four strategies being used: First, a model represents one variant (9 languages); second, elements are reused across models by referencing (10 languages); third, multi-level modeling is used for capturing variability (1 language); fourth, elements have so-called presence conditions (explained shortly, 3 languages). The first strategy is considered as a baseline in our experiments. The second and third one are compositional, as they spread differences between variants across several smaller models. The fourth one is annotative.

We selected the two variability mechanisms for our experiments based on the following criteria: (M1) *The mechanism has a graphical syntax.* (M2) *The mechanism is supported by available tools.* (M3) *The mechanism has been described in the scientific literature.* The rationale for M1 was to study variability mechanisms in the widespread graphical representation of our considered model types. Support by tools (M2) and available literature (M3) may contribute to the transfer of existing research results to industrial practice, and allow practitioners to test the mechanisms in available prototypes.

Based on these criteria, as an annotative mechanism, we identified *model templates* (implemented in FeatureMapper (Heidenreich et al. 2008), SuperMod (Schwägerl et al. 2015), and Henshin (Strüber and Schulz 2016)). For compositional, we identified two existing approaches fulfilling the criteria: Delta modeling (implemented by DeltaEcore (Seidl et al. 2014) and SiPL (Pietsch et al. 2015)) and model refinements (implemented by eMoflon (Anjorin et al. 2014)). We decided to consider model refinements, as they implement the compositional paradigm in the most straightforward way (delta modeling supports deletions, which increases its expressiveness, but requires more complex syntax and semantics).

Example. We illustrate the specific variability mechanisms used in our experiments with a simple example, inspired by Schaefer (2010). The same example was also used in the experiment to introduce the variability mechanisms to the participants.

The example represents a simple cash desk system that exists in three similar, but different variants. Figure 1 depicts the individual variants using separate class diagrams: Variant `var1` consists of a `CashDesk` with a `KeyBoard` and a `Display`. Variant `var2` has additionally exactly one `CardReader` connected to the `CashDesk`. Variant `var3` replaces the `Keyboard` with a `Scanner` and makes the `CardReader` optional (multiplicity `0..1` instead of `1`).

The depicted representation of listing variants individually is used as a baseline in the first of our experiments, referred to as the "enumerative mechanism." This solution is frequently applied in practice (Tolvanen and Kelly 2019b), where it leads to severe maintenance drawbacks. For example, a bug found in one of the variants must be fixed in all variants separately. The goal of the variability mechanisms presented below is to simplify working with such similar, but distinct variants.

Annotative Variability. The annotative mechanism considered in our experiments is *model templates* (Czarnecki and Antkiewicz 2005). Like annotative mechanisms in general, it combines all model variants into a single model with annotations. The left-hand side of Fig. 2 shows a model template for our example: a class diagram that represents the three variants of the cash desk system. Parts of the class diagram are annotated with *presence conditions*, stating the variants in which the part occurs. For brevity, we define a presence condition as a list of configuration options (disjunction). For example, the presence condition `<<var1,var2>>` indicates that the annotated part is present when either the configuration

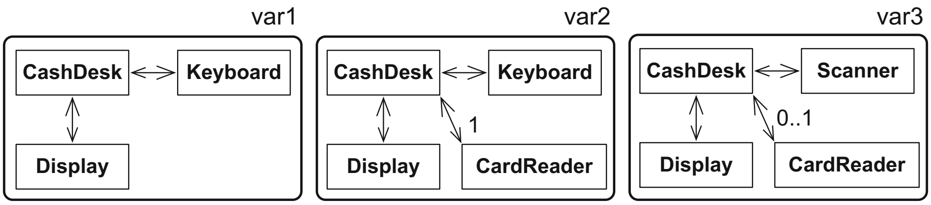

Fig. 1 Three variants of a cash desk system

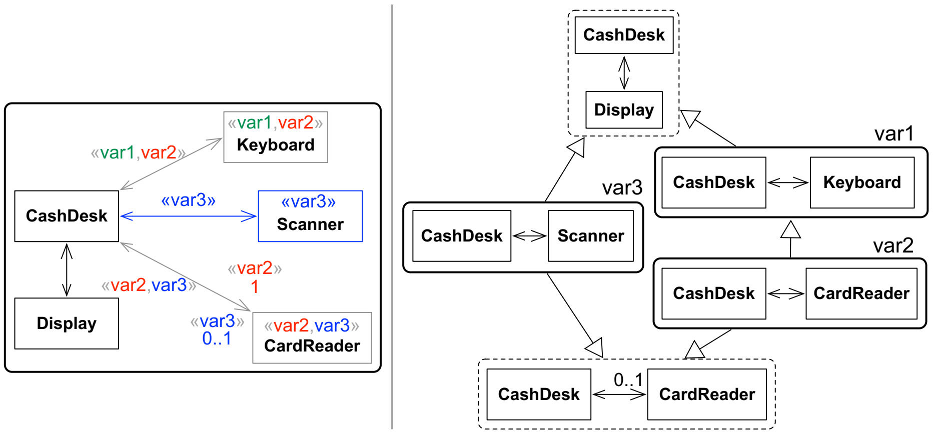

Fig. 2 Annotative and compositional variability

option `var1` or `var2` is selected. The absence of a presence condition denotes that the part is contained in all variants.

Colors are used in the following way: Elements (classes and associations) with a black outline occur in all variants, elements with a grey outline occur in two or more variants, elements with a colored outline belong to precisely one variant, whose annotation is also depicted with the same color. The use of colors to distinguish elements goes back to the original paper that introduced model templates (Czarnecki and Antkiewicz 2005). Colors may be crucial for comprehensibility. In the case of code-level variability mechanisms, Feigenspan et al. (2013) found that colors support understanding of annotative variability. We are interested to determine if this finding also applies to models.

Individual variants are derived from the combined model as follows: The user sets one of the configuration options as active. The concrete model is derived by removing all those elements whose presence condition does not contain the configuration option. For example, selecting the configuration option `var1` leads to the model variant `var1` in Fig. 1.

Compositional Variability. The compositional mechanism we considered is *refinement* (Anjorin et al. 2014). Like all compositional mechanisms, refinement provides (i) a means of decomposing variants into smaller building blocks, and (ii) a means of *merging* building blocks to form complete variants. This allows for sharing and reuse of common parts in different variants. The building blocks are visually shown as a network, as depicted in the right-hand side of Fig. 2. Commonalities of `var1` and `var3`, as well as `var2` and `var3` have been extracted into separate "super" class diagrams. These diagrams have a dashed border, as they only represent commonalities and are "abstract" in the sense that they are not complete variants. Composition of diagrams is denoted using an inheritance arrow, e.g., `var2` is formed by combining `var1`, the elements specified in `var2`, and the elements in the common super class of `var3` and `var2`. As the example demonstrates, multiple super class diagrams (see `var3`) and transitive composition (see `var2`) are possible.

Deriving individual variants is a two-step process. First, a *union* of the contents of the variant and all its transitive parents is computed; this results in a single, flat class diagram (with no parents). Second, a *merge* operator is used to combine elements that should be the same. For class diagrams, this operator combines all elements with the same name. The merge operator also defines how to resolve conflicts: for class diagrams, a common subtype must exist for nodes to be merged, and multiplicities of merged edges are combined by taking the maximum of lower bounds and minimum of upper bounds. For example, when the variant `var1` is selected, it is merged with its parent (top class diagram with dashed lines). Building the union of both class diagrams and merging the cash desk elements leads to the model variant `var1` in Fig. 1.

Considered Model Types. In our experiment family, we cover three types of models: class diagrams, state machine diagrams, and activity diagrams. Our rationale is three-fold: These model types: (i) are commonly taught in undergraduate education, allowing our participants to work with languages they are already familiar with; (ii) are widely used in industry (Langer et al. 2014; Petre 2013), indicating their representativeness, (iii) together are suited to capture three essentially different concerns: static structure, dynamic behavior from the internal system perspective, and dynamic behavior in interaction with the user.

In each of these model types, variability can be implemented using the annotative and compositional mechanisms illustrated in the example above. We now discuss each model type and any necessary customizations required to accommodate the model type in our considered mechanisms.

Class diagrams have a pronounced role in system design and analysis. In code generation contexts, they are used to generate data management components (e.g., large parts of enterprise web and mobile apps can be generated from class models (Kulkarni et al. 2002; Moreno et al. 2007; Vaupel et al. 2014)), object-oriented code in roundtrip engineering scenarios (Bruneliere et al. 2010), and ample Model-Driven Engineering (MDE) tooling in modeling platforms, such as the Eclipse Modeling Framework (EMF (Steinberg et al. 2008)). Class models can be supported via model templates and model refinement in a straightforward day, as shown in the example above.

State machine diagrams capture the dynamic behavior of a system, focusing on one of its entities or objects, in terms of its possible states and the transitions between the states, based on certain well-defined events. Like class diagrams, they play an important role in code generation (Chauvel and Jézéquel 2005; Domı et al. 2012). Expressing variability in state machine diagrams using model templates is straightforward, based on assigning presence conditions to states and transitions. Model refinement can be applied in a similar way as for class diagrams. One additional complication arises with hierarchical state machines, where merging must respect the nesting of states as defined in different model fragments. In principle, multiple model fragments can make conflicting contributions to the merge result (for example, if there is a fragment with state A nesting a substate B, and another fragment with a state B nesting a substate A). We designed our examples to avoid such situations.

Activity diagrams, like state machine diagrams, also capture the dynamic behavior of a software system. However, in contrast to state machine diagrams, activity diagrams provide means to model *user* and *user-visible* activities, as well as the flow between them. Activity diagrams allow modeling overlapping activities (*fork*), or activities that need coordination (*merge*). Expressing variability in activity diagrams using model templates is straightforward (Reuys et al. 2006), based on assigning presence conditions to activities. Model refinement can be applied in a similar way as before. One complication concerns the merging of flow: if the same activity appears in different fragments with different subsequent activities, one needs to define how the arising conflict is resolved. For our experiments, we defined a conflict-resolution rule that an actual activity always overrides an ending node. Figure 3 depicts an example: the left fragment consists of the functionality of *withdraw* and

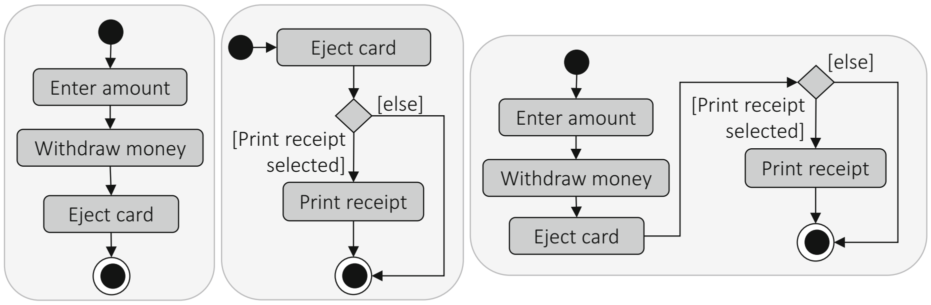

Fig. 3 Model refinement rule for activity diagrams: ending activities are overwritten by decision nodes

the middle fragment composes the activities for *printing receipts*. When merging both fragments, the final activity in the left fragment is overwritten by the decision node (represented as a diamond) in the middle fragment. The right fragment shows the merged form of both fragments.

3 Overview on Our Family of Experiments

We performed a family of three experiments, illustrated in the high-level overview in Fig. 4. Our experiment family consisted of two independent variables: the considered model type and variability mechanism. The former was varied *between* experiments, i.e., each experiment focused on a single model type. The latter was varied *within* each experiment, i.e., each experiment compared multiple variability mechanisms on the same model type.

We considered three variability mechanisms—*annotative*, *compositional*, and *enumerative*—where the enumerative mechanism (a simple listing of all variants) was considered the baseline. In all three experiments, we adopted a within-subject design, where each participant used each considered variability mechanism on all tasks. We considered three widely used models types—class diagrams, state machine diagrams, and activity diagrams—as discussed and motivated in Section 2.

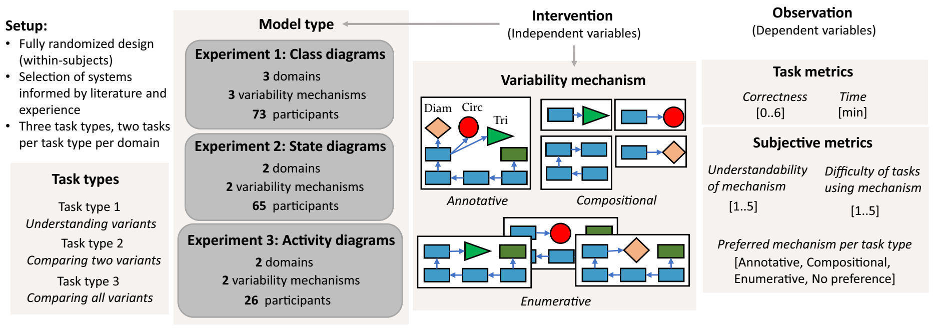

Fig. 4 Methodology overview

In Experiment 1, we considered all three variability mechanisms. Based on some important observations from the experiment 1 (explained shortly), we decided to not consider the enumerative mechanism in the remaining experiments. This allowed us to have a more in-depth comparison and reflection of annotative and compositional variability in experiments 2 and 3, based on more intricate tasks.

In every experiment, our participants worked with example models from certain *domains*, derived from the literature and our experiences. To avoid learning effects based on answers of previously completed tasks, we varied the considered example domain for each variability mechanism. Consequently, the number of considered domains in each experiment matched the number of variability mechanisms: three in Experiment 1, two in Experiment 2 and Experiment 3.

Prior to the experiments, as a preparatory study, we conducted a further experiment (explained shortly in Section 4) to shape the design of our materials and tasks. The goal was to assess the suitability of our experimental tasks and to derive potential improvements of the setup.

To recruit a significant number of participants, we involved students as participants, due to their representativeness as stand-ins for practitioners (Salman et al. 2015). The participants came from four different universities in Germany (two universities), Netherlands, and Sweden. We shortly discuss demographic aspects of our participants. In each experiment, we randomly divided participants into n groups (n = number of variability mechanisms). For subject allocation, we used a *Latin square design* (Melo et al. 2016; Montgomery 2017) to ensure that each participant used every distinct variability mechanism and domain *exactly* once, as to avoid learning effects.

To ensure homogeneity, we kept other aspects constant across our experiments as far as possible: training material, goals and research questions, experimental design, task types, task metrics, subjective assessments, analysis, and participant selection. We used different domains in each experiment, deliberately to make the results generalizable. We customized the individual tasks according to the domains, keeping the task types unchanged. To analyze comprehensibility, we designed three task types: Understanding variants, comparing two variants, and comparing all variants. Our detailed methodology and analysis is described in Section 5 and 6, respectively.

The considered domains were varied between the experiments: In Experiment 1 and Experiment 3, we chose intuitively understandable examples that inherently lend themselves towards being expressed with the considered model type. To this end, we derived the domains by taking inspiration from literature. The domain choice in Experiment 3 also incorporated participant feedback from previous iterations of the experiment. In Experiment 2, we derived two sub-domains from a software project considered in the course that the participants were recruited from. This was useful because it allowed us to conclude that all participants were familiar with the considered domains.

4 Preparatory Study

We conducted a preparatory study to evaluate our experimental design and identify possible issues and other amendable aspects. The study was performed on a population of 28 students (disjoint from the population of our experiment). The students were familiar with class models, the model type used in the experiment.

The tasks considered were *bug-finding tasks*, a typical task type for assessing the usefulness of visual representations (Melo et al. 2016; Strüber et al. 2019). Participants were handed a textual requirement specification, together with design models implementing the requirements with one of the given variability mechanisms. The design models contained a number of deviations from the textual requirements (bugs), which the participants were asked to identify. We also asked the participants to suggest potential improvements to the experiment using an open-ended question.

To provide meaningful example domains, we considered the existing literature. The first example represented a phone product line with phones being conditionally capable of making incoming and outgoing calls (Apel et al. 2009). The second example represented a project management system with managers, employees, and tasks (Fahrenberg et al. 2014). Students obtained a virtual instruction sheet and a link to an explanation video for the used variability mechanisms. The students were asked to complete the entire questionnaire in 30 min.

From this preparatory study we drew three main conclusions: First, example models with 3 to 4 classes, and 3 or 4 variants each were too simple to demonstrate a difference between both mechanisms. This conjuncture was supported by one participant's written recommendation to *"create [more] complicated examples with 6 or 7 classes and not so easy ones."*

Second, despite our efforts to provide clear requirements, a participant asked us to be "more specific and less ambiguous with the requirement specifications." Ambiguity is an inherent risk to experimental validity since its effect is hard to quantify (it is unclear how many participants assume a different understanding than intended). Another recurrent comment was that reading the descriptions was tiring, threatening the completion rate. Therefore, we decided to switch the nature of the used tasks in the main experiment to comprehension tasks that do not rely on additional artifacts.

Third, the provided instruction video was viewed as redundant, as it showed only the information that was available on the instruction sheet. In the actual experiments, we decided to omit the instruction video.

5 Methodology

In this section, we present the detailed experimental methodology for our family of three experiments, each of which is focusing on one model type. Our experimental materials and data, including the raw data, are publicly available via our replication package (Mahmood et al. 2021).

5.1 Experimental Setup

As explained in Section 3, to ensure uniformity, we reused some components of our experiment in all three executions: goals and research questions, training material, task types, task metrics, subjective assessment, quantitative feedback, and data analysis. In this section, we elaborate these common components of our experiments.

Research Questions. Our goal was to study the effect of variability mechanisms on model comprehension. Towards this goal, we formulated and investigated the following research questions:

RQ1 *To what extent do variability mechanisms impact the efficiency of model comprehension?*

We studied the effect of annotative and compositional mechanisms on the ability to solve model comprehension tasks correctly and quickly.

RQ2 *How are variability mechanisms perceived during comprehension tasks?*

We studied the perceived understandability and difficulty to complete model comprehension tasks depending on the used variability compositional mechanism, based on subjective assessments.

RQ3 *How are participant preferences for variability mechanisms distributed over different task types?*

We elicited qualitative and quantitative data about the participants' subjective preferences by asking them to choose a preferred mechanism and explain the choice.

Experimental Design. We applied a *cross-over trial*, a variant of the within-subject design (Jones and Kenward 2003), in which all participants are sequentially exposed to each treatment. The treatments in our case were the use of the different variability mechanisms during comprehension tasks. The main benefit of the chosen design is its efficiency in enhancing statistically valid conclusions for a given number of participants. The design also reduces the influence of confounding factors, such as participant expertise, because each participant serves as their own control.

A main threat to this kind of study design are learning effects: during the experiments, participants might transfer experience gained by solving one task to other tasks. We mitigated this threat by using a *Latin square design* (Melo et al. 2016; Montgomery 2017). Participants were randomly distributed across equally sized groups, such that each participant experimented with each variability mechanism and each domain once. Each group was assigned one of several *paths* through the experiment, based on the different possible combinations of domains and variability mechanisms. For example, consider the three paths for Experiment 1, which included the variability mechanisms annotative (*Ann*), compositional (*Com*), and enumerative (*Enu*), and three domains *d1*, *d2*, and *d3*:

- *Enu d1*→*Ann d2*→*Com d3* (path 1),
- *Com d1*→*Enu d2*→*Ann d3* (path 2), and
- *Ann d1*→*Com d2*→*Enu d3* (path 3).

Following our Latin square design, to avoid bias related to the complexity of the considered domains, the order of domains was fixed between paths.

Figure 5 shows the design and flow of our questionnaire. Each module corresponded to one element of the above-mentioned paths, and consisted of one model, its description, and six tasks. We discuss further threats and mitigation strategies in Section 7.

Training Material. The participants received training material in the form of handouts elaborating the variability mechanisms before beginning the experiments (available in replication package (Mahmood et al. 2021), *intro* documents in folder *material*). In the training material, we elaborated the variability mechanisms and presented different representations of the illustrative example shown in Section 2. We also showed how variants can be derived by giving valid feature selections (annotative) or composing different sub-models (compositional). We reused the same training material across all experiments, with small adjustments.

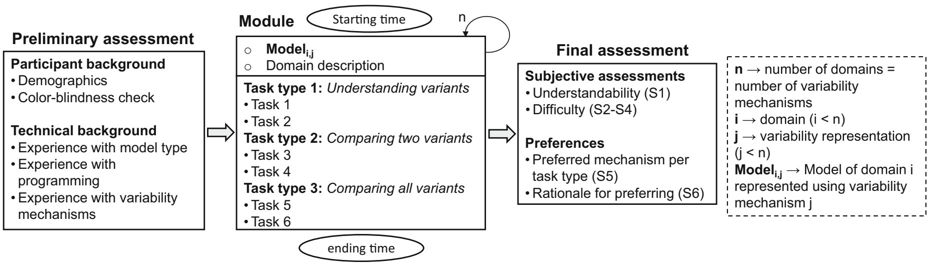
Preliminary assessment
Participant background
• Demographics
• Color-blindness check
Technical background
• Experience with model type
• Experience with programming
• Experience with variability mechanisms
Module
Starting time
n
Model$_{i,j}$
Domain description
Task type 1: Understanding variants
• Task 1
• Task 2
Task type 2: Comparing two variants
• Task 3
• Task 4
Task type 3: Comparing all variants
• Task 5
• Task 6
ending time
Final assessment
Subjective assessments
• Understandability (S1)
• Difficulty (S2-S4)
Preferences
• Preferred mechanism per task type (S5)
• Rationale for preferring (S6)
n → number of domains = number of variability mechanisms
i → domain (i < n)
j → variability representation (j < n)
Model$_{i,j}$ → Model of domain i represented using variability mechanism j

Fig. 5 Questionnaire design for our experiments, with n = 3 for experiment 1, and n = 2 for experiments 2 and 3

Specifically, we extended the material for Experiment 3 (activity diagrams), where the composition required an extra step, i.e., when merging two activity diagrams, decision nodes overwrite ending activities.

Preliminary Assessment. In the questionnaires (available in replication package (Mahmood et al. 2021), folder *questionnaires*), before the actual tasks, we asked the students to self-assess their expertise in three relevant categories using five-point Likert scales: Model type (the baseline technology of our experiment), programming (to argue for the representatives of our findings), and the considered variability mechanisms (the experimental treatment). The five-point scale consisted of positive integers from 1–5, 1 representing the lowest value, and 5 representing the highest. Table 1 shows a summary of participant self-assessment ratings for the three experiments.

Task Types and Tasks. While designing tasks, we aimed for representativeness. We designed task types to depict common activities performed by developers in variant-rich systems. For each domain represented using a given variability mechanism, participants were required to perform tasks of *three* types. Every task type consisted of *two* concrete tasks. Below, we discuss the three task types, provide an argument for their representativeness, and give examples for the concrete tasks per task type.

Task Type 1 ("understanding variants") required participants to map *elements* of the model (classes, states or activities) to variants. This task type is inspired by feature location: a common activity where parts of an artifact (code or model) implementing a feature has to be identified. Tasks of this type followed the style: *"Which variants have the elements X*

Table 1 Technical background of our participants

Exp.	Experience with				
	MType	Prog.	Ann.	Com.	Enu.
1	3.47 ± 0.60	3.62 ± 0.74	1.73 ± 0.87	1.86 ± 1.03	1.87 ± 0.93
2	2.84 ± 0.58	3.42 ± 0.74	2.26 ± 0.73	2.52 ± 0.77	–
3	3.39 ± 0.64	4.0 ± 0.46	2.1 ± 0.8	2.3 ± 0.86	–

Ratings on a 5-point Likert scale. 1: lowest 5: highest.

Exp.: Experiment **MType:** Model type **Prog:** Programming **Ann:** Annotative Variability **Com:** Compositional Variability **Enu:** Enumerative Variability *and Y? List all such variants, or write none otherwise."* Consider the Phone management domain in the first experiment (in the replication package (Mahmood et al. 2021), *material/exp1/Ship.pdf*), we formulated the following task for task type 1: *How many variants have both the classes "Camera" and "Video"?*

Task Type 2 ("comparing two variants") required participants to differentiate two variants in terms of the elements they consist of. Specifically, they needed to list the non-overlapping elements of two given variants. In practice, such a task is performed to deeply understanding how two closely related variants differ. Tasks of this type followed the style: *"How do the two variants Var1 and Var2 differ? List all differing elements if there are any.'* As an example, consider the Robocode domain in the second experiment (see the replication package (Mahmood et al. 2021), *material/exp2/Boat.pdf*), where we formulated the following task for task type 2: *Which state(s) does [the robot variant] "Grizzlyman" have that [variant] "Toxitonic" doesn't have? List all such states if there are more than one.*

Task Type 3 ("comparing all variants") involved a broader comparison, e.g., listing elements belonging to all variants. Such a task type is performed when trying to understand the entire variant space. Tasks of this type resembled the following style: *"Which elements are included in all variants?"* To reduce effort in the case of larger examples, we specified a pre-selection of a obvious elements and asked the participants to fill in the remaining ones. For example, consider the third experiment with activity diagrams as the featured model type (replication package (Mahmood et al. 2021), *material/exp3/Train.pdf*). We formulated the following task for task type 3: *What is the longest possible path of activities you can have in any of the products, from a start to an end activity? Please list all activities that the path consists of (excluding start and end activities) in order. If there are multiple longest paths, pick one.*

Task types in the questionnaire were represented as sub-sections of the modules (Fig. 5), where each sub-section consisted of two tasks. Each task consisted of one concrete question that the participants were asked to answer, as illustrated in the previous examples.

Dependent Variables. For measuring efficiency (RQ1), we elicited two metrics: correctness and completion time. The correctness of a task type was the aggregate correctness of its two concrete tasks. For each task, we evaluated the responses using a scale of 0–1 as follows: Correct responses received a score of 1, partially correct 0.5, and incorrect 0. The responses were evaluated against *oracles* that the authors produced. For each task type per sub-system, the primary and secondary authors iteratively solved each individual task until they reached a consensus (which formed the oracle). In majority of the cases, both the authors had the same responses. As a margin of error, we also checked if the responses to one task were consistently different than our oracles. This was never the case in our experiments.

A response was deemed partially correct if it included *some but not all* correct elements, or *some correct and some incorrect* ones. The scores of both tasks of a task type were summed up to obtain the correctness score in the range 0–2. This resulted in a total of three correctness scores per participant. For completion time, we asked the participants to log the starting and ending times of each module (Fig. 5), which we converted to the completion time in minutes.

To address RQ2, we asked the participants to assess the *understandability* of each mechanism and the *difficulty* of addressing each task type using each mechanism. Specifically, we asked the following questions:

(S1) How easy did you find it to understand each mechanism?

(S2) How difficult was it to answer the questions on “Understanding variants” (tasks 1 and 2) for each mechanism?
(S3) How difficult was it to answer the questions on “Comparing two variants” (tasks 3 and 4) for each mechanism?
(S4) How difficult was it to answer the questions on “Comparing all variants” (tasks 5 and 6) for each mechanism?

Following the common practice for subjective responses, we captured the response on a 5-point Likert scale for each mechanism. The points represented increasing levels of difficulty (from 1 easiest to 5 hardest). We used the same labels for the Likert scales for S1-S4, specifying explicitly for each question that 1 means *very easy*, and 5 means *very difficult*.

We aimed to investigate two hypotheses: that (i) all variability mechanisms are perceived to be equally understandable, and (ii) while performing tasks of each task type, participants experienced equal difficulty using all variability mechanisms. For the former, we conducted a statistical analysis to compare the understandability of different variability mechanisms included in the experiment. For the latter, we conducted dedicated statistical analyses to compare difficulty ratings per variability mechanism given by the participants for each task type.

For RQ3, we asked the participants to specify their *preferred mechanism* per task type. To gain deeper insight into the rationale, and complement the quantitative information with qualitative data, we also asked our participants to elaborate on their choice of preferred mechanism—a setup inspired by mixed-method research (Bergman 2011). We used the following questions:

(S5) Which mechanism do you prefer for each of the three task types?
(S6) Can you explain your subjective preferences (intuitively)?

The answer to S5 was specified by selecting one of the literals Annotative, Compositional, Enumerative, None for each of the task types. To collect the qualitative data in S6, we kept S6 open-ended.

Analysis. For hypothesis testing, we used the Wilcoxon signed-rank test (Wilcoxon 1945) which we applied to the task and subjective metrics, following recommendations according to which this test can in fact be applied to Likert-scale data (De Winter and Dodou 2010). We used the standard significance threshold of 0.05. Two measurements involved multiple comparisons (correctness and difficulty; each for 3 different task types). For these metrics, we applied the Bonferroni correction (Abdi 2007), yielding a corrected significance threshold of 0.017, obtained by dividing 0.05 by 3. We employed the A_{12} score for assessing effect size following Vargha and Delaney’s original three interpretations (Vargha and Delaney 2000): $A_{12} \approx 0.56 =$ *small*; $A_{12} \approx 0.64 =$ *medium*; and $A_{12} \approx 0.71 =$ *large*. All tests were executed with R.

For assessing the qualitative data, one of the authors used *inductive coding* to tag the participants’ comments from one of the experiments with relevant keywords. Afterwards, two other authors verified the tags and suggested improvements in the tags. Based on the discussion and feedback with the two authors, the author tagged the comments for the remaining two experiments. The tags were useful to identify interesting aspects, and their frequency helped to identify the redundant concerns.

Participants. The participants were recruited from undergraduate and graduate courses at various universities. Our rationale for recruiting students is their suitability as stand-ins for practitioners: students can perform involving unfamiliar software engineering tools equally well as practitioners (Salman et al. 2015). The students were recruited from courses with completed previous lectures and homework assignments on models featuring in the experiment. Before the experiment, it was pointed out that participation in the experiment was entirely voluntary, and data would be stored anonymously. To encourage participation, a gift card raffle was offered as a prize to interested participants.

5.2 Experiments

We now discuss the individual aspects of our experiments: model types, domains, participant demographics, and application of the Latin square design.

Experiment 1 was focused on *class diagrams*. In this experiment, in addition to the annotative and compositional mechanisms considered in all experiments, we included the enumerative mechanism as a baseline for comparison.

To select the systems, we specified a set of criteria that a subject system would need to fulfill: (*C1*) The system has been introduced in previous literature. (*C2*) The system comprises several variants. (*C3*) The system has not been introduced in a context related to a particular variability mechanism. The rationale of these criteria was to select systems that represent real variability, rather than making up artificial examples on the spot. As a result, we derived variability-enriched class diagrams for three domains: Simulink, a project management system, and a phone system. These domains were identified from literature based on their familiarity to the authors (*convenience sampling* (Wohlin et al. 2012)). For the former two domains, we were aware of several available variants in the literature. To systematically identify available variants, we performed database searches in Google Scholar, IEEExplore, and ACM's Digital Library, with the search strings "Project Management meta-model" and "Simulink meta-model." The considered variants of *Phone* correspond to the feature model from the original paper.

Simulink (Karris 2006) is a block-based modeling language that is widely applied in the design of embedded and cyber-physical systems. The absence of an official specification has given rise to the emergence of various variants. The *Project Management (PM)* (Fahrenberg et al. 2014) product line represents a family of software systems for project management, with concepts such as projects, activities, tasks, persons, and roles. The *Phone* product line, introduced by Benavides et al. (2010), represents a family of software systems for mobiles phones with various hardware functionalities, such as different cameras and displays.

For each domain, we designed three class diagrams: one per variability mechanism. With three mechanisms (*Enu*, *Ann*, *com*) and three domains (*d1*, *d2*, *d3*), following a *Latin square design* (Melo et al. 2016; Montgomery 2017), the paths were:

- *Enu d1→Ann d2→Com d3* (path 1),
- *Com d1→Enu d2→Ann d3* (path 2), and
- *Ann d1→Com d2→Enu d3* (path 3).

We recruited 73 participants, all of which were BSc students in a German university. The students were recruited from courses with completed previous lectures and homework assignments on class models. In line with our strategy to recruit students familiar with class models, students expressed an average level of expertise, amounting to 3.47 (mean) ± 0.60 (standard deviation). The self-reported programming expertise of 3.62 ± 0.74 was comparable. In contrast, the self-reported expertise in variability mechanisms was considerably lower, amounting to 1.73 ±0.87 in annotative mechanisms, 1.86 ±1.03 in compositional mechanisms, and 1.87 ±0.93 in enumerative mechanisms. The homogenous experience of our participants is beneficial for the validity of our findings, by countering a possible threat related to different previous knowledge. In the light of our justification for selecting students as participants (based on evidence for their suitability as stand-ins for developers (Salman et al. 2015)), our sample is representative for developers with similar experience levels in modeling and variability mechanisms.

Participants were randomly divided into three groups, and assigned one of the above-mentioned paths, allowing them to experiment with each variability mechanism and each domain once. Each participant was required to perform three tasks types for the three models, each represented using a different variability mechanism. At the end, the participants were asked to provide their subjective assessments and preferences (S1–S6), and rationale for their choices.

Experiment 2 considered state machine diagrams as model type. Both considered domains were based on the robotics programming game *Robocode* (Hartness 2004). The rationale was that students were involved with Robocode as a course project and were, therefore, familiar with it. Moreover, Robocode bots possessed considerable variability (Martinez et al. 2018), which allowed us to model state machines that were reasonably complex. Our considered domains represented two high-level features of Robocode robots, each using a separate state machine diagram. For each domain, we designed two state machines: one using the annotative mechanism, and another one using the compositional mechanism.

We recruited 65 students from a Swedish university (63 Bsc, 2 PhD). Participants were divided into two groups, and assigned into those randomly. In the preliminary assessment, on average, the participants rated their experience with state machine diagrams to be 2.846 ±0.587. Participants also gave an average rating of 3.42 ±0.749 to their programming experience. The self-reported experience of participants for the two variability mechanisms was considerably lower: 2.26 ±0.73 for annotative, and 2.52 ±0.77 for compositional. These ratings are, however, comparable to each other, and do not indicate any bias towards a particular mechanism.

Following our *Latin square* design, we exposed both sub-domains (*d1*, *d2*) to the groups in the same order, and reversed the order of the variability mechanism they were represented with.

- *Com d1* →*Ann d2* (path 1),
- *Ann d1* →*Com d2* (path 2).

Experiment 3 was focused on *activity diagrams* as model type. We designed activity diagrams for *two* domains: a *Flight reservation system (FRS)* and an *Email service provider (ESP)*. We created them upon our experience and by taking inspiration from the literature.

The participants in this experiment were *26 MSc students* studying at a Dutch university. Participants were randomly allocated into two groups. Participants indicated an average experience of 3.39 ±0.64 with activity diagrams, and 4.0 ±0.6 with programming. The mean scores for experiences with both variability mechanisms were significantly lower: 2.1 ±0.8 and 2.3 ±0.86 for annotative and compositional mechanisms respectively. The high ratings for experiences with activity diagrams and programming can be explained by our participants being MSc students, who have undergone significant practice with software design and implementation. The lower ratings for both variability mechanisms are also reasonable, since they were not taught in detail in the course. However, the ratings are close to one another, reducing any possible biases in the results.

Table 2 Correctness scores for experiment 1 (class diagrams)

Task type	Annotative			Compositional			Enumerative		
	Mean	Mdn	Sdt.dev	Mean	Mdn	Std.ev	Mean	Mdn	St.dev
1: Tracing elements to variants	1.7/2	2.0/2	0.6	1.5/2	2.0/2	0.7	1.7/2	2.0/2	0.6
2: Comparing two variants	1.5/2	1.5/2	0.6	1.5/2	1.5/2	0.6	1.6/2	1.5/2	0.4
3: Comparing all variants	1.6/2	2.0/2	0.7	1.2/2	1.0/2	0.7	1.5/2	2.0/2	0.7
Total	4.8/6	5.0/6	1.3	4.2/6	4.5/6	1.4	4.9/6	5.0/6	1.1

Mdn: Median, **St.dev:** Standard deviation

We designed two activity diagrams per domain, each represented using a different variability mechanism. With two domains (*d1*, *d2*), the paths after applying our *Latin square* design were:

– *Ann d1→Com d2* (path 1).
– *Com d1→Ann d2* (path 2),

6 Results

We now present the results from our experiments, structured along our research questions. For each, we have one subsection presenting the relevant results from all three experiments. This supports a direct comparison of our observations along our two independent variables (modeling language and variability mechanisms).

6.1 RQ1: Efficiency

In RQ1, we studied the effect of annotative and compositional mechanisms on *efficiency*, that is, the ability of our participants to solve model comprehension tasks correctly and quickly. To this end, we computed correctness scores and completion time based on the task responses, collected in the *modules* of our questionnaire (see Fig. 5). As explained in Section 5, the correctness score of a task type ranged between 0 and 2, based on aggregating the scores for the two questions in each task type. The completion time was determined by measuring the difference between the starting and ending time for completing all tasks for one particular variability mechanism.

Table 3 Correctness scores for experiment 2 (state machine diagrams)

Task type	Annotative			Compositional			Enumerative		
	Mean	Mdn	Sdt.dev	Mean	Mdn	Std.ev	Mean	Mdn	St.dev
1: Tracing elements to variants	1.2/2	1.0/2	0.8	1.2/2	1.0/2	0.7	–	–	–
2: Comparing two variants	1.0/2	1.0/2	0.8	1.1/2	1.0/2	0.8	–	–	–
3: Comparing all variants	1.2/2	1.0/2	0.8	0.8/2	1.0/2	0.7	–	–	–
Total	3.4/6	3.0/6	2.0	3.1/6	3.0/6	1.8	–	–	–

Mdn: Median, **St.dev:** Standard deviation

Table 4 Correctness scores for experiment 3 (activity diagrams)

Task type	Annotative			Compositional			Enumerative		
	Mean	Mdn	Sdt.dev	Mean	Mdn	St.dev	Mean	Mdn	St.dev
1: Tracing elements to variants	1.4/2	1.5/2	0.6	1.2/2	1.0/2	0.6	–	–	–
2: Comparing two variants	1.4/2	1.5/2	0.6	0.9/2	1.0/2	0.7	–	–	–
3: Comparing all variants	1.4/2	1.5/2	0.5	0.9/2	1.0/2	0.6	–	–	–
Total	4.2/6	4.0/6	1.0	3.0/6	3.0/6	1.3	–	–	–

Mdn: Median, **St.dev:** Standard deviation

Correctness. Tables 2, 3 and 4 provide a high-level overview of the correctness scores. Each table corresponds to one experiment and model type, and shows mean scores, median scores (Mdn), and standard deviations (St.dev) per task type and variability mechanism. A complementary, more detailed overview is offered by Fig. 6a to c which visualize the distribution of scores obtained for each task.

Considering class diagrams (Experiment 1, Table 2), the participants generally performed equally well with the annotative and the enumerative mechanisms. In contrast, the use of the compositional mechanism lead to a noticeable drop in mean performance. Hypothesis testing showed that the difference for compositional to other types was significant for task types 1 and 3. For type 1, we found $p = 0.01$ for the comparison to annotative, with a medium-ranged effect size of $A_{12} = 0.62$ ($p = 0.02$ for the comparison to enumerative, surpassing the corrected threshold). For type 3, we found $p < 0.01$ when comparing compositional to both annotative and enumerative, with medium effect sizes ($A_{12} = 0.66$ and 0.64, respectively). We did not find significant differences between the mechanisms for type 2. Annotative and enumerative do not differ significantly in any considered case.

The similar results obtained for annotative variability and the baseline (enumerative) for this and the following RQs motivated us to focus in experiments 2 and 3 on the comparison between annotative and compositional variability. Our rationale was that by not considering enumerative, we could allocate more time to annotative and compositional, which we could use for more involved questions within the task types that would allow a more in-depth comparison between these mechanisms.

Consequently, in Experiment 2 and Experiment 3, we observed lower average correctness scores than in Experiment 1. With state machine diagrams (Experiment 2, Table 3), there were no prominent differences between the results for annotative and compositional mechanisms for type 1 (1.2 vs 1.2) and 2 (1.0 vs 1.1). A significant difference was evident for the most complex task type, type 3, where annotative outperformed compositional with a score of 1.2 vs. 0.8. For type 3, hypothesis testing revealed a significant difference in performance using annotative and compositional with $p = 0.01$ and medium effect size ($A_{12} = 0.61$). We did not find significant differences between the mechanisms for types 1 and 2. An explanation is offered by the nature of the considered examples: Answering the questions was possible without an in-depth understanding of how the composition operator works. Hence, an inherent disadvantage of the compositional mechanism does not manifest itself in these examples. For task type 3, where we find statistically different results, this concern does not apply.

With activity diagrams (Experiment 3, Table 4), the differences were more pronounced. Participants performed better using annotative mechanism for type 1 (1.4 vs 1.2), type 2 (1.4

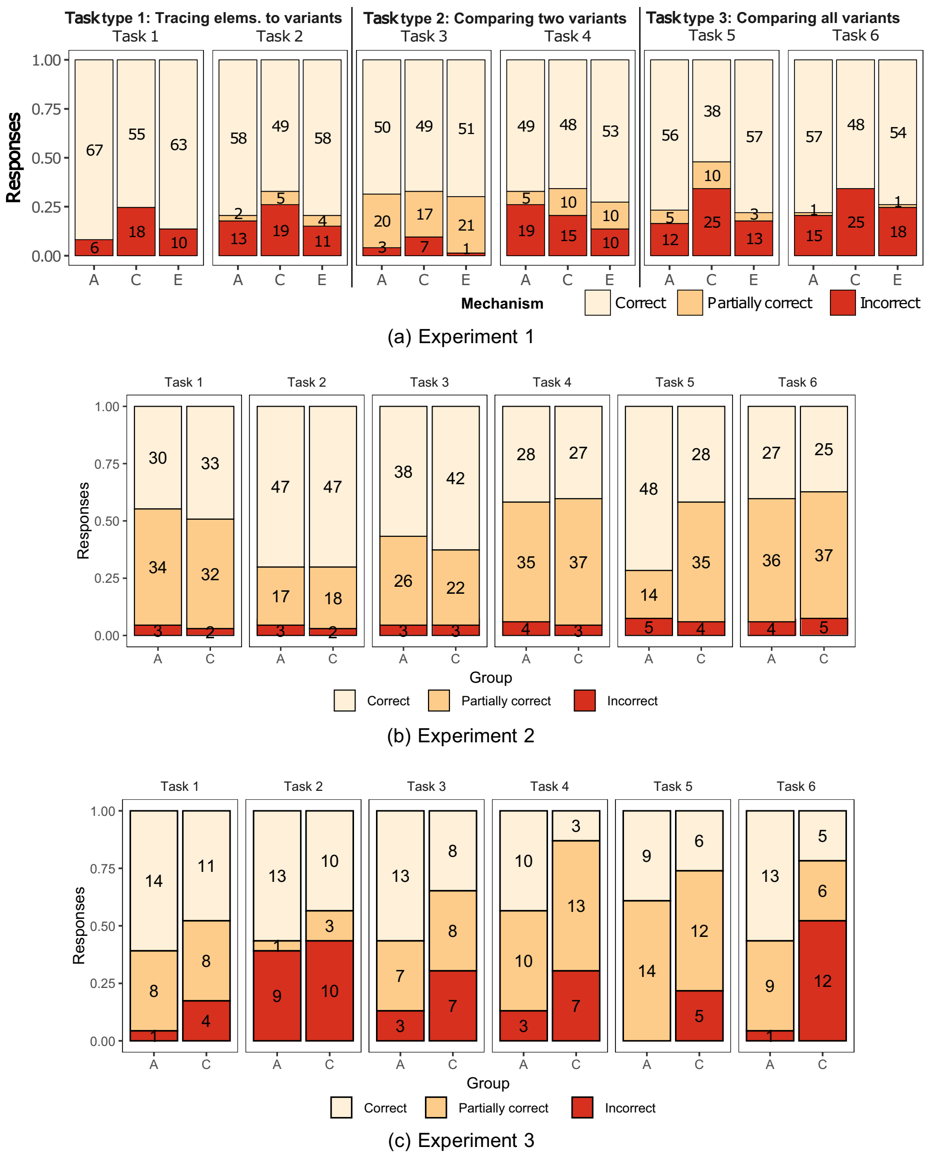

Fig. 6 Correctness scores for all three experiments **A:** Annotative **C:** Compositional **E:** Enumerative)

vs 0.9) and type 3 (1.4 vs 0.9). Hypothesis testing reveals significant differences for all task types, with p $<$ 0.01 for all types. The effect size of this comparison was small-to-medium for type 1 ($A_{12} = 0.59$), medium-to-large for type 2 ($A_{12} = 0.67$) and large for type 3 ($A_{12} = 0.76$). In the overall performance, the participants on average achieved a score 1.3 times higher with annotative than with compositional mechanism (4.2 vs 3.0). Compared to experiment 2, one noteworthy aspect that adds to the explanation of these findings is that

Table 5 Completion times (in minutes) of our participants for all three experiments

Exp	Mechanism	Min	Mn	Mdn	Max	St.dev
1	Annotative	3	6.6	6	15	2.6
	Compositional	4	8.8	8	17	3.2
	Enumerative	3	7.1	6	19	3.1
2	Annotative	3	10.4	10	23	4.3
	Compositional	5	11.3	11	22	3.7
3	Annotative	3	14.2	14	28	6.1
	Compositional	8	16.8	14	32	6.7

Mn: Mean, **Mdn:** Median, **St.dev:** Standard deviation

performing the tasks required a better understanding of the involved composition algorithm, since relevant information was spread over several composition fragments.

Annotative outperformed compositional variability on average in all three experiments. Specifically, the annotative mechanism lead to higher correctness scores on average than the compositional one for all task types and domains except one. The differences were significant for six out of nine cases (3 task types and 3 experiments). Compared to the baseline (enumerative variability, Experiment 1), annotative variability did not lead to significant performance differences, whereas compositional variability did.

Completion Time. Table 5 provides an overview of the completion times of participants for solving all tasks in all experiments. According to this data, the average time taken for performing tasks using annotative mechanism was always less than time taken using other mechanisms. Figure 7a, b, and c represent the distribution of completeness times taken by participants using different variability mechanisms for Experiment 1, 2, and 3 respectively.

In line with the observations about correctness, the speed of performing tasks was highest in experiment 1, due to the simpler tasks in each task type, which allowed us to consider the enumerative baseline solution in addition to annotative and compositional variability. The participants were fastest on average when using the annotative mechanism (mean completion time: 6.6 min), somewhat faster, but not significantly so than when using enumerative (7.1 min, $p = 0.12$). Participants, however, tasks took significantly longer when using the compositional mechanism (8.8 min). The differences between compositional and both annotative and enumerative were highly significant with $p < 0.001$. The effect size was large when comparing compositional and annotative ($A_{12} = 0.71$) and medium-to-large for compositional to enumerative ($A_{12} = 0.67$).

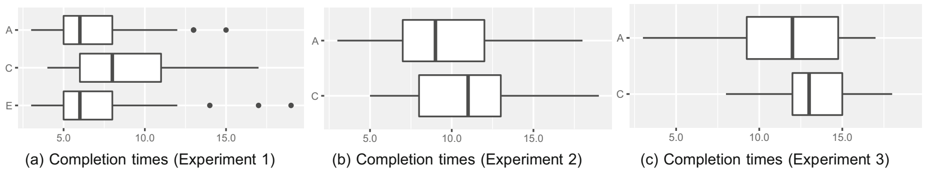

Fig. 7 Completion time in minutes for (a) Experiment 1 (b) Experiment 2 and c) Experiment 3. *Y-axis:* Variability mechanisms (**A:** Annotative **C:** Compositional **E:** Enumerative) *X-axis:* Time taken in minutes

In experiment 2, participants took roughly the same amount of time on average using the annotative and compositional mechanisms (10.4 vs 11.3). Hypothesis testing showed that the difference between times taken using both mechanisms was not significant ($p > 0.25$). These observations are in line with the correctness-related ones, in which only one of the considered task types (the most difficult one) lead to significant differences.

In experiment 3, participants were faster in performing their tasks on average using the annotative mechanism than with the compositional one (14.2 vs 16.8 min). However, this difference is not significant ($p > 0.38$). Contrasted with the significant differences in the correctness scores for all three task types, we find that the tendencies of both observations agree, but the implications for correctness appear to be greater than those for completion time.

The annotative mechanism lead to the shortest on-average completion times in all experiments. Participants took longest to complete the tasks when using the compositional mechanism. Yet, only in one out of three experiments, statistical significance was found. Compared to the baseline (enumerative variability, Experiment 1), annotative variability outperformed the baseline solution, but not significantly so.

6.2 RQ2: Subjective Perception

We report on the participant's subjective perceptions of understandability and difficulty to complete the tasks, based on our subjective assessment questions (S1–S4 in our questionnaire; see Fig. 5). The questions in this category were answered on a five-item Likert scale, with lower scores indicating better results. Table 6 gives an overview of the results,

Table 6 Participant perceptions (understandability[a] and difficulty[b] ratings) for Experiment 1, 2, and 3

Exp	Quality		Annotative			Compositional			Enumerative		
			Mean	Mdn	St.dev	Mean	Mdn	St.dev	Mean	Mdn	St.dev
1	Understandability		2.6/5	3/5	1.1	3.2/5	3/5	1.1	2.2/5	2/5	1.2
	Difficulty	Task type 1	2.3/5	2/5	1.2	3.1/5	3/5	1.2	2.3/5	2/5	1.1
		Task type 2	2.5/5	2/5	1.3	3.0/5	3/5	1.3	2.2/5	2/5	1.2
		Task type 3	2.5/5	2/5	1.2	3.2/5	3/5	1.3	2.5/5	2/5	1.1
2	Understandability		3.0/5	3/5	1.0	2.9/5	3/5	0.9	–	–	–
	Difficulty	Task type 1	2.6/5	3/5	1.0	2.6/5	3/5	1.0	–	–	–
		Task type 2	2.6/5	3/5	0.9	2.6/5	2/5	0.9	–	–	–
		Task type 3	2.9/5	3/5	1.0	2.9/5	3/5	1.0	–	–	–
3	Understandability		2.7/5	2/5	1.3	3.4/5	3/5	1.1	–	–	–
	Difficulty	Task type 1	2.3/5	2/5	1.0	3.2/5	3/5	0.9	–	–	–
		Task type 2	2.5/5	2/5	1.3	3.0/5	3/5	1.3	–	–	–
		Task type 3	2.5/5	2/5	1.2	3.2/5	3/5	1.3	–	–	–

[a]Scores on a 5-point Likert scale with 1: very easy, 5: very hard to understand

[b] Scores on a 5-point Likert scale with 1: very easy, 5: very difficult to perform task

Mdn: Median, **St.dev:** Standard deviation

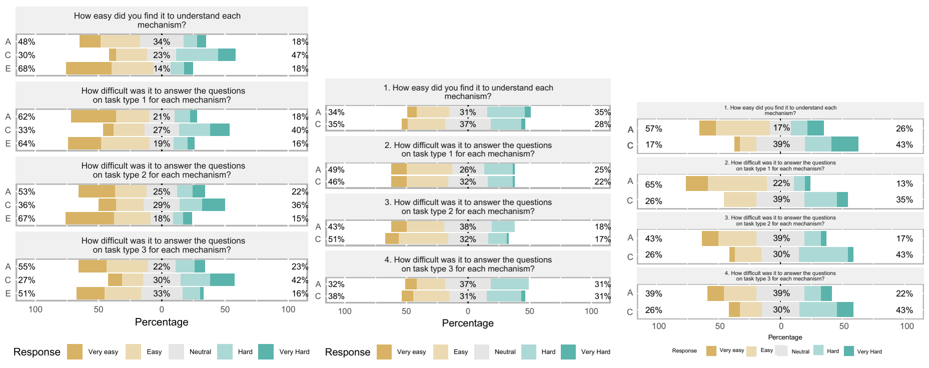

Fig. 8 Subjective perceptions of our participants. *Y-axis:* Variability mechanisms (**A:** Annotative **C:** Compositional **E:** Enumerative)

which are refined by the visualizations in Fig. 8a to c. The figures also include the exact formulations of all questions.

In Experiment 1 (class diagrams, Fig. 8a), enumerative was considered the easiest to understand (mean: 2.2), followed by the annotative mechanism (mean: 2.6). Compositional mechanism was the hardest to understand (mean: 3.2). The differences between the *understandability* of all three mechanisms were significant, with varying effect size measures: for enumerative vs. annotative, p = 0.006 with $A_{12} = 0.61$ (small to medium effect); for compositional vs. annotative, p = 0.004 with $A_{12} = 0.66$ (medium effect); for enumerative vs. compositional, p $\leq$ 0.001, with $A_{12} = 0.73$ (large effect).

The *difficulty* rating was fully consistent with both the objective task metrics (RQ1) and the understandability ratings. Regarding difficulty, annotative and enumerative were considered to be less difficult than compositional for all task types. Comparing the annotative and enumerative mechanism, the given mean ratings were approximately equal, amounting to 2.3, 2.5, 2.5 for annotative, and 2.3, 2.2, 2.5 for enumerative. We did not find statistical significance for this comparison, reinforcing our decision to not consider enumerative in the follow-up experiments. In contrast, the mean ratings for compositional of 3.1, 3.0 and 3.2 were much higher, indicating lower understandability. In all comparisons of compositional to another mechanism, we found significance. In all cases but one (task type 2, annotative vs. compositional: p = 0.03; $A_{12} = 0.62$), the p-value was below 0.003 and the effect size between $A_{12} = 0.65$ and 0.69, indicating a medium-to-large effect.

In experiment 2 (state machine diagrams, Fig. 8b), for understandability, participants gave approximately similar ratings to both annotative and compositional (3.0 vs. 2.9). The difference was not statistically significant. Considering difficulty, the mean scores per task type were identical (2.6, 2.6, 2.9 vs. 2.6, 2.6, 2.9), implying no statistically significant difference, largely consistent with the correctness scores and completion times for the same experiment (RQ1). Still, task type 3 gave rise to the only case in all our data where subjective assessment and objective performance were not aligned: the reported difficulty levels do not differ, while the correctness scores do so with statistical significance.

In experiment 3 (activity diagrams, Fig. 8c), participants considered annotative to be notably more understandable than compositional (2.7 vs. 3.4). The difference was statistically significant (p = 0.3), with $A_{12} = 0.68$ (medium-to-large effect size). Concerning difficulty, annotative was considered to be easier than compositional for all task types

(2.3, 2.5, 2.5 vs 3.2, 3.0, 3.2). The difference for type 2 was significant, with $p = 0.005$, and $A_{12} = 0.73$ (large effect). These observations agreed with the correctness scores and completion times in RQ1.

The subjective perceptions of our participants largely agreed with the objective performance measurements: In two of the three experiments, participants found the annotative mechanism more understandable and easier to work than the compositional one. The annotative mechanism was found (non-significantly) harder to understand, but equally easy to work with as the baseline (enumerative variability, Experiment 1).

6.3 RQ3: Subjective Preferences

We report on our participants' preferences, based on questions S5 and S6 in our questionnaire (see Fig. 5). In S5, we asked our participants to specify a preferred mechanism per task type (quantitative data). In S6, we asked them for textual feedback to explain the rationale for their preferences (qualitative data). In what follows, first, we present and discuss the distribution of preferences (S5), before explaining our observations based on the provided rationale (S6).

6.3.1 Quantitative Distribution of Subjective Preferences

Figure 9 and Table 7 provide an overview of our quantitative data: the percentages of selected answers when asked to specify a preferred variability mechanism per task type. For Experiment 1, we find that the preferences varied strongly between the tasks. Annotative was preferred by most participants for task types 1 and 3, albeit with only a moderate to slight difference to the enumerative mechanism: 50.7% vs. 34.2% for type 1, and 43.8% vs. 42.5% for type 3. In contrast, the enumerative mechanism was preferred with a large margin for task type 2, comparing two variants (which are explicitly present in the enumerative representation). Compositional came in last in all comparisons, with percentages between 12.3% and 15.1%. Participants generally expressed a preference; the *no-preference* option was only selected in 1.4% of all cases. Intuitively, the preference for enumerative for type 2

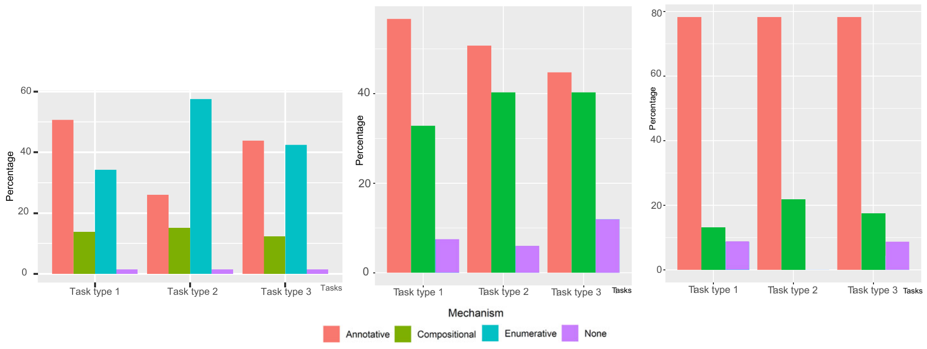

Fig. 9 Preference distributions of our participants in Experiment 1–3 (from left to right)

Table 7 Distribution of preferred mechanisms per task type

Task type	Ann.	Com.	Enu.	None
Experiment 1				
1 Understanding variants	50.7%	13.7%	34.2%	1.4%
2 Comparing two variants	26.0%	15.1%	57.5%	1.4%
3 Comparing all variants	43.8%	12.3%	42.5%	1.4%
Experiment 2				
1 Understanding variants	58.6%	33.8%	–	7.6%
2 Comparing two variants	52.3%	41.5%	–	6.2%
3 Comparing all variants	46.2%	41.5%	–	12.3%
Experiment 3				
1 Understanding variants	78.3%	8.7%	–	13.0%
2 Comparing two variants	78.3%	21.7%	–	0%
3 Comparing all variants	78.3%	17.4%	–	4.3%

Ann: Annotative Variability
Com: Compositional Variability
Enu: Enumerative Variability

is not surprising: in a comparison between two variants, explicitly representing the variants seems beneficial. Based on the preference of annotative for type 1 and 3, we hypothesize that this representation is suitable for tasks that require a good overview of all variants and the ability to trace elements to variants.

For Experiment 2, participants' preferences were consistent with their understandability ratings per mechanism. Participants preferred annotative over compositional for all task types: 58.6% vs. 33.8% for type 1, 52.3% vs. 41.5% for type 2, and 46.2% vs. 41.5% for type 3. The more pronounced preference of annotative for type 1 and 2 can be attributed to the consolidated view it offers, making it easier to understand and compare variants (also confirmed by the subjective assessments presented shortly). A considerable number of participants selected the *no-preference* option, especially for type 3 (12.3%). While this percentage is higher than in the other two experiments, one might still find it surprisingly low: Even though there were hardly any differences in the subjective assessment of understandability and difficulty (with low standard deviations), most participants still specified a preference. One could interpret this finding as supporting a role of personal *inclination* or *taste* in preferring a mechanism.

For Experiment 3, the differences were far more distinguished. The cleary majority of the participants preferred annotative over compositional for all task types (78.3% vs. 8.7% for type 1, 78.3% vs. 21.7% for type 2, and 78.3% vs. 17.4% for type 3). For type 1, 13% of the participants chose the *no-preference* option. The preferences were consistent with both the correctness scores and difficulty ratings. These observations are in line with our previous findings for this experiment, in which annotative was found easier to use and lead to better efficiency for all task types.

Compared to compositional, annotative was generally preferred for all task types, although with varying margins between the experiments. When also offered the baseline representation (enumerative, Experiment 1), the preferences varied between the task types: Annotative was preferred for comparing all variants, and for tracing elements to variants. Enumerative was largely preferred for comparing two variants to each other.

6.3.2 Qualitative Explanations of Subjective Preferences

To obtain additional insights, we asked the participants to explain their preferences intuitively using an open-ended question. Based on our manual assessment performed on the answers, we discuss the recurring aspects deemed as relevant by the participants below. Importantly, some of the reported aspects could be mitigated when working with proper tool support. Indeed, our findings might be useful for informing the question of what improved visualizations should focus on, as we further discuss in Section 6.4.

Conciseness. Models created using the annotative mechanism offer a *consolidated* view, where all variants of the domain are concisely shown as a single model. Eight of our participants mentioned this as a benefit: *"In the annotative one you had all the information asked on the first look; comparing was easy since the different [variants] were all in the same diagram."* With compositional, the models were *modular* in nature. This however made the models *scattered*, and therefore, hard to deal with. Five of our participants remarked that the modularity made the models clearer and eased the tasks: *"It [was] easier to conduct direct comparisons between different products in the [compositional mechanism], due to its modular nature."* However, the scatteredness of the models was linked to difficulty in performing tasks: *"[using compositional, you] need to look at a lot of screens/windows."* A related aspect mentioned by four of our participants was that the *co-located* presentation of information made the information readily available, and helped in performing tasks: *"The enumerative shows all required parts at once and you don't have to look really close to see all required parts and connections."*

Understandability. Eight of our participants found enumerative mechanism easy to understand and work with: *"the enumerative variant is the easiest."* None of the participants found it difficult to deal with. 26 of our participants found annotative easier to understand and follow: *"Annotative is simple to understand and just beautiful."* Two participants remarked that annotative mechanism was intuitive: *"[Annotative] is ... way more intuitive, it gives a better overview."* In contrast, six participants found annotative to be hard to manage, mainly because of the *cluttered* models: *"The main reason why I had some challenges with the [annotative] version was probably because the text was so clustered and hard to read."* Five of our participants found compositional to be easier to understand, either because they were already familiar with the mechanism, or they liked the modular structure: *"It was easier to understand compositional because of their modularity in my opinion."* 15 of our participants experienced difficulty in understanding the compositional mechanism, mainly because of its scattered nature: *"The compositional one seems more scrambled, and therefore a bit harder to follow and get a complete picture of."*

Familiarity. We observed that familiarity is a factor in preferring one mechanism over the other, if only in a participant's initial assessment. Specifically, we observed two interesting aspects of familiarity. First, participants preferred mechanisms they had experience with: *"Since I haven't worked with [annotative] variability before it was easier to grasp the compositional version."* Second, once participants experienced with a particular mechanism, they changed their preferences along the way: *"At first the annotative was hard (it took a while to understand what the starting state was), but once I understood it, it was easier to handle than the compositional one."*

Scalability. In their explanation of why they preferred a particular variability mechanism, several participants extrapolated from the considered case to more complex ones, and foresaw potential issues with the scalability of the notation. "*[In enumerative,] although you need more models/space, you can see everything relatively easy. However, if you have maybe like 20 variants, enumerative is probably not the way to go.*", "*last the Enumerative, the 6 variants were okay but when there are even more it is to much" [sic]*, and finally *"The problem with the Compositional was [the] bigger and more complex it gets, it is harder to understand in a short time.*"

Efficiency. Participants preferred mechanisms which they found quicker to work with, based on their subjective impressions. We observed three relevant dimensions. First, participants favoured the mechanisms which offered *readiness*; having readily available information at one look made the tasks easier. The mechanisms favoured with this rationale were enumerative and annotative: *"The enumerative shows all required parts at once and you don't have to look really close to see all required parts and connections,"* and *"In [compositional], you have to follow a path to find the differences, where as with [annotative], you can see it right away."* Second, four participants expressed that the additional step of combining fragments to create variants in compositional was intensive: *"Since you do not have to think about how to compose diagrams I think that annotative is easier."* Third, the scattered nature of compositional models made the tasks repetitive, an aspect two of our participants found laborious: *"The tagged approach of [annotative] does not require me to mentally jump back and forth across the diagram like I have to do whilst using compositional diagrams."*

Labels. When creating models, we deliberately chose labels that differed from the names of the model elements to make the models reasonably complex. An example from the Email Service Provider domain is *Security*, which was the label used to map the activities *encrypt email* and *decrypt email*. Participants rendered labels to be useful, especially in annotative models: *"the labels with variant on classes [were] very [helpful]."* Labels in the models with annotative mechanism were even more favoured, owing to their arrangement, which is collocated with model elements: *"The feature placement in the annotative mechanism was more localized, making it feel better to work with."*

Colors. A design choice was to show the feature names in the annotative representation in distinguished colors, based on previous recommendations for code-level mechanisms (Feigenspan et al. 2013). Doing so balances out a disadvantage of annotative representations: the use of labels increases information density and visual crowding (Whitney and Levi 2011), thus affecting readability. In line with these findings, participants noted that it was *"easier to compare classes in Annotative because of colors"*, and that *"the colouring of annotative diagrams make [task type 1] really easy"*.

Overview. We observed that participants chose mechanisms which offered a good *overview* of the variants. Overview of variants is partially similar to our task type 1 (understanding variants), however, it has a broader relevance to other model-related tasks such as comparison. Six of our participants preferred annotative, explaining that it gave a good overview: *"It seems easier for me to ... get a fuller understanding of the system when reading the annotative mechanism."* Two participants preferred enumerative, and one participant compositional with this rationale. Two of our participants experienced difficulty in getting an overview with the compositional mechanism: *"There were many more places to look when comparing with the compositional mechanism and it became a bit difficult to overview."*

Flow. In both behavioural model types (state machine diagrams and activity diagrams), participants strongly preferred annotative, expressing that the *flow* of the model made the tasks easier. 11 participants expressed their preference for annotative: *"[It was] easier to understand how everything is connected with the annotative."* None of the participants found compositional to be helpful in understanding flow, seven expressing that compositional made it hard to perform tasks because it did not support understanding flow: *"I felt it was harder to navigate with the compositional statechart diagram."*

Task Specific. Participants' preferences changed with task types. One participant commented: *"[task type 1 and task type 2] require more memorization about which features are active and which actions belong to them. As such, I prefer the compositional mechanism for these tasks, which is less cluttered and more cleanly displays only those sub-diagrams that are useful for a particular product. For the task type 3, it is likely that every feature will be enabled in at least one product. This means that you need to form a mental image of the diagram where all features are enabled. The fact that this is already contained in the annotative diagram is now an advantage. The compositional mechanism however, does not do this and as such requires more mental gymnastics in order to reason about a comparison of all products."*

Additional Aspects. Two of our participants experienced *uncertainty*, both with compositional: *"At first glance Compositional seemed easy but I was never 100% sure about my answers concerning it."* Two of our participants expressed that compositional mechanism assisted them to see only the *relevant* parts of the models: *"Compositional does the work of ignoring the unimportant parts for me; the boxes for each feature only contains the relevant state. ."* One participant observed that models are *harder to create* with the compositional mechanism: *"compositional based [...] is harder to construct and implement than the annotation based."* Two participants preferred annotative because it provided less information to be processed by the brain: *"[With compositional] you need to remember a lot of things, product names and it's features [..] Therefore I prefer the annotative notation, because you have to remember less mandatory features."*

We propose related recommendations in Section 6.4.

A main contributing factor to the large preference of annotative was its conciseness; some inherent disadvantages could be balanced out by the use of labels and colors. Participant gave a variety of additional explanations, related to understandability, familiarity, scalability, efficiency, ability to provide a good overview, and understanding of flow.

6.4 Discussion and Recommendations

The objective and subjective differences between variability mechanisms observed in our study can be considered by tool and language developers for improving user experience, an important prerequisite for MDE adoption (Abrahão et al. 2017). We discuss our findings in the light of derived recommendations.

Provide Flexible, Task-Oriented Representations. We find that there is no globally preferable variability mechanism—indeed, the "best" mechanism may depend on the task to be performed. Tool and language developers can support user performance and satisfaction by providing multiple representations, tailored to the task at hand. We propose to consider a spectrum of solutions, each trading off the desirable qualities *flexibility* and *simplicity*: As the most simple, but least flexible solution, one can augment a given representation with task-specific, read-only views, e.g., given an annotative representation, generate individual enumerated variants (or a subset thereof, see below). A second, more advanced solution is to make these additional representations editable, which offers more flexibility, but gives rise to a new instance of the well-known view-update problem (Czarnecki et al. 2009)—-the particular challenge here is to deal with the implications of layout changes. The third, most advanced solution is *projectional editing* (Behringer et al. 2017), in which developers interact with freely customizable representations of an underlying structure. Projectional editing offers the highest degree of flexibility, but poses a learning threshold to users for adapting to a new editing paradigm.

Support the Simple Solution, for Appropriate Use-Cases. Our participants preferred the simple enumerative solution for a subset of tasks. While being commonly applied in practice (e.g., in 9 out of 23 cases studied by Tolvanen and Kelly (2019b)), this solution is inherently problematic: In small to moderate product lines, organizations struggle with the propagation of changes between cloned variants (Schlie et al. 2017). In large product lines, considering a distinct model for each of thousands of variants is simply infeasible. Instead, we suggest to address use-cases that involve a clearly defined subset of variants: In staged configuration processes (Czarnecki et al. 2005; Schroeter et al. 2012), such subsets are derived by incrementally reducing the variant space, thus obtaining partial configurations of the system. Variability viewpoints (Kaufmann et al. 2014), which are applied at companies like Daimler, reduce the variant space based on the perspective of a specific stakeholder. To address theses use cases, we suggest to provide support for selecting and interacting with a subset of enumerated variants, while using a proper variability mechanism for maintaining the overall system.

Use Colors, and Use Them Carefully. In line with the existing literature (Feigenspan et al. 2013), we find that colors can be helpful for mitigating the drawbacks of annotative techniques. However, relying on colors in an unchecked way is undesirable due to the prevalence of color-blindness. Up to 8% of males and 0.5% of females of Northern European descent are affected by red-green color blindness (Chan et al. 2014). A recommendation for language and tool designers is to avoid representations that solely rely on color, and to use dedicated color-blindness simulators such as Sim Daltonism (https://michelf.ca/projects/sim-daltonism/) to check their tools. A further issue with colors might be scalability, in a situation when there are many variation points that would need to be shown in different colors. A mitigation would be to flexibly reassign colors depending on the current screen focus, but even this approach would have limitations there is a need for a high number of annotation colors inside the same screen. Alternative visualization aids (for example, filtering) could be combined with coloring and are highly desirable.

Composition: the Whole is More Than the Sum of Its Parts. Participants struggled with compositional mechanisms especially in tasks that required them to understand how fragments interact with each other. In the case of behavior modeling (Experiment 2 and 3), the observed performance differences are more pronounced when participants had to understand cross-fragment flow of activities or states. To be able to solve the tasks, they essentially had to execute the composition algorithm in their mind. Potentially, this drawback of compositional mechanisms can be balanced out by providing special visualizations that illustrate the composition of (subsets of) fragments. Such visualizations would reduce cognitive load during model comprehension and would also allow to provide instant feedback upon changes.

Structured Overview of Recommendations. Based on our results and the considerations in this section, we derive a structured overview of recommendations to tool developers and users, which we outline in Table 8. As the main factors that should determine the selection of the variability mechanism, we identify the *expected number of variants* and the *expected type of tasks*. This is because we saw different optimal (best-perfoming and preferred) mechanisms depending on the task type, and because the enumeration of variants, which works particularly well for comparing two variants, does not scale to large variant sets. To distinguish small from large variant sets, we use a size threshold of 10, based on the magnitude of the variant sets considered in our experiments. For small large sets, we propose to use either annotative or enumerative, depending on the tasks most likely to occur. For large sets, the considerations in the beginning of this section are particularly relevant: flexible, task-specific editing support can be used to show the information currently of interest. This helps to balance out scalability issues, notably the scalability issue with colors in the annotative, and the issue with having an overly large set of variants in the enumerative mechanism.

7 Threats to Validity

We discuss the threats to validity of our study, following the recommendations by Wohlin et al. (2012).

External Validity. Our experiment focuses on class models, state machine diagrams, and activity diagrams, three ubiquitous model types. We discuss representativeness and practical relevance in MDE contexts in Section 2. Furthermore, we consider only two variability mechanisms in our comparison, an annotative and a compositional one. While studying a broader selection of modeling languages and mechanisms is left to future work, the

Table 8 Overview of recommendations derived from our results

Expected number of variants		Expected type of tasks		Recommendation
Small ($<$10)	Large ($\geq$10)	Comparing all variants, understanding individual variants	Comparing two variants	
●		●		Annotative or enumerative
●			●	Enumerative
●		●	●	Annotative or enumerative
	●	●		Flexible
	●		●	Flexible
	●	●	●	Flexible

qualitative data presented in Section 6.3.2 is not necessarily specific to these model types and mechanisms, and yields a promising outlook on generalizability.

Another issue is whether our results generalize to larger systems, specifically, those with more variants and model elements. Since the number of variants grows exponentially with the number of features, the enumerative representation will eventually be outperformed by the other ones. We discuss possible roles for the enumerative representation in larger systems in Section 6.4.

Finally, external validity is threatened by our sample of participants, made up of students with limited prior experience with variability mechanisms. Student participants can be representative stand-ins for practitioners in experiments that involve new development methods (Salman et al. 2015). While considering a broader spectrum of experience levels would be worthwhile, we arguably focus on a critical population: In a given organization, consider the onboarding of a new team member with a similar experience level to our participants. Poor comprehension would pose a major hurdle to becoming productive, and, therefore, pose a risk for the organization. As mitigation measures, we established via our questionnaire that our participants had no unacceptable advantage of being much more familiar with either compositional or annotative variability, and selected subject domains that were either simple or already known to the students from previous course units (thus avoiding potential misunderstandings as a source of error).

Internal Validity. Within-subject designs help to elicit a representative number of data points to support statistically valid conclusions. We addressed their drawbacks as follows: To address learning effects, we applied counterbalancing. Between the different groups, we distributed the order of variability mechanisms equally, while keeping the domain and task order constant. To balance the assignment of participants to classes, we randomized the assignment.

To avoid researcher bias in the selection of our examples, we selected domains that were not used before with a specific variability mechanism. Studying comprehension in larger models is desirable, but has some principle limitations with regard to the amount of information that participants can be exposed to in the scope of an experiment (participant fatigue).

A possible source of bias is our choice of subject domains: a particular domain might have been used in previous teaching units to explain a particular variability mechanism, which would give an unfair advantage to these mechanism. We intentionally selected domains that were simple and/or known to the students. However, to our knowledge, they were not used to teach particular variability mechanisms. Using our questionnaire, we asked our students to specify their previous knowledge about variability mechanisms, and did not find a noteworthy difference between the mechanisms.

Construct Validity. We operationalized comprehensibility with comprehension tasks, arguing for the importance of the considered tasks in Section 5. The choice of tasks was informed by our pre-study, in which we encountered trade-offs regarding participant fatigue and confounding factors when using more demanding tasks (Section 4). Since we find significant performance differences between mechanisms, the difficulty level of our tasks seems appropriate; however, other tasks might exist (e.g. understanding a single feature and its context), and task completion could also be facilitated if users are supported by specialized tools (e.g. query engines). Generally, systematic knowledge on the design space of model comprehension tasks would help to maximize realism in comprehension experiments, but such knowledge is currently lacking.

Our setup did not involve tools, representing an unavoidable trade-off: While having the participants use a tool environment would have been more realistic, it would have lead to confounding factors related to usability obstacles and participants' familiarity with the tool. On the other hand, working with a printout that could be derived from a tool (like in our experiments) also has some significant commonalities to working with the tool. We study the question of how well users can comprehend models in a particular visual representation. Given that the representation looks the same in the tool and on printout, we believe that construct validity for our results and recommendations is established. Notably, while adequate tool support might be able to mitigate some issues of particular representations, the developers of these tools should at least be aware of these issues.

Colors were only used in the annotative representation, where their usefulness (for distinguishing elements from different variants) seems more obvious than in the compositional one (where such elements are already distinguished by being contained in different modules). A follow-up study for studying the impact of colors in different representations might provide additional insight.

Subjective measures are generally less reliable than objective ones. However, previous findings suggest that they are correlated with objective performance measurements (Gopher and Braune 1984). In fact, we find an agreement between the subjective and objective measurements performed in our experiments.

Conclusion Validity. Towards supporting conclusion validity, we used robust statistical tests. We report effect sizes and, to deal with threats related to multiple comparisons, applied a conservative correction to the considered significance threshold. Threats arising from random irrelevancies outside the experimental setting are reduced in experiments 1 and 2, which were administered with all participants in the same room. Experiment 3, administered during the COVID19 pandemic, is more prone to such irrelevancies, which, however, would equally apply to all treatments.

8 Related Work

Annotative vs. Compositional Variability. Annotative approaches are traditionally seen as inherently problematic. Spencer and Collyer (1992), for example, argues that #ifdef usage in C as a means to cope with variability is harmful, leading to convoluted, unreadable, and unmaintainable code (the infamous *"#ifdef hell"*). Spencer appeals to basic principles of good software engineering: explicit interfaces, information hiding, and encapsulation. Kästner et al. (2011) argue that compositional approaches tend to promise advantages, which, however, only become manifest under rather specific assumptions. They emphasize that only empirical research can provide conclusive evidence.

Aleixo et al. (2012) compare both mechanism types in the context of *Software Process Line* engineering, i.e., applying concepts and tools from SPL engineering to software processes. They compare two established tools: EPF Composer, which uses a compositional mechanism, and GenArch-P, which uses an annotative one. Similar to our conclusions, they report that the annotative mechanism performs better, especially with regards to a criterion the authors call *adoption*, i.e., how much knowledge is required to initially apply the mechanism.

Empirical Studies of Variability Mechanisms. Krüger et al. (2019) present a comparative experimental study of two variability mechanisms: decomposition into classes, and annotations of code sections. They find that annotations have a positive effect on program comprehension, while the decomposition approach shows no significant improvement and, in some cases, a negative effect. While these findings are in line with ours, this study focuses on Java programs, and compares the considered mechanisms to a different baseline, pure OO code without any traces of variants. In our case, we considered the frequent case in industry of copied and reused model variants.

Fenske et al. (2017) present an empirical study based on revision histories from eight open-source systems, in which they study the effect of #ifdef preprocessors to maintainability. They analyze maintainability in terms of change frequency, which is known to be correlated with error-proneness and change effort. In contrast to the traditional belief, they find that a negative effect of #ifdefs to maintainability cannot be confirmed.

Santos et al. (2019) study the effect of using two code-level variability representations: feature-oriented programming (FOP) and conditional compilation (CC), on program comprehension. In their case, they focus on debugging task–a different scope than in our case. Specifically, they investigate the impact of FOP and CC on various maintenance tasks involving bug-finding. They conclude that there is no significant difference between the correctness, understanding, and response time of using both representations.

Feigenspan et al. (2013) study the potential of background colors as an aid to support program comprehension of source code with #ifdef preprocessors. In three controlled experiments with varying tasks and program sizes, they find that background colors contributed to better program comprehension and were preferred by the participants. We base the use of color in our experiments on these findings, and confirm them for the previously unconsidered case of a model-level variability mechanism.

Empirical Studies of Model Comprehension. Labunets et al. (2017) study graphical and tabular models representations in security risk assessment. In two experiments, they find that participants prefer both representations to a similar degree, but perform significantly better when using the tabular one. The authors build on cognitive fit theory (Vessey 1991) to explain their findings: tables represent the data in a more suitable way for the considered task. Like we do, this study supports the need for task-tailored representations.

Nugroho (2009) studies the effect of *level of detail* (LoD) on model comprehension. In an experimental evaluation with students, the author finds that a more detailed representation contributes to improved model understanding. Ramadan et al. (2020) find a positive effect to comprehension of security and privacy aspects when graphical annotations are included in the considered models. Our results are in line with these findings, since the annotative mechanism includes the names of the associated variants as one point of additional information.

Acreţoaie et al. (2018) empirically assess three model transformation languages with regard to comprehensibility. They consider a textual language and two graphical ones, one of which uses stereotype annotations to specify change actions in UML diagrams. They observe best completion times and lowest cognitive load when using the graphical language with annotations, and best correctness when using the textual language. Studying this tradeoff further, by studying variability mechanisms in graphical and textual representations, would be an interesting extension of our work.

Model-Level Variability. There are two main research directions of variability engineering at the model level: variability modeling and model-level variability mechanisms. The former focuses on the modeling of the problem space (e.g., features and their relationships); the latter, which is the scope of this paper, focuses on the solution space (implementation of variability in domain models). Regarding variability modeling, there is a number of experience reports from various domains. Alférez et al. (2019) observe that existing variability modeling approaches do not suffice to capture a number of aspects of video domain, including numeric parameters, multifeatures and constraints. They qualitatively compare a set of 13 approaches based on their ability to support the above-mentioned aspects, and present a new textual variability modeling language (VM) for modeling videos. Berger et al. investigate variability modeling of topological spaces (Berger et al. 2014b). They share their experiences from modeling large-scale fire alarm systems using UML2 class diagrams, and show that class diagrams are an effective tool to model topological variability to generate configurator tools. García et al. (2019) give an experience report on the modeling of variability in robotic applications. They use feature models as a notation for defining variability of robotic applications in various dimensions, such as environment, hardware, and mission variability.

Most existing work on model-level variability mechanisms focuses either on particular mechanisms (see Section 2) or the use of such mechanisms for other tasks; e.g., product line testing (Reuys et al. 2006) and product derivation (Ziadi and Jézéquel 2006). Kühn and Cazzola (2016) compare two particular approaches to variability engineering in DSLs along three dimensions–feasibility, scalability, and sustainability—based on intuitive arguments. In summary, the impact of variability mechanisms on model comprehension has not been investigated yet in an empirical study. Our study is the first to use a controlled experiment to produce systematic evidence on the impact of variability representations on model comprehension, and as such, can be used to guide the choice of variability mechanisms when modelling the static and dynamic structure of product lines.

9 Conclusion

We presented a family of controlled experiments, in which we studied the effect of the variability mechanisms of two fundamental kinds—annotative and compositional mechanisms—on model comprehension. Our considered model types—class diagrams, state machine diagrams, and activity diagrams—are among the most popular and common models used in software engineering. We conducted the study with student participants with relevant background knowledge. For models with a scope and size similar to our examples and for similar tasks, we can conclude that:

- Annotative variability resulted in better comprehensibility than compositional variability for all task types.
- Compositional mechanism can impair comprehensibility in tasks that require a good overview of all variants.
- Annotative variability is preferred over the compositional one by a majority of the participants for all task types in all model types.
- The preferred variability mechanism depends on the task at hand.

We presented several recommendations to language and tool developers. We discuss a spectrum of solutions for maintaining several task-tailored representations. Having such solutions is especially important in large systems where maintaining a separate model per variant is infeasible, but developers might still want to interact with (sets of) variants of interest in a particular task. We endorse the recommendation to use colors for improving comprehension in annotative variability, and discuss its limitations. If a compositional mechanism is desired, users should be supported with visualizations, instead of being required to perform composition in their minds.

We envision four directions of future work. First, we want to understand the effect of tools to model comprehension. Second, we wish to systematically explore the space of typical tasks during model comprehension. Additional experiments would allow us to come up with a catalog of task-specific recommendations for variability mechanism use. Third, we are interested in broadening the scope of our experiments to take different modeling languages into account, including textual ones, which represent a middle ground between traditional programming languages and graphical modeling languages, and transformation languages, for which many different reuse mechanisms have recently been developed (Chechik et al. 2016; Strüber and Anjorin 2016). Fourth, further insight from our collected data could be obtained by performing sub-group analysis. Questions of interest are the effect of prolonged interaction of developers with different model types on their performance for the three task types, and the effect of mechanisms on the types of errors made by our participants (e.g., false negatives vs. false positives).

Acknowledgements We thank the reviewers and the participants from our pre- and three main experiments, as well as the reviewers for their valuable comments. This work is supported by the Swedish Research Council and the Wallenberg Academy.

Funding Open access funding provided by University of Gothenburg.

Declarations

Conflict of Interest The authors have no competing interests to declare that are relevant to the content of this article.

References

Abdi H (2007) Bonferroni and šidák corrections for multiple comparisons. Encycl Meas Stat 3:103–107

Abrahão S, Bourdeleau F, Cheng B, Kokaly S, Paige R, Stöerrle H, Whittle J (2017) User experience for model-driven engineering: challenges and future directions. In: MODELS. IEEE, pp 229–236

Acreţoaie V, Störrle H, Strüber D (2018) Vmtl: a language for end-user model transformation. Softw Syst Model 17(4):1139–1167

Aleixo FA, Freire MA, da Costa DA, Neto EC, Kulesza U (2012) A comparative study of compositional and annotative modelling approaches for software process lines. In: SBES, pp 51–60

Alférez M, Acher M, Galindo JA, Baudry B, Benavides D (2019) Modeling variability in the video domain: language and experience report. Softw Qual J 27(1):307–347

Ali S, Briand LC, Rehman MJU, Asghar H, Iqbal MZZ, Nadeem A (2007) A state-based approach to integration testing based on uml models. Inf Softw Technol 49(11-12):1087–1106

Alwidian S, Amyot D (2019) Union models: support for efficient reasoning about model families over space and time. In: SAM. Springer, pp 200–218

Anjorin A, Saller K, Lochau M, Schürr A (2014) Modularizing triple graph grammars using rule refinement. In: FASE, pp 340–354

Apel S, Kästner C (2009) An overview of feature-oriented software development. J Object Technol 8(5):49–84

Apel S, Janda F, Trujillo S, Kästner C (2009) Model superimposition in software product lines. In: ICMT. Springer, pp 4–19

Apel S, Batory D, Kästner C, Saake G (2013) Feature-oriented software product lines: concepts and implementation

Assunção WK, Vergilio SR, Lopez-Herrejon RE (2017) Discovering software architectures with search-based merge of uml model variants. In: ICSR. Springer, pp 95–111

Assunção WK, Vergilio SR, Lopez-Herrejon RE (2020) Automatic extraction of product line architecture and feature models from uml class diagram variants. Inf Softw Technol 117(106):198

Behringer B, Palz J, Berger T (2017) Peopl: projectional editing of product lines. In: ICSE. IEEE

Benavides D, Segura S, Ruiz-Cortés A (2010) Automated analysis of feature models 20 years later: a literature review. Inf Syst 35(6):615–636

Berger T, Pfeiffer RH, Tartler R, Dienst S, Czarnecki K, Wasowski A, She S (2014a) Variability mechanisms in software ecosystems. Inf Softw Technol 56(11):1520–1535

Berger T, Stănciulescu Ş, ØGård O, Haugen Ø, Larsen B, Wąsowski A (2014b) To connect or not to connect: experiences from modeling topological variability. In: SPLC, pp 330–339

Berger T, Steghöfer JP, Ziadi T, Robin J, Martinez J (2020) The state of adoption and the challenges of systematic variability management in industry. Empir Softw Eng 25:1755–1797

Bergman MM (2011) The good, the bad and the ugly in mixed methods research and design

Bruneliere H, Cabot J, Jouault F, Madiot F (2010) Modisco: a generic and extensible framework for model driven reverse engineering. In: Proceedings of the IEEE/ACM international conference on automated software engineering, pp 173–174

Chan XBV, Goh SMS, Tan NC (2014) Subjects with colour vision deficiency in the community: what do primary care physicians need to know? Asia Pac Family Med 13(1):10

Chauvel F, Jézéquel JM (2005) Code generation from uml models with semantic variation points. In: MODELS. Springer, pp 54–68

Chechik M, Famelis M, Salay R, Strüber D (2016) Perspectives of model transformation reuse. In: IFM. Springer, pp 28–44

Chen PPS (1976) The entity-relationship model—toward a unified view of data. ACM Trans Database Syst (TODS) 1(1):9–36

Clarke D, Helvensteijn M, Schaefer I (2010) Abstract delta modeling. CW Reports

Classen A, Heymans P, Schobbens PY, Legay A, Raskin JF (2010) Model checking lots of systems: efficient verification of temporal properties in software product lines. In: ICSE. ACM, pp 335–344

Czarnecki K, Antkiewicz M (2005) Mapping features to models: a template approach based on superimposed variants. In: GPCE. Springer, pp 422–437

Czarnecki K, Pietroszek K (2006) Verifying feature-based model templates against well-formedness ocl constraints. In: GPCE. ACM, pp 211–220

Czarnecki K, Helsen S, Eisenecker U (2005) Staged configuration through specialization and multilevel configuration of feature models. Softw Process: Improvement Pract 10(2):143–169

Czarnecki K, Foster JN, Hu Z, Lämmel R, Schürr A, Terwilliger JF (2009) Bidirectional transformations: a cross-discipline perspective. In: ICMT. Springer, pp 260–283

De Winter JC, Dodou D (2010) Five-point likert items: t test versus Mann-Whitney-Wilcoxon. Pract Assess Res Eval 15(11):1–12

Domı E, Pérez B, Rubio ÁL et al (2012) A systematic review of code generation proposals from state machine specifications. Inf Softw Technol 54(10):1045–1066

Fahrenberg U, Acher M, Legay A, Wąsowski A (2014) Sound merging and differencing for class diagrams. In: FASE

Faitelson D, Tyszberowicz SS (2017) UML diagram refinement (focusing on class- and use case diagrams). In: ICSE, pp 735–745

Famelis M, Lúcio L, Selim G, Di Sandro A, Salay R, Chechik M, Cordy JR, Dingel J, Vangheluwe H, Ramesh S (2015) Migrating automotive product lines: a case study. In: International conference on theory and practice of model transformations. Springer, pp 82–97

Feigenspan J, Kästner C, Apel S, Liebig J, Schulze M, Dachselt R, Papendieck M, Leich T, Saake G (2013) Do background colors improve program comprehension in the #ifdef hell? Empir Softw Eng 18(4):699–745

Fenske W, Schulze S, Saake G (2017) How preprocessor annotations (do not) affect maintainability: a case study on change-proneness. In: GPCE, vol 52, pp 77–90

Flores R, Krueger C, Clements P (2012) Mega-scale product line engineering at general motors. In: Proceedings of SPLC

García S, Strüber D, Brugali D, Di Fava A, Schillinger P, Pelliccione P, Berger T (2019) Variability modeling of service robots: experiences and challenges. In: Vamos, pp 1–6

Gessenharter D, Rauscher M (2011) Code generation for uml 2 activity diagrams. In: ECMFA

Gopher D, Braune R (1984) On the psychophysics of workload: why bother with subjective measures? Hum Factors 26(5):519–532

Hartness K (2004) Robocode: using games to teach artificial intelligence. J Comput Sci Coll 19(4):287–291

Heidenreich F, Kopcsek J, Wende C (2008) Featuremapper: mapping features to models. In: ICSE-companion. ACM, pp 943–944

Jones B, Kenward MG (2003) Design and analysis of cross-over trials. Chapman and Hall/CRC

Karris ST (2006) Introduction to Simulink with engineering applications. Orchard Publications

Kästner C, Apel S, Ostermann K (2011) The road to feature modularity? In: SPLC Companion, p 5

Kaufmann T, Manz C, Weyer T (2014) Extending the SPES modeling framework for supporting role-specific variant management in the engineering process of embedded software. In: SE, pp 77–86

Krüger J, Calikli G, Berger T, Leich T, Saake G (2019) Effects of explicit feature traceability on program comprehension

Kühn T, Cazzola W (2016) Apples and oranges: comparing top-down and bottom-up language product lines. In: SPLC, pp 50–59

Kulkarni V, Venkatesh R, Reddy S (2002) Generating enterprise applications from models. In: OOIS. Springer, pp 270–279

Labunets K, Massacci F, Paci F, Marczak S, de Oliveira FM (2017) Model comprehension for security risk assessment: an empirical comparison of tabular vs. graphical representations. Empir Softw Eng 22(6):3017–3056

Langer P, Mayerhofer T, Wimmer M, Kappel G (2014) On the usage of uml: initial results of analyzing open uml models. Modellierung

Mahmood W, Strber D, Anjorin A, Berger T (2021) Online appendix to “effects of variability in models: a family of experiments”. https://sites.google.com/view/variabilitymodelingexperiment/home, https://zenodo.org/record/5578645

Martinez J, Ziadi T, Bissyande TF, Klein J, Le Traon Y (2015) Automating the extraction of model-based software product lines from model variants. In: ASE. IEEE, pp 396–406

Martinez J, Tërnava X, Ziadi T (2018) Software product line extraction from variability-rich systems: the robocode case study. In: Proceedings of the 22nd international systems and software product line conference, vol 1, pp 132–142

Melo J, Brabrand C, Wasowski A (2016) How does the degree of variability affect bug finding? In: ICSE, pp 679–690

Montgomery DC (2017) Design and analysis of experiments. Wiley

Moreno N, Fraternali P, Vallecillo A (2007) Webml modelling in uml. IET Softw 1(3):67–80

Mussbacher G, Araújo J, Moreira A, Amyot D (2012) Aourn-based modeling and analysis of software product lines. Softw Qual J 20(3):645–687

Nugroho A (2009) Level of detail in uml models and its impact on model comprehension: a controlled experiment. Inf Softw Technol 51(12):1670–1685

Peldszus S, Strüber D, Jürjens J (2018) Model-based security analysis of feature-oriented software product lines. In: GPCE. ACM, pp 93–106

Petre M (2013) Uml in practice. In: ICSE. IEEE, pp 722–731

Pietsch C, Kehrer T, Kelter U, Reuling D, Ohrndorf M (2015) Sipl–a delta-based modeling framework for software product line engineering. In: ASE. IEEE, pp 852–857

Pohl K, Böckle G, van Der Linden FJ (2005) Software product line engineering: foundations, principles and techniques. Springer Science & Business Media

Ramadan Q, Strüber D, Salnitri M, Jürjens J, Riediger V, Staab S (2020) A semi-automated bpmn-based framework for detecting conflicts between security, data-minimization, and fairness requirements. Softw Syst Model 1–37

Reuys A, Reis S, Kamsties E, Pohl K (2006) The scented method for testing software product lines. In: Software product lines. Springer, pp 479–520

Rubin J, Chechik M (2013) N-way model merging. In: FSE. ACM, pp 301–311

Salman I, Misirli AT, Juristo N (2015) Are students representatives of professionals in software engineering experiments? In: ICSE, pp 666–676

Santos AR, do Carmo Machado I, de Almeida ES, Siegmund J, Apel S (2019) Comparing the influence of using feature-oriented programming and conditional compilation on comprehending feature-oriented software. Empir Softw Eng 24(3):1226–1258

Schaefer I (2010) Variability modelling for model-driven development of software product lines. In: Vamos, pp 85–92

Schlie A, Wille D, Schulze S, Cleophas L, Schaefer I (2017) Detecting variability in matlab/simulink models: an industry-inspired technique and its evaluation. In: SPLC, pp 215–224

Schroeter J, Mucha P, Muth M, Jugel K, Lochau M (2012) Dynamic configuration management of cloud-based applications. In: SPLC, pp 171–178

Schwägerl F, Buchmann T, Westfechtel B (2015) Supermod—a model-driven tool that combines version control and software product line engineering. In: ICSOFT, vol 2. IEEE, pp 1–14

Seidl C, Schaefer I, Aßmann U (2014) Deltaecore-a model-based delta language generation framework. Modellierung 2014

Spencer H, Collyer G (1992) #ifdef considered harmful, or portability experience with C news. In: USENIX

Steinberg D, Budinsky F, Merks E, Paternostro M (2008) EMF: eclipse modeling framework. Pearson Education

Strüber D, Anjorin A (2016) Comparing reuse mechanisms for model transformation languages: design for an empirical study. In: Hufamo@ moDELS, pp 27–32

Strüber D, Schulz S (2016) A tool environment for managing families of model transformation rules. In: ICGT. Springer, pp 89–101

Strüber D, Mukelabai Mukelabai JK, Fischer S, Linsbauer L, Martinez J, Berger T (2019) Facing the truth: benchmarking the techniques for the evolution of variant-rich systems. In: SPLC, pp 26:1–26:12

Strüber D, Anjorin A, Berger T (2020) Variability representations in class models: an empirical assessment. In: MODELS, pp 240–251

Taentzer G, Salay R, Strüber D, Chechik M (2017) Transformations of software product lines: a generalizing framework based on category theory. In: MODELS. IEEE, pp 101–111

Tolvanen JP, Kelly S (2011) Creating domain-specific modeling languages for product lines. In: 2011 15th international software product line conference. IEEE, pp 358–358

Tolvanen JP, Kelly S (2019a) Describing variability with domain-specific languages and models. In: Proceedings of the 23rd international systems and software product line conference, vol A, pp 329–329

Tolvanen JP, Kelly S (2019b) How domain-specific modeling languages address variability in product line development: investigation of 23 cases. In: SPLC, pp 24:1–24:9

van der Linden FJ, Schmid K, Rommes E (2007) Software product lines in action: the best industrial practice in product line engineering. Springer

Van Gurp J, Bosch J, Svahnberg M (2001) On the notion of variability in software product lines. In: ICSA. IEEE, pp 45–54

Vargha A, Delaney HD (2000) A critique and improvement of the cl common language effect size statistics of Mcgraw and Wong. J Educ Behav Stat 25(2):101–132

Vaupel S, Taentzer G, Harries JP, Stroh R, Gerlach R, Guckert M (2014) Model-driven development of mobile applications allowing role-driven variants. In: MODELS. Springer, pp 1–17

Vessey I (1991) Cognitive fit: a theory-based analysis of the graphs versus tables literature. Decis Sci 22(2):219–240

White J, Hill JH, Gray J, Tambe S, Gokhale AS, Schmidt DC (2009) Improving domain-specific language reuse with software product line techniques. IEEE Softw 26(4):47–53

Whitney D, Levi DM (2011) Visual crowding: a fundamental limit on conscious perception and object recognition. Trends Cognit Sci 15(4):160–168

Wilcoxon F (1945) Individual comparisons by ranking methods. Biom Bull 1(6):80–83

Wille D, Runge T, Seidl C, Schulze S (2017) Extractive software product line engineering using model-based delta module generation. In: Vamos. ACM, pp 36–43

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer Science & Business Media

Ziadi T, Jézéquel JM (2006) Software product line engineering with the uml: Deriving products. In: Software product lines. Springer, pp 557–588

Zschaler S, Sánchez P, Santos J, Alférez M, Rashid A, Fuentes L, Moreira A, Araújo J, Kulesza U (2009) Vml*—a family of languages for variability management in software product lines. In: International conference on software language engineering. Springer, pp 82–102

Affiliations

Wardah Mahmood[1] · Daniel Strüber[2] · Anthony Anjorin[3] · Thorsten Berger[1,4]

Daniel Strüber
d.strueber@cs.ru.nl

Anthony Anjorin
anthony.anjorin@iav.de

Thorsten Berger
thorsten.berger@rub.de

1 Chalmers | University of Gothenburg, Gothenburg, Sweden
2 Radboud University Nijmegen, Nijmegen, Netherlands
3 IAV Automotive Engineering, Berlin, Germany
4 Ruhr University Bochum, Bochum, Germany